Projet PiCoq
Deliverable D123
December 2014

De la KAM avec un Processus d'Ordre Supérieur

Damien Pous, Alan Schmitt

De la KAM avec un Processus d'Ordre Suprieur

D. Pous[1] & A. Schmitt[2]

1: CNRS, damien.pous@ens-lyon.fr
2: Inria, alan.schmitt@inria.fr

Résumé

Nous prsentons un encodage simple et direct de la machine abstraite de Krivine (KAM) dans le calcul de processus d'ordre suprieur HOcore, en utilisant un nombre trs restreint de canaux de communication. Cet encodage montre qu'il est possible de capturer l'expressivit du λ-calcul en HOcore ds que l'on fixe l'ordre d'valuation. Nous donnons galement une nouvelle borne infrieure pour le nombre minimal de restrictions ncessaire pour rendre l'quivalence de programmes dans HOcore indcidable. [1]

1. Introduction

Le calcul de processus HOcore est remarquable par sa similarit au λ-calcul, auquel il ajoute une notion de concurrence. Malgr cette similarit, aucun encodage du λ-calcul en HOcore n'a t prsent ce jour. En effet, l'appariement entre une fonction et son argument est trs syntaxique et trs rigide en λ-calcul, alors qu'il est beaucoup plus lche en HOcore car il correspond la prsence simultane sur le mme nom de canal d'un message et d'un rcepteur pour ce message. Cette diffrence cruciale, couple l'impossibilit de gnrer de nouveaux noms de canaux, implique que toute traduction doit fixer le nombre de redex pouvant tre activs en parallle. Ce nombre pouvant tre non-born pour certains λ-termes, ceci empche toute traduction *tant que la stratgie d'valuation n'a pas t fixe*. C'est cette deuxime voie que nous explorons ici, en choisissant une stratgie en appel par nom, dcline sous la forme d'une machine abstraite de Krivine (KAM).

Une fois ce choix de conception arrt, la traduction est trs naturelle : on reprsente la structure rcursive de la pile de la KAM comme deux messages transportant respectivement le terme de tte et, rcursivement, la queue de la pile. La β-rduction est simplement la communication entre le terme actif, reprsentant la fonction, avec la tte de la pile. De manire surprenante, cette traduction permet galement de capturer l'oprateur de contrle call-cc de la KAM. La rification de la continuation en tant que pile contenue dans un message permet en effet de facilement la dupliquer ou la remplacer.

Les leons que l'on peut tirer de cet encodage sont doubles. Tout d'abord, en ce qui concerne l'expressivit de HOcore, il montre comment on peut s'appuyer sur l'ordre suprieur pour atteindre un calcul Turing complet. Les travaux prcdents [5] tudiaient l'expressivit en se basant sur des machines de Minsky, qui n'ont besoin que de savoir compter et de dtecter qu'un compteur vaut 0. L'ordre suprieur n'est pas ncessaire pour compter, il est en revanche utilis pour dtecter l'galit 0 (voir [3] et [1] pour des versions de calculs sans ordre suprieur utilisant d'autres fonctionnalits pour dtecter le 0). La seconde leon porte sur la dcidabilit de la congruence barbue. En effet, nous avons montr prcdemment que la congruence barbue est dcidable pour HOcore [5, 2] si aucun nom de canal n'est cach, et cette congruence devient indcidable si quatre noms de canaux sont cachs. Notre traduction n'ayant besoin que de deux noms de canaux, il permet d'affiner le nombre minimal de restrictions globales ncessaire pour rendre la congruence barbue forte indcidable : il en suffit de deux.

1. Ce travail a bnfici du soutien de l'Agence Nationale pour la Recherche dans le cadre du projet PiCoq ANR 10 BLAN 0305.

Le reste de ce papier est organis comme suit. Nous prsentons le calcul HOcore en section 2, et la KAM en section 3. La traduction et sa preuve de correspondance oprationnelle est prsente en section 4 avant de conclure en section 5.

2. Prsentation de HOcore

2.1. Syntaxe

Le calcul HOcore [5] peut tre vu comme une restriction du π-calcul d'ordre suprieur [7], auquel on aurait enlev l'oprateur de restriction de nom. Il peut galement tre vu comme un λ-calcul parallle, o l'application d'une fonction $\lambda x.Q$ un argument P est remplace par une communication sur un canal a entre un envoi de message $\overline{a}\langle P\rangle$ mis en parallle d'une rception de message $a(x).Q$.

La syntaxe de HOcore est la suivante.

$$P ::= a(x).P \;\Big|\; \overline{a}\langle P\rangle \;\Big|\; P \parallel P \;\Big|\; x \;\Big|\; \mathbf{0}$$

Un processus P peut soit tre une rception de message sur un certain canal, note $a(x).P$, soit une mission de message, note $\overline{a}\langle P\rangle$, soit la mise en parallle de processus $P \parallel Q$, soit une variable x, soit le processus inactif $\mathbf{0}$. Nous distinguons les *variables* x, y, z des *noms de canaux* a, b, c.

Intuitivement, l'unique rgle de rduction est la rgle de communication suivante, o l'opration $[P/x]Q$ est la *substitution* de la variable x par le processus P dans Q. Le processus P, mis sur le canal a, est transmis au prfixe de rception $a(x).Q$.

$$\overline{a}\langle P\rangle \parallel a(x).Q \longrightarrow [P\,/\,x]Q \qquad (\dagger)$$

Notons que le calcul est asynchrone : l'mission de message n'a pas de continuation. Dans un calcul synchrone, l'envoi de message est de la forme $\overline{a}\langle P\rangle.Q$, o le processus Q est la continuation dmarre aprs l'mission du message sur a. Nous montrerons dans la section 4.3 que les messages synchrones permettent un encodage de la KAM ne ncessitant qu'un seul nom de canal.

Variables Une variable x est dite *lie* si elle est sous la porte d'un lieur pour cette variable (une rception de message $a(x).P$), *libre* sinon. Par exemple, dans le processus $a(x).(P \parallel y)$ avec $x \neq y$, les occurrences de x dans P sont lies, mais y est libre.

La machine de Krivine pour le λ-calcul en appel par nom [4] permet de ne considrer que des termes clos, et d'viter les difficults usuelles dues la capture de variables libres. Il en est de mme avec l'encodage que nous prsentons ici : nous ne manipulerons que des termes dont toutes les variables seront lies. Les noms de canaux sont eux tous libres, HOcore n'ayant pas d'oprateur de restriction.

2.2. Smantique

La smantique de HOcore est dfinie par un systme de transitions tiquetes (LTS). Les tiquettes sont dfinies par la syntaxe suivante :

$$\alpha ::= \overline{a}\langle P\rangle \mid a(P) \mid \tau \ .$$

Elles correspondent respectivement l'mission d'un processus, la rception d'un processus, et une communication interne. Le LTS est dfini inductivement, par les rgles suivantes.

$$\frac{}{\overline{a}\langle P\rangle \xrightarrow{\overline{a}\langle P\rangle} \mathbf{0}}\ \textsc{Out} \qquad \frac{}{a(x).Q \xrightarrow{a(P)} [P\,/\,x]Q}\ \textsc{Inp}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q}\ \textsc{Par1} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'}\ \textsc{Par2}$$

$$\frac{P \xrightarrow{\overline{a}\langle R\rangle} P' \qquad Q \xrightarrow{a(R)} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}\ \textsc{Tau1} \qquad \frac{P \xrightarrow{a(R)} P' \qquad Q \xrightarrow{\overline{a}\langle R\rangle} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}\ \textsc{Tau2}$$

Congruence structurelle Notons que le LTS prcdent est trs rigide : il conserve strictement la structure des termes (l'inverse, une smantique rductionelle utilise gnralement une notion de congruence structurelle, qui permet par exemple de rorganiser les parenthses modulo associativit de la mise en parallle). L'avantage est que les termes obtenus aprs une rduction sont plus facile analyser ; l'inconvnient est que ce LTS tend gnrer de nombreuses occurrences du processus $\mathbf{0}$, qui sont inutiles. Par exemple, on a formellement les transitions suivantes :

$$(\overline{a}\langle P\rangle \parallel \overline{b}\langle Q\rangle) \parallel a(x).b(y).(x \parallel y) \xrightarrow{\tau} (\mathbf{0} \parallel \overline{b}\langle Q\rangle) \parallel b(y).(P \parallel y) \xrightarrow{\tau} (\mathbf{0} \parallel \mathbf{0}) \parallel (P \parallel Q)$$

On va donc s'appuyer sur une notion de congruence structurelle trs restreinte qui permet de s'affranchir de ces occurrences de $\mathbf{0}$.

Une relation $\mathcal{R}$ est une *congruence* si c'est une relation d'quivalence (rflexive, symtrique, transitive) qui respecte les diffrents constructeurs du langage (par exemple si $P\ \mathcal{R}\ Q$ alors nous avons $a(x).P\ \mathcal{R}\ a(x).Q$). On quotiente dans la suite l'ensemble des processus par la plus petite congruence telle que la composition parallle admette le processus $\mathbf{0}$ comme lment neutre gauche et droite.

2.3. Expressivit

HOcore est qualifi de minimal, car il ne contient que le strict ncessaire l'ordre suprieur. Par exemple, il n'inclut pas d'oprateurs de restriction ou de rplication. HOcore est tout de mme Turing complet : un encodage fidle des machines de Minsky est prsent dans [5]. En particulier, le problme de la terminaison y est indcidable.

Un objectif de ce papier est de montrer qu'il est possible d'encoder directement des modles de calcul plus complexes, comme le λ-calcul, dans HOcore.

2.4. quivalence de processus

Une des questions cruciales de l'tude de calculs de processus est de savoir si deux processus « font la mme chose ». Ainsi, dans l'optique de la programmation modulaire, on doit tre capable de dire si deux bibliothques logicielles sont interchangeables. Deux processus sont quivalents si, dans n'importe quel contexte, ce que l'on observe de leur activit est similaire. Formellement, on dfinit la *congruence barbue* [6] comme la plus grande relation symtrique telle que :

- si $P \simeq Q$ et $P \xrightarrow{\tau} P'$, alors il existe un Q' tel que $Q \xrightarrow{\tau} Q'$ et $P' \simeq Q'$: la congruence barbue est prserve par rductions ;
- si $P \simeq Q$, alors $C[P] \simeq C[Q]$ pour tout contexte C, un contexte tant un processus avec un trou : la congruence barbue est une congruence ;

– si $P \simeq Q$ et $P \xrightarrow{\overline{a}\langle P''\rangle} P'$, alors il existe Q' et Q'' tels que $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$: la congruence barbue met en relation des processus avec les mmes *observables*, ou barbes, qui sont ici la possibilit d'mettre un message sur un canal donn.

On peut dfinir de faon similaire la *congruence barbue faible*, note $\cong$, en considrant dans la premiere clause des squences arbitraires de transitions, plutt que des transitions simples.

Un des rsultats fondamentaux de HOcore est la dcidabilit de la congruence barbue [5]. Les processus de HOcore ne pouvant pas cacher leurs rductions (le langage n'a pas d'oprateur de restriction), il est toujours possible de dfinir des contextes explorant la structure d'un processus. En revanche, ds que l'on autorise des rductions anonymes, par exemple grce des restrictions globales empchant l'observation sur certains noms, la congruence barbue devient indcidable. Les travaux prcdents ont montr que quatre telles restrictions globales suffisent.

3. La KAM

La machine abstraite de Krivine est une machine trs simple pour valuer les termes du λ-calcul en appel par nom. Elle permet galement de dfinir des oprateurs de contrle.

Une configuration de la KAM est compos d'un terme du λ-calcul et d'une pile, qui est une liste de λ-termes. Comme indiqu plus haut, tous les λ-termes considrs sont *clos* (ils ne contiennent pas de variable libre).

$$C ::= M \star \pi$$
$$M ::= x \mid MN \mid \lambda x.M$$
$$\pi ::= M :: \pi \mid []$$

Les rgles de rductions de la KAM (sans oprateur de contrle) sont les suivantes ; un calcul s'arrte quand un tat $\lambda x.M \star []$ est atteint.

$$MN \star \pi \mapsto M \star N :: \pi \qquad \text{(Push)}$$
$$\lambda x.M \star N :: \pi \mapsto [N\,/\,x]M \star \pi \qquad \text{(Grab)}$$

La KAM avec oprateurs de contrles ajoute deux constructions syntaxiques : l'oprateur de capture de continuation `cc`, utilisable dans les programmes, et les constantes de pile k_π, qui ne peuvent tre cres que par appel `cc`. Les deux rgles suivantes sont alors ajoutes.

$$\texttt{cc} \star M :: \pi \mapsto M \star k_\pi :: \pi \qquad \text{(CallCC)}$$
$$k_\pi \star M :: \pi' \mapsto M \star \pi \qquad \text{(Restore)}$$

4. KAM en HOcore

Nous commenons par traduire la KAM sans oprateurs de contrle ; nous tendons ensuite notre encodage ces oprateurs en section 4.2.

4.1. Version asynchrone

Nous prsentons un codage n'utilisant que deux noms libres. La pile courante est un message sur le nom c (“c” pour *continuation*). Le contenu de la pile est un message sur le nom a (“a” comme

argument), correspondant la tte de la pile, et un message sur c contenant la queue de la pile. La traduction de la pile vide est arbitrairement fixe au processus $\overline{b}\langle\mathbf{0}\rangle$, pour un troisime nom libre b permettant d'observer la fin du calcul (les rsultats dmontrs ci-dessous sont toujours valides lorsque ce processus est remplac par un processus arbitraire, tant que celui-ci n'introduit pas de divergence ou de non-dterminisme).

La traduction d'une pile $1 :: 2 :: 3 :: []$ prend donc la forme $\overline{a}\langle 1\rangle \parallel \overline{c}\langle\overline{a}\langle 2\rangle \parallel \overline{c}\langle\overline{a}\langle 3\rangle \parallel \overline{c}\langle\overline{b}\langle\mathbf{0}\rangle\rangle\rangle\rangle$.

Pour ne pas alourdir les notations, nous utilisons le mme symbole pour traduire des tats, des piles et des λ-termes. Nous supposons galement que les noms des variables lies u et s (u n'apparat que pour la traduction des oprateurs de contrle) sont diffrents des noms de variables du λ-terme.

$$
\begin{aligned}
[\![[]]\!] &\triangleq \overline{b}\langle\mathbf{0}\rangle \\
[\![M :: \pi]\!] &\triangleq \overline{a}\langle[\![M]\!]\rangle \parallel \overline{c}\langle[\![\pi]\!]\rangle \\
[\![MN]\!] &\triangleq c(s).([\![M]\!] \parallel \overline{c}\langle\overline{a}\langle[\![N]\!]\rangle \parallel \overline{c}\langle s\rangle\rangle) \\
[\![\lambda x.M]\!] &\triangleq c(s).(a(x).\,[\![M]\!] \parallel s) \\
[\![x]\!] &\triangleq x \\
[\![M \star \pi]\!] &\triangleq [\![M]\!] \parallel \overline{c}\langle[\![\pi]\!]\rangle
\end{aligned}
$$

Notons que dans la traduction d'une configuration, la pile, compose de messages imbriqus sur a et c, est elle mme encapsule l'intrieur d'un message sur le nom c.

Nous montrons ci-dessous que la rduction de $[\![M \star \pi]\!]$ est dterministe pour tout M et tout π. De plus, chaque tape de calcul de la KAM correspond une ou deux tapes de calcul du processus traduit (ou trois lorsque l'on prend en compte les oprateurs de contrle).

Lemme 1 (Substitution)**.** *Pour tous M, x, N avec N clos, nous avons $[\![[N\,/\,x]M]\!] = [[\![N]\!]\,/\,x]\,[\![M]\!]$.*

Démonstration. Par une simple induction sur M. □

Lemme 2 (Simulation)**.** *Pour tous M, M', π, π' tels que $M \star \pi \mapsto M' \star \pi'$, nous avons $[\![M \star \pi]\!] \xrightarrow{\tau}{}^{+} [\![M' \star \pi']\!]$.*

Démonstration. Il suffit de considrer les rgles de rduction de la machine de Krivine :

PUSH ($MN \star \pi \mapsto M \star N :: \pi$) : on vrifie que

$$
\begin{aligned}
[\![MN \star \pi]\!] &= (c(s).\,[\![M]\!] \parallel \overline{c}\langle\overline{a}\langle[\![N]\!]\rangle \parallel \overline{c}\langle s\rangle\rangle) \parallel \overline{c}\langle[\![\pi]\!]\rangle \\
&\xrightarrow{\tau} [\![M]\!] \parallel \overline{c}\langle\overline{a}\langle[\![N]\!]\rangle \parallel \overline{c}\langle[\![\pi]\!]\rangle\rangle \\
&= [\![M \star N :: \pi]\!] \quad .
\end{aligned}
$$

Il suffit donc d'une seule transition pour simuler cette rgle.

GRAB ($\lambda x.M \star N :: \pi \mapsto [N\,/\,x]M \star \pi$) : il faut cette fois deux transitions :

$$
\begin{aligned}
[\![\lambda x.M \star N :: \pi]\!] &= (c(s).(a(x).\,[\![M]\!]) \parallel s) \parallel \overline{c}\langle\overline{a}\langle[\![N]\!]\rangle \parallel \overline{c}\langle[\![\pi]\!]\rangle\rangle \\
&\xrightarrow{\tau} (a(x).\,[\![M]\!]) \parallel \overline{a}\langle[\![N]\!]\rangle \parallel \overline{c}\langle[\![\pi]\!]\rangle \\
&\xrightarrow{\tau} [[\![N]\!]\,/\,x]\,[\![M]\!] \parallel \overline{c}\langle[\![\pi]\!]\rangle \\
&= [\![[N\,/\,x]M]\!] \parallel \overline{c}\langle[\![\pi]\!]\rangle \qquad \text{(par le Lemme 1)} \\
&= [\![[N\,/\,x]M \star \pi]\!] \quad .
\end{aligned}
$$

□

Il nous faut maintenant prouver que les transitions des processus traduits sont dterministes, et qu'elles correspondent des rductions dans la KAM. Or, comme on le voit dans la preuve prcdente, certaines transitions sont intermdiaires et doivent tre compltes afin de correspondre prcisment une rduction de la KAM.

Afin de simplifier la preuve, nous passons par une machine abstraite lgrement diffrente de la KAM, dans laquelle certaines tapes sont artificiellement ddoubles : les tapes de calcul des processus traduits correspondent ainsi exactement aux rductions de la KAM modifie.

La modification est la suivante : on introduit une configuration intermdiaire, note $\lambda' x.M \star \pi$, et la rgle GRAB est ddouble comme suit :

$$\lambda x.M \star \pi \mapsto \lambda' x.M \star \pi \qquad (\text{GRAB}_1)$$

$$\lambda' x.M \star N :: \pi \mapsto [N \,/\, x]M \star \pi \qquad (\text{GRAB}_2)$$

Fait 3. *Une configuration admet une squence infinie de rductions dans la KAM originelle si et seulement elle admet une squence infinie de rductions dans la KAM modifie.*

En accord avec le second cas dans la preuve du lemme 2, la fonction de traduction est tendue aux configurations intermdiaires en posant :

$$\llbracket \lambda' x.M \star \pi \rrbracket \stackrel{\triangle}{=} (a(x).\llbracket M \rrbracket) \parallel \llbracket \pi \rrbracket \quad .$$

Lemme 4. *La fonction de traduction est injective.*

Démonstration. On prouve qu'elle est injective sur les λ-termes, puis sur les piles, puis sur les configurations. □

Notons $\llbracket \cdot \rrbracket^{-1}$ la fonction partielle inverse de la traduction, i.e., telle que $\llbracket \llbracket C \rrbracket \rrbracket^{-1} = C$ pour toute configuration C de la KAM modifie. On se convainc aisment que cette fonction est calculable.

Lemme 5 (Rflection)**.** *Pour toute configuration C et processus P, si $\llbracket C \rrbracket \xrightarrow{\tau} P$, alors $\llbracket P \rrbracket^{-1}$ est dfini et $C \mapsto \llbracket P \rrbracket^{-1}$.*

Démonstration. On raisonne par cas sur la configuration C :

- $C = MN \star \pi$: on a $\llbracket C \rrbracket = (c(s).\llbracket M \rrbracket \parallel \overline{c}\langle \overline{a}\langle \llbracket N \rrbracket \rangle \parallel \overline{c}\langle s \rangle \rangle) \parallel \overline{c}\langle \llbracket \pi \rrbracket \rangle$, d'o $P = \llbracket M \rrbracket \parallel \overline{c}\langle \overline{a}\langle \llbracket N \rrbracket \rangle \parallel \overline{c}\langle \llbracket \pi \rrbracket \rangle \rangle$ puisqu'une seule transition est possible. On vrifie alors que $\llbracket P \rrbracket^{-1} = M \star N :: \pi$, et $C \mapsto \llbracket P \rrbracket^{-1}$ par la rgle (PUSH).
- $C = \lambda x.MN \star \pi$: on a $\llbracket C \rrbracket = (c(s).(a(x).\llbracket M \rrbracket) \parallel s) \parallel \overline{c}\langle \llbracket \pi \rrbracket \rangle$, d'o $P = (a(x).\llbracket M \rrbracket) \parallel \llbracket \pi \rrbracket$. On vrifie alors que $\llbracket P \rrbracket^{-1} = \lambda' x.M \star \pi$, et $C \mapsto \llbracket P \rrbracket^{-1}$ par la rgle (GRAB_1).
- $C = \lambda' x.MN \star \pi$: si la pile π est vide, alors $\llbracket C \rrbracket = (a(x).\llbracket M \rrbracket) \parallel \overline{b}\langle \mathbf{0} \rangle$ n'admet pas de transition τ, ce qui contredit l'hypothse sur P. On a donc $\pi = N :: \pi'$, et $\llbracket C \rrbracket = (a(x).\llbracket M \rrbracket) \parallel \overline{a}\langle N \rangle \parallel \overline{c}\langle \llbracket \pi' \rrbracket \rangle$. On a ncessairement $P = [\llbracket N \rrbracket / x]\,\llbracket M \rrbracket \parallel \overline{c}\langle \llbracket \pi' \rrbracket \rangle$; on vrifie alors que $\llbracket P \rrbracket^{-1} = [N/x]M \star \pi'$ l'aide du Lemme 1, et que $C \mapsto \llbracket P \rrbracket^{-1}$ par la rgle (GRAB_2). □

Thorme 6 (Dterminisme)**.** *Pour toute configuration C et tous processus P, P', P'', si $\llbracket C \rrbracket \xrightarrow{\tau}{}^{\star} P$, $P \xrightarrow{\tau} P'$ et $P \xrightarrow{\tau} P''$, alors $P' = P''$.*

Démonstration. Par le Lemme 5, on peut se ramener au cas o $P = \llbracket C \rrbracket$. En reprenant la preuve de ce mme lemme, par analyse de cas sur C, on constate qu'au plus une communication est possible dans le processus traduit $\llbracket C \rrbracket$. □

Thorme 7 (Correspondance oprationelle)**.** *Pour toutes configurations C, C', $C \mapsto C'$ si et seulement si $[\![C]\!] \xrightarrow{\tau} [\![C']\!]$.*

Démonstration. Consquence immdiate des Lemmes 2 et 5. □

Thorme 8. *Pour toute configuration C, nous avons C termine si et seulement si $[\![C]\!] \xrightarrow{\tau}{}^{\star} \xrightarrow{\overline{b}\langle \mathbf{0} \rangle}$.*

Démonstration. La machine abstraite s'arrte exactement sur les configurations de la forme $\lambda' x.M \star []$, dont les encodages sont de la forme $(a(x).[\![M]\!]) \parallel \overline{b}\langle \mathbf{0} \rangle$, qui ont pas de transitions τ, et qui ont une barbe sur b. Inversement, les seules configurations dont la traduction est capable d'mettre sur b sont celles de la forme $\lambda' x.M \star []$. On peut donc conclure par le Thorme 7. □

En tant que fragment du π-calcul d'ordre suprieur [7], HOcore peut naturellement tre tendu en ajoutant un oprateur de restriction de nom. Nous ne considrons ici qu'une extension plus rduite n'utilisant que des restrictions globales (i.e., les restrictions ne peuvent tre dans des messages, donc elles ne peuvent tre rpliques). La syntaxe est tendue de la manire suivante.

$$T ::= \nu a.T \mid P$$

L'extension du LTS est immdiate : les seules transitions autorises pour les termes $\nu a.T$ sont celles de T dont les noms ne mentionnent pas a. Nous notons $\mathtt{n}(\alpha)$ les noms de canaux de α.

$$\frac{T \xrightarrow{\alpha} T' \qquad a \notin \mathtt{n}(\alpha)}{\nu a.T \xrightarrow{\alpha} \nu a.T'}$$

Intuitivement, $\nu a.P$ correspond au processus P auquel on interdit de communiquer sur a avec l'extrieur (en mission comme en rception) : le nom de canal a est connu de lui seul, toutes les communications sur a ne peuvent donc avoir lieu qu' l'intrieur de P.

Thorme 9. *Soit $\Omega \triangleq \nu a.\overline{a}\langle a(x).(\overline{a}\langle x \rangle \parallel x) \rangle \parallel a(x).(\overline{a}\langle x \rangle \parallel x)$. Pour tout λ-terme M et toute pile π, on a $\nu a.\nu c.[\![M \star \pi]\!] \simeq \Omega$ si et seulement si $M \star \pi$ ne termine pas.*

Démonstration. Ω est un processus divergent, dont la seule transition est $\Omega \xrightarrow{\tau} \Omega$. Par consquent, si $M \star \pi$ ne termine pas, alors $\nu a.\nu c.[\![M \star \pi]\!]$ lui est quivalent, puisqu'il ne pourra jamais mettre sur son seul nom libre (b). Inversement, si $M \star \pi$ termine, alors $\nu a.\nu c.[\![M \star \pi]\!]$ finira par mettre sur b, ce qui le distingue du processus Ω. □

Corollaire 10. *L'quivalence contextuelle est indcidable dans HOcore avec deux restrictions globales.*

Un raisonnement similaire permet d'obtenir l'indcidabilit de l'quivalence contextuelle faible dans HOcore avec deux restrictions globales ; on peut mme utiliser dans ce cas le processus vide, $\mathbf{0}$, plutt que le processus divergent Ω (notons cependant que pour cette preuve, on n'a pas la possibilit d'encoder la pile vide par un processus arbitraire ; en particulier, le processus vide ne conviendrait pas : il est ncessaire d'avoir une observable visible lorsque le fond de pile est atteint). En contrepartie, dans le cas fort, le fond de pile peut tre traduit par $\mathbf{0}$ en effectuant la comparaison avec Ω.

4.2. Oprateurs de contrle

Nous ajoutons maintenant les oprateurs de contrle (call-cc). Rappelons les deux rgles de rduction de la KAM dfinissant ces oprateurs :

$$\mathtt{cc} \star M :: \pi \mapsto M \star k_\pi :: \pi \qquad \text{(CallCC)}$$
$$k_\pi \star M :: \pi' \mapsto M \star \pi \qquad \text{(Restore)}$$

Etant donn un processus P, on dfinit

$$K(P) \triangleq c(s_0).(s_0 \parallel a(u).c(\_).(u \parallel \overline{c}\langle P\rangle)).$$

Les deux oprateurs sont alors traduits comme suit :

$$\begin{aligned}[\![\mathtt{cc}]\!] &\triangleq c(s_0).(s_0 \parallel c(s).a(u).(u \parallel \overline{c}\langle\overline{a}\langle K(s)\rangle \parallel \overline{c}\langle s\rangle\rangle))\\ [\![k_\pi]\!] &\triangleq K([\![\pi]\!])\end{aligned}$$

Cet encodage ncessite trois transitions pour simuler les rgles (CallCC) et (Restore) de la KAM avec oprateurs de contrle. Comme prcdemment pour la rgle (Grab), on introduit donc quatre configurations intermdiaires et artificielles dans la KAM, dont la smantique est dfinie par les six rgles suivantes.

$$\mathtt{cc} \star \pi \mapsto \mathtt{cc}_1 \star \pi \qquad (\text{CallCC}_1)$$
$$\mathtt{cc}_1 \star M :: \pi \mapsto \mathtt{cc}_2 \star M :: \pi \qquad (\text{CallCC}_2)$$
$$\mathtt{cc}_2 \star M :: \pi \mapsto M \star k_\pi :: \pi \qquad (\text{CallCC}_3)$$

$$k_\pi \star \pi' \mapsto k_\pi^1 \star \pi' \qquad (\text{Restore}_1)$$
$$k_\pi^1 \star M :: \pi' \mapsto k_\pi^2 \star M :: \pi' \qquad (\text{Restore}_2)$$
$$k_\pi^2 \star M :: \pi' \mapsto M \star \pi \qquad (\text{Restore}_3)$$

Ces quatre configurations s'encodent comme suit dans HOcore.

$$\begin{aligned}[\![\mathtt{cc}_1 \star \pi]\!] &\triangleq [\![\pi]\!] \parallel c(s).a(u).(u \parallel \overline{c}\langle\overline{a}\langle K(s)\rangle \parallel \overline{c}\langle s\rangle\rangle)\\ [\![\mathtt{cc}_2 \star M :: \pi]\!] &\triangleq \overline{a}\langle[\![M]\!]\rangle \parallel a(u).(u \parallel \overline{c}\langle\overline{a}\langle K([\![\pi]\!])\rangle \parallel \overline{c}\langle[\![\pi]\!]\rangle\rangle)\\ &= \overline{a}\langle[\![M]\!]\rangle \parallel a(u).(u \parallel \overline{c}\langle\overline{a}\langle[\![k_\pi]\!]\rangle \parallel \overline{c}\langle[\![\pi]\!]\rangle\rangle)\\ &= \overline{a}\langle[\![M]\!]\rangle \parallel a(u).(u \parallel \overline{c}\langle[\![k_\pi :: \pi]\!]\rangle)\\ [\![k_\pi^1 \star \pi']\!] &\triangleq [\![\pi']\!] \parallel a(u).c(\_).(u \parallel \overline{c}\langle[\![\pi]\!]\rangle)\\ [\![k_\pi^2 \star M :: \pi']\!] &\triangleq \overline{c}\langle[\![\pi']\!]\rangle \parallel c(\_).([\![M]\!] \parallel \overline{c}\langle[\![\pi]\!]\rangle)\\ &= \overline{c}\langle[\![\pi']\!]\rangle \parallel c(\_).\,[\![M \star \pi]\!]\end{aligned}$$

On vrifie alors que les processus traduits correspondants ont exactement les transitions suivantes.

$$[\![\mathtt{cc} \star M :: \pi]\!] \xrightarrow{\tau} [\![\mathtt{cc}_1 \star M :: \pi]\!] \xrightarrow{\tau} [\![\mathtt{cc}_2 \star M :: \pi]\!] \xrightarrow{\tau} [\![M \star k_\pi :: \pi]\!]$$
$$[\![k_\pi \star M :: \pi']\!] \xrightarrow{\tau} [\![k_\pi^1 \star M :: \pi']\!] \xrightarrow{\tau} [\![k_\pi^2 \star M :: \pi']\!] \xrightarrow{\tau} [\![M \star \pi]\!]$$

Les preuves des lemmes 1, 2, 4 et 5 ainsi que des thormes 6 et 7 s'tendent sans difficult.

Le lecteur attentif aura remarqu que la traduction de l'oprateur cc effectue d'abord une rception $c(s)$ avant la rception $a(u)$. Ce choix est purement esthtique : changer l'ordre des rceptions est tout fait possible, mais ne permet pas d'utiliser $[\![k_\pi :: \pi]\!]$ dans la traduction de $\mathtt{cc}_2$, la rendant moins succincte.

4.3. Version synchrone

Nous montrons maintenant qu'il est possible d'obtenir une traduction de la KAM en utilisant un seul nom, si le calcul considr est *synchrone*. Une version synchrone de HOcore ne modifie que l'mission de message, remplaant la construction $\overline{a}\langle P\rangle$ par $\overline{a}\langle P\rangle.Q$. Cette dernire met toujours un message sur a

transportant P. En revanche, ds que le message est reu, le processus Q, appel *continuation*, est lanc : la rgle de rduction intuitive devient

$$\overline{a}\langle P\rangle.Q \parallel a(x).R \longrightarrow Q \parallel [P\,/\,x]R \qquad (\ddagger)$$

Formellement, cela se traduit dans le LTS par une simple modification de la rgle axiome Out :

$$\frac{}{\overline{a}\langle P\rangle.Q \xrightarrow{\overline{a}\langle P\rangle} Q}\ \textsc{Out}$$

La traduction de la KAM (tendue) est donne ci-dessous. Nous traduisons dsormais les piles comme tant des messages synchrones : le contenu du message tant la tte de la pile et la continuation du message la queue de la pile. Nous traduisons ainsi une pile $1 :: 2 :: 3 :: []$ par $\overline{a}\langle 1\rangle.\overline{a}\langle\overline{a}\langle 2\rangle.\overline{a}\langle\overline{a}\langle 3\rangle.\overline{a}\langle\overline{b}\langle\mathbf{0}\rangle\rangle\rangle\rangle$.

Au sein d'une configuration, comme dans le cas asynchrone, la pile sera de plus encapsule l'intrieur d'un message additionel sur a.

Comme ci-dessus, pour tout processus P, nous dfinissons

$$K(P) \triangleq a(s_0).(s_0 \parallel a(u).a(\_).(u \parallel \overline{a}\langle P\rangle)).$$

(Par convention, nous notons $\overline{a}\langle P\rangle$ les messages $\overline{a}\langle P\rangle.\mathbf{0}$ dont la continuation est vide). Nous dfinissons alors la traduction synchrone comme suit.

$$\begin{aligned}
[\![[]]\!] &\triangleq \overline{b}\langle\mathbf{0}\rangle\\
[\![M :: \pi]\!] &\triangleq \overline{a}\langle[\![M]\!]\rangle.\overline{a}\langle[\![\pi]\!]\rangle\\
[\![MN]\!] &\triangleq a(s).([\![M]\!] \parallel \overline{a}\langle\overline{a}\langle[\![N]\!]\rangle.\overline{a}\langle s\rangle\rangle)\\
[\![\lambda x.M]\!] &\triangleq a(s).(a(x).\,[\![M]\!] \parallel s)\\
[\![x]\!] &\triangleq x\\
[\![M \star \pi]\!] &\triangleq [\![M]\!] \parallel \overline{a}\langle[\![\pi]\!]\rangle\\
[\![\mathtt{cc}]\!] &\triangleq a(s_0).(s_0 \parallel a(u).a(s).(u \parallel \overline{a}\langle\overline{a}\langle K(s)\rangle.\overline{a}\langle s\rangle\rangle))\\
[\![k_\pi]\!] &\triangleq K([\![\pi]\!])
\end{aligned}$$

A nouveau, les preuves de la section 4.1 s'adaptent sans difficult. L'quivalence contextuelle est donc indcidable dans HOcore synchrone avec une seule restriction de nom globale.

5. Conclusion

Nous avons prsent deux traductions directes de la machine abstraite de Krivine avec oprateurs de contrle dans HOcore, utilisant un nom libre dans le cas synchrone, et deux dans le cas asynchrone. Cela nous a permis d'affiner la borne sur le nombre de restrictions globales de noms de canaux suffisant pour obtenir l'indcidabilit des quivalences contextuelles fortes et faible (en l'absence de restriction de nom, l'quivalence contextuelle forte est dcidable que le calcul soit synchrone ou asynchrone [5]—le cas faible est ouvert).

Nous pouvons remarquer que les trois fonctionnalits fondamentales utilises pour traduire la KAM sont les suivantes. Les deux premires portent sur l'ordre suprieur : les messages transportent des processus, et la rception effectue une substitution identique celle du λ-calcul. La troisime fonctionnalit sur laquelle nous nous appuyons porte sur le contrle de l'valuation : nous devons distinguer entre la tte de la pile et le reste de la pile. Pour ce faire, nous pouvons utiliser deux noms diffrents (version asynchrone), ou squentialiser les missions (version synchrone) et n'utiliser ainsi qu'un seul nom. Nous

ne pensons pas qu'il soit possible d'obtenir une traduction de la KAM avec un seul nom dans un calcul asynchrone.

La relative simplicit de notre traduction, et le fait qu'elle permette de traiter immdiatement les oprateurs de contrle, nous laisse esprer qu'elle permettra une meilleure comprhension de l'expressivit du calcul HOcore, et peut-tre, terme, de rsoudre la question de la dcidabilit de l'quivalence contextuelle faible—dans le calcul sans restriction de nom.

Références

[1] J. Aranda, F. D. Valencia, and C. Versari. On the expressive power of restriction and priorities in ccs with replication. In L. Alfaro, editor, *Foundations of Software Science and Computational Structures*, volume 5504 of *Lecture Notes in Computer Science*, pages 242–256. Springer Berlin Heidelberg, 2009.

[2] S. Boulier and A. Schmitt. Formalisation de hocore en coq. In *Actes des 23èmes Journées Francophones des Langages Applicatifs*, Jan. 2012.

[3] N. Busi, M. Gabbrielli, and G. Zavattaro. On the expressive power of recursion, replication and iteration in process calculi. *Mathematical Structures in Computer Science*, 19(6) :1191–1222, Dec. 2009.

[4] J.-L. Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3) :199–207, 2007.

[5] I. Lanese, J. A. Pérez, D. Sangiorgi, and A. Schmitt. On the expressiveness and decidability of higher-order process calculi. *Information and Computation*, 209(2) :198–226, Feb. 2011. Extended abstract presented at *Logic in Computer Science (LICS)*, 2008.

[6] R. Milner and D. Sangiorgi. Barbed bisimulation. In *19th ICALP*, volume 623 of *LNCS*, pages 685–695. Springer Verlag, 1992.

[7] D. Sangiorgi and D. Walker. *The π-calculus : a Theory of Mobile Processes*. Cambridge University Press, 2001.

Components as Location Graphs

Jean-Bernard Stefani

Components as Location Graphs

Jean-Bernard Stefani

INRIA

Abstract. This paper presents a process calculus framework for modeling ubiquitous computing systems and dynamic component-based structures as location graphs. A key aspect of the framework is its ability to model nested locations with sharing, while allowing the dynamic reconfiguration of the location graph, and the dynamic update of located processes.

1 Introduction

Motivations. Computing systems are increasingly built as distributed, dynamic assemblages of hardware and software components. Modelling these assemblages requires capturing different kinds of dependencies and containment relationships between components. The software engineering literature is rife with analyses of different forms of whole-part, aggregation or composition relationships, and of their attendants characteristics such as emergent property, overlapping lifetimes, and existential dependency [2]. These analyses explicitly consider the possibility for a component to be shared at once between different wholes, an important requirement in particular if one expects to deal with multiple architectural views of a system.

Consider, for instance, a software system featuring a database DB and a client of the database C. The database comprises the following (sub)components: a cache CC, a data store DS and a query engine QE. Both data store and query engine reside in the same virtual machine V_0, for performance reasons. Client and cache reside in another virtual machine V_1, also for performance reasons. We have here a description which combines two architectural views, in the sense of [17]: a *logical* view, that identifies two high-level components, the database DB and its client C, and the sub-components CC, QE and DS of the database, and a *process* view, that maps the above components on virtual machines V_0 and V_1. We also have two distinct containment or whole-part relationships: being placed in a virtual machine, and being part of the DB database. A virtual machine is clearly a container: it represents a set of resources dedicated to the execution of the components it contains, and it manifests a failure dependency for all the components it executes (should a virtual machine fail, the components it contains also fail). The database DB is clearly a composite: it represents the result of the composition of its parts (cache, query engine, and data store) together with their attendant connections and interaction protocols; it encapsulates the behavior of its subcomponents; and its lifetime constrains those of its parts (e.g. if the

database is destroyed so are its subcomponents). The cache component CC in this example is thus part of two wholes, the database DB and the virtual machine V_1.

Surprisingly, most formal models of computation and software architecture do not provide support for a direct modelling of containment structures with sharing. On the one hand, one finds numerous formal models of computation and of component software with strictly hierarchic structures, such as Mobile Ambients and their different variants [7, 6], the Kell calculus [25], BIP [4], Ptolemy [26], or, at more abstract level, Milner's bigraphs [20]. In bigraphs, for instance, it would be natural to model the containment relationships in our database example as instances of sub-node relationships in bigraphs, because nodes correspond to agents in a bigraph. Yet this is not possible because the sub-node relation in a bigraph is restricted to form a forest. To model the above example as a bigraph would require choosing which containment relation (placement in a virtual machine or being a subcomponent of the database) to represent by means of the sub-node relation, and to model the other relation by means of bigraph edges. This asymetry in modelling is hard to justify for both containment relations are proper examples of whole-part relationships.

On the other hand, one finds formal component models such as Reo [1], π-ADL [22], Synchronized Hyperedge Replacement (SHR) systems [14], SRM-Light [15], that represent only interaction structures among components, and not containment relationships, and models that support the modeling of non-hierarchical containment structures, but with other limitations. Our own work on the Kell calculus with sharing [16] allows to model non-hierarchical containment structures but places constraints on the dependencies that can be modelled. For instance, the lifetime dependency constraints associated with the virtual machines and the database in our example above (if the aggregate or composite dies so do its sub-components) cannot be both easily modeled. The reason is that the calculus still enforces an ownership tree between components for the purpose of passivation: components can only passivate components lower down in the tree (i.e. suspend their execution and capture their state). The formal model which comes closer to supporting non strictly hierarchical containment structures is probably CommUnity [28], where component containment is modelled as a form of superposition, and can be organized as an arbitrary graph. However, in CommUnity, possible reconfigurations in a component assemblage are described as graph transformation rules that are separate from the behavior of components, making it difficult to model reconfigurations initiated by the component assemblage itself.

To sum up, we are missing a model of computation that allows us to directly model both different forms of interactions and different forms of containment relationships between components; that supports both planned (i.e. built in component behaviors) and unplanned (i.e. induced by the environment) dynamic changes to these relationships, as well as to component behaviors.

Contribution. In this paper, we introduce a model of computation, called *G-Kells*, which meets these requirements. We develop our model at a more concrete

level than bigraph theory, but we abstract from linguistic details by developing a process calculus framework parameterized by a notion of process and certain semantical operations. Computation in our model is carried out by located processes, i.e. processes that execute at named *locations*. Locations can be nested inside one another, and a given location can be nested inside one or more locations at once. Locations constitute scopes for interactions: a set of processes can interact when situated in locations nested within the same location. Behaviors of located processes encompass interaction with other processes as well as reconfiguration actions which may change the structure of the location graph and update located processes. In addition, the G-Kells framework supports a notion of dynamic priority that few other component models support, apart from those targeting real-time and reactive systems such as BIP and Ptolemy.

Outline. The paper is organized as follows. The framework we introduce can be understood as an outgrowth of our prior work with C. Di Giusto [13], in which we proposed a process calculus interpretation of the BIP model. We recall briefly in Section 2 the main elements of this work, to help explain the extensions we introduce in our G-Kells framework. Section 3 presents the G-Kells framework and its formal operational semantics. Section 4 discusses the various features of the G-Kells framework, and related work. Section 5 concludes the paper.

2 CAB: a process calculus interpretation of BIP

One way to understand the G-Kells model we introduce in the next section, is to see it as a higher-order, dynamic extension of the CAB model [13], a process calculus interpretation of the BIP model. We recall briefly in this section the main elements of CAB.

The CAB model captures the key features of the BIP model: (i) hierarchical components; (ii) composition of components via explicit "glues" enforcing multiway synchronization constraints between sub-components; (iii) priority constraints regulating interactions among components. Just as the BIP model, the CAB model is parameterized by a family $\mathcal{P}$ of primitive behaviors. A CAB component, named l, can be either a primitive component $C = l[P]$, where P is taken from $\mathcal{P}$, or a composite component $C = l[C_1, \ldots, C_n \star G]$, built by composing a set of CAB components $\{C_1, \ldots, C_n\}$ with a glue process G. When l is the name of a component C, we write $C.\mathtt{nm} = l$. We note $\mathcal{C}$ the set of CAB components, $\mathcal{N}_l$ the set of location names, and $\mathcal{N}_c$ the set of channel names. Behaviors in $\mathcal{P}$ are defined as labelled transition systems, with labels in $\mathcal{N}_c$.

In CAB, the language for glues G is a very simple language featuring:

- *Action prefix* $\xi.G$, where ξ is an action, and G a continuation glue (in contrast to BIP, glues in CAB can be stateful).
- *Parallel composition* $G_1 \mid G_2$, where G_1 and G_2 are glues. This operator can be interpreted as an *or* operator, that gives the choice of meeting the priority and synchronization constraints of G_1 or of G_2.
- *Recursion* $\mu X.\, G$, where X is a process variable, and G a glue.

Actions embody synchronization and priority constraints that apply to subcomponents in a composition. An action ξ consists of a triplet $\langle \pi \cdot a \cdot \sigma \rangle$, where π is a priority constraint, σ is a synchronization constraint, and a is a channel name, signalling a possibility of synchronization on channel a. Priority and synchronization constraints take the same form: $\{l_i : a_i \mid i \in I\}$, where l_i are location names, and a_i are channel names. A synchronization constraint $\sigma = \{l_i : a_i \mid i \in I\}$ requires each sub-component l_i to be ready to synchronize on channel a_i. Note that in a synchronization constraint $\sigma = \{l_i : a_i \mid i \in I\}$ we expect each l_i to appear only once, i.e. for all $i, j \in I$, if $i \neq j$ then $l_i \neq l_j$. A priority constraint $\pi = \{l_i : a_i \mid i \in I\}$ ensures each subcomponent named l_i is *not* ready to synchronize on channel a_i.

Example 1. A glue G of the form $\langle \{l : a\}; c; \{l_1 : a_1, l_2 : a_2\} \rangle.G'$ specifies a synchronization between two subcomponents named l_1 and l_2: if l_1 offers a synchronization on channel a_1, and l_2 offers a synchronization on a_2, then their composition with glue G offers a synchronization on c, provided that the subcomponent named l does not offer a synchronization on a. When the synchronization on a takes place, implying l_1 and l_2 have synchronized with composite on a_1 and a_2, respectively, a new glue G' is put in place to control the behavior of the composite.

Note that the same component l can appear in both the priority and the synchronization constraint of the same action ξ.

Example 2. An action of the form $\langle \{l : a\} \cdot c \cdot \{l : b, l' : b\} \rangle$ specifies that a synchronization on c is possible provided both subcomponents l and l' offer a synchronization on b, and component l does not offer a synchronization on a.

The operational semantics of the CAB model is defined as the labeled transition system whose transition relation, $\rightarrow \subseteq \mathcal{C} \times (\mathcal{N}_l \times \mathcal{N}_c) \times \mathcal{C}$, is defined by the inference rules in Figure 2, where we use the following notations:

- $\mathbf{C}$ denotes a finite (possibly empty) set of components
- $\mathbf{C}_\sigma$ denotes the set $\{C_i \mid i \in I\}$, i.e. the set of subcomponents involved in the multiway synchronization directed by the synchronization constraint σ in rule Comp. Likewise, ${\mathbf{C}'}_\sigma$ denotes the set $\{C'_i \mid i \in I\}$.
- $\mathbf{C} \models_p \{l_i : a_i \mid i \in I\}$ denotes the fact that $\mathbf{C}$ meets the priority constraint $\pi = \{l_i : a_i \mid i \in I\}$, i.e. for all $i \in I$, there exists $C_i \in \mathbf{C}$ such that $C_i.\mathtt{nm} = l_i$ and $\neg(C_i \xrightarrow{l_i:a_i})$, meaning there are no C' such that $C_i \xrightarrow{l_i:a_i} C'$.

The Comp rule in Figure 2 relies on the transition relation between glues defined as the least relation verifying the rules in Figure 1.

The transition relation is well defined despite the presence of negative premises, for the set of rules in Figure 2 is stratified by the height of components, given by the function `height`, defined inductively as follows:

$$\mathtt{height}(l[P]) = 0 \qquad \mathtt{height}(l[\mathbf{C} \star G]) = 1 + \mathtt{max}\{\mathtt{height}(C) \mid C \in \mathbf{C}\}$$

Indeed, in rule Comp, if $\mathtt{height}(l[\mathbf{C} \star G]) = n$, then the components in $\mathbf{C}$ that appear in the premises of the rule have a maximum height of $n - 1$. The

$$\textsc{Act}\ \ \xi.G \xrightarrow{\xi} G \qquad\qquad \textsc{Rec}\ \ \frac{G\{^{\mu\, X.\, G}/_{X}\} \xrightarrow{\xi} G'}{\mu\, X.\, G \xrightarrow{\xi} G'}$$

$$\textsc{Parl}\ \ \frac{G \xrightarrow{\xi} G'}{G \mid G_2 \xrightarrow{\xi} G' \mid G_2} \qquad\qquad \textsc{Parr}\ \ \frac{G \xrightarrow{\xi} G'}{G_2 \mid G \xrightarrow{\xi} G_2 \mid G'}$$

Fig. 1. LTS semantics for CAB glues

$$\textsc{Prim}\ \ \frac{P \xrightarrow{\alpha} P'}{l[P] \xrightarrow{l:\alpha} l[P']}$$

$$\textsc{Comp}\ \ \frac{G \xrightarrow{\langle \pi \,\cdot\, a \,\cdot\, \sigma \rangle} G' \qquad \sigma = \{l_i : a_i \mid i \in I\} \qquad \forall i \in I,\ C_i \xrightarrow{l_i:a_i} C_i' \qquad \mathbf{C} \models_p \pi}{l[\mathbf{C} \star G] \xrightarrow{l:a} l[(\mathbf{C} \setminus \mathbf{C}_\sigma) \cup \mathbf{C'}_\sigma \star G']}$$

Fig. 2. LTS semantics for CAB($\mathcal{P}$)

transitions relation $\rightarrow$ is thus the transition relation associated with the rules in Figure 2 according to Definition 3.14 in [5], which is guaranteed to be a minimal and supported model of the rules in Figure 2 by Theorem 3.16 in [5].

We now give some intuition on the operational semantics of CAB. The evolution of a primitive component $C = l[P]$, is entirely determined by its primitive behavior P, following rule Prim. The evolution of a composite component $C = l[\mathbf{C}\star G]$ is directed by that of its glue G, which is given by rules Act, Parl, Parr and Rec. Note that the rules for glues do not encompass any synchronization between branches G_1 and G_2 of a parallel composition $G_1 \mid G_2$. Rule Comp specifies how glues direct the behavior of a composite (a form of superposition): if the glue G of the composite $l[\mathbf{C}\star G]$ offers action $\langle \pi \cdot a \cdot \sigma \rangle$, then the composite offers action $l : a$ if both the priority ($\mathbf{C} \models_p \pi$) and synchronization constraints are met. For the synchronization constraint $\sigma = \{l_i : a_i \mid i \in I\}$ to be met, there must exist subcomponents $\{C_i \mid i \in I\}$ ready to synchronize on channel a_i, i.e. such that we have, for each i, $C_i \xrightarrow{l_i:a_i} C_i'$, for some C_i'. The composite can then evolve by letting each each C_i perform its transition on channel a_i, and by letting untouched the components in $\mathbf{C}$ not involved in the synchronization (in the rule Comp, the components C_i in $\mathbf{C}$ are simply replaced by their continuation C_i' on the right hand side of the conclusion).

The CAB model is simple but already quite powerful. For instance, it was shown in [13] that CAB($\emptyset$), i.e. the instance of the CAB model with no primi-

tive components, is Turing complete[1]. Priorities are indispensable to the result, though: as shown in [13], CAB($\emptyset$) without priorities, i.e. where glue actions have empty priority constraints, can be encoded in Petri nets. We now turn to the G-Kells model itself.

3 G-Kells: a framework for location graphs

3.1 Syntax

The G-Kells process calculus framework preserves some basic features of the CAB model (named locations, actions with priority and synchronization constraints, multiway synchronization within a location) and extends it in several directions at once:

- First, we abstract away from the details of a glue language. We only require that a notion of process be defined by means of a particular kind of labelled transition system. The G-Kells framework will then be defined by a set of transition rules (the equivalent of Figure 2 for CAB) that takes as a parameter the transition relation for processes.
- We do away with the tree structure imposed by CAB for components. Instead, components will now form directed graphs between named locations, possibly representing different containment relationships among components.
- In addition, our location graphs are entirely dynamic, in the sense that they can evolve as side effects of process transitions taking place in nodes of the graphs, i.e. locations.
- CAB was essentially a pure synchronization calculus, with no values exchanged between components during synchronization. The G-Kells framework allows higher-order value passing between locations: values exchanged during synchronization can be arbitrary, including names and processes.

The syntax of G-Kells components is quite terse, and is given in Figure 3. The set of G-Kells components is called $\mathcal{K}$. Let us explain the different constructs:

- $\mathsf{0}$ stands for the null component, which does nothing.
- $l[P]$ is a *location* named l, which hosts a *process* P. As we will see below, a process P can engage in interactions with other processes hosted at other locations, but also modify the graph of locations in various ways.
- $l.r \multimap h$ denotes an *edge* in the location graph. An edge $l.r \multimap h$ connects the *role* r of a location l to another location h.
- $C_1 \parallel C_2$ stands for the parallel composition of components C_1 and C_2, which allows for the independent, as well as synchronized, evolution of C_1 and C_2.

[1] The CAB model is defined in [13] with an additional rule of evolution featuring silent actions. For simplicity, we have not included such a rule in our presentation here, but the stated results still stand for the CAB model presented in this paper.

$$C ::= \mathsf{0} \quad | \quad l[P] \quad | \quad l.r \twoheadrightarrow h \quad | \quad C \parallel C$$

$$l, h \in \mathcal{N}_l \quad r \in \mathcal{N}_r$$

Fig. 3. Syntax of G-Kells components

A role is just a point of attachment to nest a location inside another. A role r of a location l can be *bound*, meaning there exists an edge $l.r \twoheadrightarrow h$ attaching a location h to r, or unbound, meaning that there is no such edge. We say likewise that location h is bound to location l, or to a role in location l, if there exists an edge $l.r \twoheadrightarrow h$. Roles can be dynamically attached to a location, whether by the location itself or by another location. One way to understand roles is by considering a location $l[P]$ with unbound roles $r_1, \ldots, r_n$ as a frame for a composite component. To obtain the composite, one must complete the frame by binding all the unbound roles $r_1, \ldots, r_n$ to locations $l_1, \ldots, l_n$, which can be seen as subcomponents. Note that several roles of a given location can be bound to the same location, and that a location can execute with unbound roles.

Locations serve as scopes for interactions: as in CAB, interactions can only take place between a location and all the locations bound to its roles, and a location offers possible interactions as a result. Unlike bigraphs, all interactions are thus local to a given location. One can understand a location in two ways: either as a composite glue superposing, as in CAB, priority and synchronization constraints on the evolution of its subcomponents, i.e. the locations bound to it, or as a connector, providing an interaction conduit to the components it binds, i.e. the locations bound to it. More generally, one can understand intuitively a whole location graph as a component C, with unbound locations acting as external interfaces for accessing the services provided by C, locations bound to these interfaces corresponding to subcomponents of C, and unbound roles in the graph to possible places of attachment of missing subcomponents.

We do not have direct edges of the form $l \twoheadrightarrow h$ between locations to allow for processes hosted in a location, say l, to operate without knowledge of the names of locations bound to l through edges. This can be leveraged to ensure a process is kept isolated from its environment, as we discuss in Section 4. We maintain two invariants in G-Kells components: at any one point in time, for any location name l, there can be at most one location named l, and for any role r and location l, there can be at most one edge of the form $l.r \twoheadrightarrow k$.

Example 3. Let us consider how the example we discussed in the introduction can be modeled using G-Kells. Each of the different elements appearing in the configuration described (database DB, data store DS, query engine QE, cache CC, client C, virtual machines V_0 and V_1) can be modeled as locations, named accordingly. The database location has three roles s, q, c, and we have three edges $DB.s \twoheadrightarrow DS$, $DB.q \twoheadrightarrow QE$, $DB.c \twoheadrightarrow CC$, binding its three subcomponents DS, QE and CC. The virtual machines locations have two roles each, 0 and 1, and we have four edges $V_1.0 \twoheadrightarrow C$, $V_1.1 \twoheadrightarrow CC$,

$V_0.0 \twoheadrightarrow DS$, $V_0.1 \twoheadrightarrow QE$, manifesting the placement of components C, CC, DS, QE in virtual machines. Now, the database location hosts a process supporting the semantics of composition between its three subcomponents, e.g. the cache management protocol directing the interactions between the cache and the other two database subcomponents[2]. The virtual machine locations host processes supporting the failure semantics of a virtual machine, e.g. a crash failure semantics specifying that, should a virtual machine crash, the components it hosts (bound through roles 0 and 1) should crash similarly. We will see in Section 4 how this failure semantics can be captured.

3.2 Operational semantics

We now formally define the G-Kells process calculus framework operational semantics by means of a labelled transition system.

Names, values and environments

Notations. We use boldface to denote a finite set of elements of a given set. Thus if S is a set, and s a typical element of S, we write $\mathbf{s}$ to denote a finite set of elements of S, $\mathbf{s} \subseteq S$. We use ϵ to denote an empty set of elements. We write $\wp_f(S)$ for the set of finite subsets of a set S. If S_1, S_2, S are sets, we write $S_1 \uplus S_2 = S$ to mean $S_1 \cup S_2 = S$ and S_1, S_2 are disjoint, i.e. S_1 and S_2 form a partition of S. We sometimes write $s, \mathbf{s}$ to denote $\{s\} \cup \mathbf{s}$. If C is a G-Kell component, $\Gamma(C)$ represents the set of edges of C. Formally, $\Gamma(C)$ is defined by induction as follows : $\Gamma(\mathtt{0}) = \emptyset$, $\Gamma(l[P]) = \emptyset$, $\Gamma(l.r \twoheadrightarrow h) = \{l.r. \twoheadrightarrow h\}$, $\Gamma(C_1 \parallel C_2) = \Gamma(C_1) \cup \Gamma(C_2)$.

Names and Values. We use three disjoint, infinite, denumerable sets of names, namely the set $\mathcal{N}_c$ of channel names, the set $\mathcal{N}_l$ of location names, and the set $\mathcal{N}_r$ of role names. We set $\mathcal{N} = \mathcal{N}_c \cup \mathcal{N}_l \cup \mathcal{N}_r$. We note $\mathcal{P}$ the set of processes. We note $\mathcal{V}$ the set of values. We posit the existence of three functions $\mathtt{fcn} : \mathcal{V} \to \mathcal{N}_c$, $\mathtt{fln} : \mathcal{V} \to \mathcal{N}_l$ $\mathtt{frn} : \mathcal{V} \to \mathcal{N}_r$ that return, respectively, the set of free channel names, free location names, and free role names occurring in a given value. The restriction of $\mathtt{fcn}$ (resp. $\mathtt{fln}, \mathtt{frn}$) to $\mathcal{N}_c$ (resp. $\mathcal{N}_l, \mathcal{N}_r$) is defined to be the identity on $\mathcal{N}_c$ (resp. $\mathcal{N}_l, \mathcal{N}_r$). The function $\mathtt{fn} : \mathcal{V} \to \mathcal{N}$, that returns the set of free names of a given value, is defined by $\mathtt{fn}(V) = \mathtt{fcn}(V) \cup \mathtt{fln}(V) \cup \mathtt{frn}(V)$. The sets $\mathcal{N}$, $\mathcal{P}$ and $\mathcal{V}$, together with the functions $\mathtt{fcn}$, $\mathtt{fln}$, and $\mathtt{frn}$, are parameters of the G-Kells framework. We denote by $\mathcal{E}$ the set of edges, i.e. the set of triples $l.r \twoheadrightarrow h$ of $\mathcal{N}_l \times \mathcal{N}_r \times \mathcal{N}_l$. We stipulate that names, processes, edges and finite sets of edges are values: $\mathcal{N} \cup \mathcal{P} \cup \mathcal{E} \cup \wp_f(\mathcal{E}) \subseteq \mathcal{V}$. We require the existence of a relation $\mathtt{match} \subseteq \mathcal{V}^2$, used in ascertaining possible synchronization.

[2] Notice that the database location does not run inside any virtual machine. This means that, at this level of abstraction, our *process* architectural view of the database composite is similar to a network connecting the components placed in the two virtual machines.

Environments. Our operational semantics uses a notion of *environment*. An environment Γ is just a subset of $\mathcal{N} \cup \mathcal{E}$, i.e. a set of names and a set of edges. The set of names in an environment represents the set of *already used* names in a given context. The set of edges in an environment represents the set of edges of a location graph. the set of environments is noted $\mathcal{G}$.

Processes

Process transitions. We require the set of processes to be equipped with a transition system semantics given by a labelled transition system where transitions are of the following form:

$$P \xrightarrow{\langle \pi \cdot \alpha \cdot \sigma \cdot \omega \rangle} P'$$

The label $\langle \pi \cdot \alpha \cdot \sigma \cdot \omega \rangle$ comprises four elements: a priority constraint π (an element of Pr), an offered interaction α, a synchronization constraint σ (an element of S), and an effect ω (an element of A). The first three are similar in purpose to those in CAB glues. The last one, ω, embodies queries and modifications of the surrounding location graph.

An offered interaction α takes the form $\{a_i\langle V_i \rangle \mid i \in I\}$, where I is a finite index set, a_i are channel names, and V_i are values.

Evalution functions. We require the existence of evaluation functions on priority constraints ($\texttt{eval}_\pi$), and on synchronization constraints ($\texttt{eval}_\sigma$) of the following types $\texttt{eval}_\pi : \mathcal{G} \times \mathcal{N}_l \times \mathsf{Pr} \to \wp_f(\mathcal{N}_l \times \mathcal{N}_r \times \mathcal{N}_c)$, and $\texttt{eval}_\sigma : \mathcal{G} \times \mathcal{N}_l \times \mathsf{S} \to \wp_f(\mathcal{N}_l \times \mathcal{N}_r \times \mathcal{N}_c \times \mathcal{V})$. The results of the above evaluation functions are called *concrete (priority or synchronization) constraints*. The presence of evaluation functions allows us to abstract away from the actual labels used in the semantics of processes, and to allow labels used in the operational semantics of location graphs, described below, to depend on the environment and the surrounding location graph.

Example 4. One can, for instance, imagine a kind of broadcast synchronization constraint of the form $* : a\langle V \rangle$, which, in the context of an environment Γ and a location l, evaluates to a constraint requiring all the roles bound to a locations in Γ to offer an interaction on channel a, i.e.: $\texttt{eval}_\sigma(\Gamma, l, * : a\langle V \rangle) = \{l.r : a\langle V \rangle \mid \exists h,\, l.r \twoheadrightarrow h \in \Gamma\}$.

We require the existence of an evaluation function on effects ($\texttt{eval}_\omega$) with the type $\texttt{eval}_\omega : \mathcal{G} \times \mathcal{N}_l \times \mathsf{A} \to \wp_f(\mathsf{E})$, where E is the set of *concrete effects*. A concrete effect can take any of the following forms, where l is a location name:

- $l : \texttt{newl}(h, P)$, $l : \texttt{newch}(c)$, $l : \texttt{newr}(r)$, respectively to create a new location named h with initial process P, to create a new channel named c, and to create a new role named r.
- $l : \texttt{add}(h, r, k)$, $l : \texttt{rmv}(h, r, k)$, respectively to add and remove a graph edge $h.r \twoheadrightarrow k$ to and from the surrounding location graph.
- $l : \texttt{gquery}(\Gamma)$, to discover a subgraph Γ of the surrounding location graph.
- $l : \texttt{swap}(h, P, Q)$, to swap the process P running at location h for process Q.

– $l : \texttt{kill}(h)$, to remove location h from the surrounding location graph.

Concrete effects embody the reconfiguration capabilities of the G-Kells framework. Effects reconfiguring the graph itself come in pairs manifesting introduction and elimination effects: thus, adding and removing a node (location) from the graph ($\texttt{newl}, \texttt{kill}$), and adding and removing an edge from the graph ($\texttt{add}, \texttt{rmv}$). Role creation ($\texttt{newr}$) is introduced to allow the creation of new edges. Channel creation ($\texttt{newch}$) allows the same flexibility provided by name creation in the π-calculus. The swap effect ($\texttt{swap}$) is introduced to allow the atomic update of a located process. The graph query effect ($\texttt{gquery}$) is perhaps a bit more unorthodox: it allows a form of reflection whereby a location can discover part of its surrounding location graph. It is best understood as an abstraction of graph navigation and query capabilities which have been found useful for programming reconfigurations in component-based systems [11].

Operational semantics of location graphs

Transitions. The operational semantics of location graphs is defined by means of a transition system, whose transition relation $\rightarrow$ is defined, by means of the inference rules presented below, as a subset of $\mathcal{G} \times \mathcal{K} \times \mathcal{L} \times \mathcal{K}$. Labels take the form $\langle \pi \cdot \sigma \cdot \omega \rangle$, where π and σ are *located priority and synchronization constraints*, respectively, and ω is a finite set of *located effects* (for simplicity, we reuse the same symbols than for constraints and effects). The set of labels is noted $\mathcal{L}$.

A located priority constraint π is just a concrete priority constraint, and takes the form $\{l_i.r_i : a_i \mid i \in I\}$, where I is a finite index set, with l_i, r_i, a_i in location, role and channel names, respectively. A located synchronization constraint takes the form $\{u_j : a_j\langle V_j \rangle\}$, where a_j are channel names, u_j are either location names l_j or pairs of location names and roles, noted $l_j.r_j$, and V_j are values. Located effects can take the following forms: $l : \texttt{rmv}(h, r, k)$, $l : \texttt{swap}(h, P, Q)$, $l : \texttt{kill}(h)$, $h : \overline{\texttt{rmv}}(h, r, k)$, $h : \overline{\texttt{swap}}(h, P, Q)$, and $h : \overline{\texttt{kill}}(h)$, where l, h, k are location names, r is a role name, P, Q are processes. The predicate $\texttt{located}$ on E identifies located effects. In particular, for a set $\omega \subset \mathsf{E}$, we have $\texttt{located}(\omega)$ if and only if all elements of ω are located.

A transition $\langle \Gamma, C, \langle \pi \cdot \sigma \cdot \omega \rangle, C' \rangle \in \rightarrow$ is noted $\Gamma \triangleright C \xrightarrow{\langle \pi \cdot \sigma \cdot \omega \rangle} C'$ and must obey the folowing constraint:

– If $\Gamma = \mathbf{u} \cup \Delta$, where $\mathbf{u}$ is a set of names, and Δ is a set of edges, then $\texttt{fn}(\Delta) \subseteq \mathbf{u}$ and $\texttt{fn}(\Gamma) = \mathbf{u}$.
– $\texttt{fn}(C) \subseteq \texttt{fn}(\Gamma)$, i.e. the free names occurring in C must appear in the already used names of Γ.
– If $l.r \twoheadrightarrow h$ and $l.r \twoheadrightarrow k$ are in $\Gamma \cup \Gamma(C)$, then $h = k$, i.e. we only have a single edge binding a role r of a given location l to another location h.

Auxiliary relation $\rightarrow_\bullet$. The definition of $\rightarrow$ makes use of an auxiliary relation $\rightarrow_\bullet \subseteq \mathcal{G} \times \mathcal{K} \times \mathcal{L}_\bullet \times \mathcal{K}$, where $\mathcal{L}_\bullet$ is a set of elements of the form $\langle \pi \cdot \sigma \cdot \omega \rangle$ where π and σ are located priority and synchronizaton constraints, and where ω is a finite

set of concrete effects. Relation $\rightarrow_\bullet$ is defined as the least relation satisfying the rules in Figure 4, where we use the following notation: if $\alpha = \{a_i\langle V_i\rangle \mid i \in I\}$, then $l : \alpha = \{l : a_i\langle V_i\rangle \mid i \in I\}$, and if $\alpha = \epsilon$, then $l : \alpha = \epsilon$.

Rule Act simply expresses how process transitions are translated into location graph transitions, and process level constraints and effects are translated into located constraints and concrete effects, via the evaluation functions introduced above. Rule NewL specifies the effect of an effect $l : \texttt{newl}(h, Q)$, which creates a new location $h[Q]$. Notice how effect $l : \texttt{newl}(h, Q)$ is removed from the set of effects in the transition label in the conclusion of the rule: auxiliary relation $\rightarrow_\bullet$ is in fact used to guarantee that a whole set of concrete effects are handled atomically. All the rules in Figure 4 except Act, which just prepares for the evaluation of effects, follow the same pattern. Rules NewC and NewR specify the creation of a new channel name and of a new role name, respectively. The rules just introduce the constraint that the new name must not be an already used name in the environment Γ. Our transitions being in the early style, the use of the new name is already taken into account in the continuation location graph C (in fact in the continuation process P' appearing on the left hand side of the instance of rule Act that must have led to the building of C). This handling of new names is a bit unorthodox but it squares nicely with the explicitly indexed labelled transition semantics of the π-calculus given by Cattani and Sewell in [8]. Rule AddE specifies the effect of adding a new edge to the location graph. Rule Gquery allows the discovery by processes of a subgraph of the location graph. In the rule premise, we use the notation Γ_l to denote the set of edges reachable from location l, formally: $\Gamma_L = \{h.r \twoheadrightarrow k \in \Gamma \mid l \twoheadrightarrow^+_\Gamma h\}$, where $l \twoheadrightarrow^+_\Gamma h$ means that there exists a non-empty chain $l.r \twoheadrightarrow l_1$, $l_1.r_1 \twoheadrightarrow l_2, \ldots, l_{n-1}.r_{n-1} \twoheadrightarrow l_n$, $l_n.r_n \twoheadrightarrow h$, with $n \geq 1$, linking l to h in the location graph Γ. As in the case of name creation rules, the exact effect on processes is left unspecified, the only constraint being that the discovered graph be indeed a subgraph of the location graph in the environment.

Transition relation $\rightarrow$. The transition relation $\rightarrow$ is defined by the rules in Figure 5. We use the following notations and definitions in Figure 5:

- Function $\texttt{seval}$ is defined by induction as follows (for any $l, r, h, V, \sigma, \sigma'$):

$$\begin{array}{ll} \texttt{seval}(\Gamma, \sigma) = \texttt{seval}(\Gamma, \sigma') & \textit{if } \sigma = \{l.r : a\langle V\rangle, h : a\langle W\rangle\} \cup \sigma' \\ & \quad \wedge\, l.r \twoheadrightarrow h \in \Gamma \wedge \texttt{match}(V, W) \\ \texttt{seval}(\Gamma, \sigma) = \sigma & \textit{otherwise} \end{array}$$

- Function $\texttt{aeval}$ is defined by induction as follows (for any $l, r, h, k, P, Q, \sigma, \sigma'$):

$$\begin{array}{ll} \texttt{aeval}(\Gamma, \sigma) = \texttt{aeval}(\Gamma, \sigma') & \textit{if} \sigma = \{l : \texttt{swap}(h, P, Q), \overline{\texttt{swap}}(h, P, Q)\} \cup \sigma' \\ \texttt{aeval}(\Gamma, \sigma) = \texttt{aeval}(\Gamma, \sigma') & \textit{if} \sigma = \{l : \texttt{rmv}(h, r, k), \overline{\texttt{rmv}}(h, r, k)\} \cup \sigma' \\ \texttt{aeval}(\Gamma, \sigma) = \texttt{aeval}(\Gamma, \sigma') & \textit{if} \sigma = \{l : \texttt{kill}(h), \overline{\texttt{kill}}(h)\} \cup \sigma' \\ \texttt{aeval}(\Gamma, \sigma) = \sigma & \textit{otherwise} \end{array}$$

$$\textsc{Act}_\bullet \quad \frac{P \xrightarrow{\langle \pi \cdot \alpha \cdot \sigma \cdot \omega \rangle} P' \qquad \pi_\bullet = \mathtt{eval}_\pi(\Gamma, l, \pi) \qquad \sigma_\bullet = \mathtt{eval}_\sigma(\Gamma, l, \sigma) \qquad \omega_\bullet = \mathtt{eval}_\omega(\Gamma, l, \omega)}{\Gamma \triangleright l[P] \xrightarrow{\langle \pi_\bullet \cdot l:\alpha \cup \sigma_\bullet \cdot \omega_\bullet \rangle}_\bullet l[P']}$$

$$\textsc{NewL} \quad \frac{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot l:\mathtt{newl}(h,Q), \omega \rangle}_\bullet C \qquad h \notin \Gamma}{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot \omega \rangle}_\bullet C \parallel h[Q]}$$

$$\textsc{NewC} \quad \frac{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot l:\mathtt{newch}(c), \omega \rangle}_\bullet C \qquad c \notin \Gamma}{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot \omega \rangle}_\bullet C}$$

$$\textsc{NewR} \quad \frac{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot l:\mathtt{newr}(r), \omega \rangle}_\bullet C \qquad r \notin \Gamma}{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot \omega \rangle}_\bullet C}$$

$$\textsc{AddE} \quad \frac{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot l:\mathtt{add}(h,r,k), \omega \rangle}_\bullet C \qquad \neg(\exists k,\, h.r \rightarrowtriangle k \in \Gamma)}{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot \omega \rangle}_\bullet C \parallel h.r \rightarrowtriangle k}$$

$$\textsc{GQuery} \quad \frac{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot l:\mathtt{gquery}(\Delta), \omega \rangle}_\bullet C \qquad \Delta \subseteq \Gamma_l}{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot \omega \rangle}_\bullet C}$$

Fig. 4. Rules for auxiliary relation $\rightarrow_\bullet$

$$\textsc{Swap} \quad \Gamma \triangleright l[P] \xrightarrow{\langle \epsilon \cdot \epsilon \cdot \overline{\mathtt{swap}}(l,P,Q) \rangle} l[Q] \qquad\qquad \textsc{Kill} \quad \Gamma \triangleright l[P] \xrightarrow{\langle \epsilon \cdot \epsilon \cdot \overline{\mathtt{kill}}(l) \rangle} \mathsf{0}$$

$$\textsc{Rmv} \quad \Gamma \triangleright h.r \rightarrowtriangle k \xrightarrow{\langle \epsilon \cdot \epsilon \cdot \overline{\mathtt{rmv}}(h,r,k) \rangle} \mathsf{0}$$

$$\textsc{Act} \quad \frac{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot \omega \rangle}_\bullet C \qquad \mathtt{located}(\omega)}{\Gamma \triangleright l[P] \xrightarrow{\langle \pi \cdot \sigma \cdot \omega \rangle} C}$$

$$\textsc{Comp} \quad \frac{\begin{array}{c}\Gamma' = \Gamma \cup \Gamma(C_1) \cup \Gamma(C_2) \qquad \pi = \pi_1 \cup \pi_2 \qquad \mathtt{fn}(\Gamma') \uplus \Delta_1 \uplus \Delta_2 = \mathcal{N} \\ \Gamma' \triangleright (\!|C_1 \parallel C_2|\!)_\pi \models \pi \qquad \sigma = \mathtt{seval}(\Gamma', \sigma_1 \cup \sigma_2) \qquad \omega = \mathtt{aeval}(\Gamma', \omega_1 \cup \omega_2) \\ \Gamma' \cup \Delta_1 \triangleright C_1 \xrightarrow{\langle \pi_1 \cdot \sigma_1 \cdot \omega_1 \rangle} C_1' \qquad \Gamma' \cup \Delta_2 \triangleright C_2 \xrightarrow{\langle \pi_2 \cdot \sigma_2 \cdot \omega_2 \rangle} C_2'\end{array}}{\Gamma \triangleright C_1 \parallel C_2 \xrightarrow{\langle \pi \cdot \sigma \cdot \omega \rangle} C_1' \parallel C_2'}$$

Fig. 5. Rules for transition relation $\rightarrow$

– Assume $\pi = \{l_i.r_i : a_i \mid i \in I\}$, then $(\!|C|\!)_\pi$ is obtained by replacing in C all the locations $\{l_i[P_i] \mid i \in I\}$ with locations $\{l_i[(\!|P_i|\!)] \mid i \in I\}$, where $(\!|P|\!)$ is defined by the LTS obtained from that of P by the following rule:

$$\textsc{Trim}\ \frac{P \xrightarrow{\langle \pi \cdot \alpha \cdot \sigma \cdot \omega \rangle} P'}{(\!|P|\!) \xrightarrow{\langle \epsilon \cdot \alpha \cdot \sigma \cdot \omega \rangle} P'}$$

  In other terms, $(\!|C|\!)_\pi$ is obtained by disregarding the priority constraints that are generated by locations l_i mentioned in the priority constraint π.
– $\Gamma \triangleright C \models \{l_i.r_i : a_i \mid i \in I\}$ means that, for all $i \in I$, $\Gamma \triangleright C \models l_i.r_i : a_i$. We write $\Gamma \triangleright C \models l.r : a$ to mean that

$$\neg(\exists h, \pi, \sigma, \omega, V, C',\ l.r \twoheadrightarrow h \ \wedge\ \Gamma \triangleright C \xrightarrow{\langle \pi \cdot l:a\langle V \rangle, \sigma \cdot \omega \rangle} C')$$

Rule Swap specifies that at any point in time a location can see its process swapped for another. Likewise, rules Kill and Rmv specify that at any point in time a location or an edge, respectively, can be removed from a location graph. Rule Act specifies the termination of the atomic execution of a set of concrete effects. All the effects remaining must be located effects, which expect some counterpart (provided by the rules Swap, Kill, and Rmv, when invoking rule Comp) to proceed.

Rule Comp is the workhorse of our operational semantics of location graphs. It specifies how to determine the transitions of a parallel composition $C_1 \parallel C_2$, by combining the priority constraints, synchronization constraints and effects obtained from the determination of contributing transitions from C_1 and C_2. The latter takes place in extended environment Γ', that contains the original environment Γ, but also the edges present in C_1 and C_2, defined as $\Gamma(C_1)$ and $\Gamma(C_2)$, respectively. To ensure the names created as side effects of C_1 and C_2 transitions are indeed unique, the determination of the contributing transition of C_1 takes place in an environment where the already used names include those in Γ as well as those in Δ_1, which gathers names that may be created as a side effect of the contributing transition of C_2. Likewise for determining the contributing transition of C_2. The constraint $\mathtt{fn}(\Gamma') \uplus \Delta_1 \uplus \Delta_2 = \mathcal{N}$ ensures that names in Δ_1 and Δ_2 are disjoint, as well as disjoint from the already used names of $\mathtt{fn}(\Gamma')$.

The original aspect of rule Comp lies with the computation of located synchronization contraints and effects resulting from the parallel composition of G-Kells components: we allow it to be dependent on the global environment Γ' with the clauses $\sigma = \mathtt{seval}(\Gamma', \sigma_1 \cup \sigma_2)$ and $\omega = \mathtt{aeval}(\Gamma', \omega_1 \cup \omega_2)$, which, in turn, allows to enforce constraints dependent on the location graph as in the definition of function `seval`. In fact, as we discuss in Section 4 below, we can envisage instances of the framework where different types of location-graph-dependent constraints apply.

The use of environments in rule Comp to obtain a quasi-compositional rule of evolution is reminiscent of the use of environment and process *frames* in the parallel rule of the ψ-calculus framework [3]. We say our rule Comp is *quasi-compositional* for the handling of priority is not compositional: it relies on the

global condition $\Gamma' \triangleright (\!|C_1 \parallel C_2|\!)_\pi$, which requires computing with (an altered view of) the whole composition. The use of the $(\!|\cdot|\!)_\_$ operator in rule Comp is reminiscent of the handling of priorities in other works [9]. An easy way to turn rule Comp into a completely compositional rule would be to adopt a more "syntactic" approach, defining $(\!|C|\!)_\pi$ to be obtained by replacing *all* locations $l[P]$ in C by their trimmed version $l[(\!|P|\!)]$, and defining environments to include information on possible actions by trimmed locations. On the other hand, we can also adopt a purely "semantic" (albeit non-compositional) variant by defining $(\!|C|\!)_\pi$ to be just C. However, in this case, we don't know whether the completeness result in Theorem 1 below still stands.

Example 5. To illustrate how the rules work, consider the following process transitions:

$$P \xrightarrow{\langle \epsilon \cdot \epsilon \cdot *:a\langle V\rangle \cdot \mathtt{newl}(h,P)\rangle} P' \qquad P_1 \xrightarrow{\langle \epsilon \cdot a\langle V_1\rangle \cdot \epsilon \cdot \{\mathtt{newl}(h_1,P_1)\}\rangle} P_1' \quad P_2 \xrightarrow{\langle \epsilon \cdot a\langle V_2\rangle \cdot \epsilon \cdot \epsilon\rangle} P_2'$$

Let $\Gamma = \{l, l_1, l_2\} \cup \{l.r_1 \twoheadrightarrow l_1, l.r_2 \twoheadrightarrow l_2\}$, and let Δ, Δ' be such that $\Delta \uplus \Delta' = \mathcal{N} \setminus \{l, l_1, l_2, h, h_1\}$. We assume further that $\mathtt{match}(V, V_1)$ and $\mathtt{match}(V, V_2)$, that all names l, l_1, l_2, h, h_1 are distinct, and that

$$\begin{aligned}
&\mathtt{eval}_\sigma(\Gamma, l, * : a\langle V\rangle) = \{l.r_1 : a\langle V\rangle, l.r_2 : a\langle V\rangle\}\\
&\mathtt{eval}_\omega(\Gamma, l, \{\mathtt{newl}(h,P)\}) = \{l : \mathtt{newl}(h,P)\}\\
&\mathtt{eval}_\omega(\Gamma, l_1, \{\mathtt{newl}(h_1,P_1)\}) = \{l_1 : \mathtt{newl}(h_1,P_1)\}\\
&\mathtt{eval}_\omega(\Gamma, l_2, \epsilon) = \epsilon \quad \mathtt{eval}_\pi(\Gamma, l, \epsilon) = \epsilon \quad \mathtt{eval}_\pi(\Gamma, l_1, \epsilon) = \epsilon \quad \mathtt{eval}_\pi(\Gamma, l_2, \epsilon) = \epsilon
\end{aligned}$$

Applying rule Act$_\bullet$, we get

$$\begin{aligned}
&\Gamma \cup \{h_1\}, \Delta \triangleright l[P] \xrightarrow{\langle \epsilon \cdot \{l.r_1:a\langle V\rangle, l.r_2:a\langle V\rangle\} \cdot \{l:\mathtt{newl}(h,P)\}\rangle}_\bullet l[P']\\
&\Gamma \cup \{h\}, \Delta' \triangleright l_1[P_1] \xrightarrow{\langle \epsilon \cdot l_1:a\langle V_1\rangle \cdot \{l_1:\mathtt{newl}(h_1,P_1)\}\rangle}_\bullet l_1[P_1']\\
&\Gamma \cup \{h\}, \Delta' \triangleright l_2[P_2] \xrightarrow{\langle \epsilon \cdot l_2:a\langle V_2\rangle \cdot \epsilon\rangle}_\bullet l_2[P_2']
\end{aligned}$$

Applying rule NewL, we get

$$\begin{aligned}
&\Gamma \cup \{h_1\}, \Delta \triangleright l[P] \xrightarrow{\langle \epsilon \cdot \{l.r_1:a\langle V\rangle, l.r_2:a\langle V\rangle\} \cdot \epsilon\rangle}_\bullet l[P'] \parallel h[P]\\
&\Gamma \cup \{h\}, \Delta' \triangleright l_1[P_1] \xrightarrow{\langle \epsilon \cdot \{h_1:a\langle V_1\rangle\} \cdot \epsilon\rangle}_\bullet l_1[P_1'] \parallel h_1[P_1]
\end{aligned}$$

Applying rule Act we get

$$\begin{aligned}
&\Gamma \cup \{h_1\}, \Delta \triangleright l[P] \xrightarrow{\langle \epsilon \cdot \{l.r_1:a\langle V\rangle, l.r_2:a\langle V\rangle\} \cdot \epsilon\rangle} l[P'] \parallel h[P]\\
&\Gamma \cup \{h\}, \Delta' \triangleright l_1[P_1] \xrightarrow{\langle \epsilon \cdot \{l_1:a\langle V_1\rangle\} \cdot \epsilon\rangle} l_1[P_1'] \parallel h_1[P_1]\\
&\Gamma \cup \{h\}, \Delta' \triangleright l_2[P_2] \xrightarrow{\langle \epsilon \cdot \{l_2:a\langle V_2\rangle\} \cdot \epsilon\rangle} l_2[P_2']
\end{aligned}$$

Finally, applying rule Comp we get

$$\begin{aligned}
\Gamma \cup \{h\}, \Delta' \triangleright l_1[P_1] \parallel l_2[P_2] &\xrightarrow{\langle \epsilon \cdot \{l_1:a\langle V_1\rangle, l_2:a\langle V_2\rangle\} \cdot \epsilon\rangle} l_1[P_1'] \parallel h_1[P_1] \parallel l_2[P_2']\\
\Gamma \triangleright l[P] \parallel l_1[P_1] \parallel l_2[P_2] &\xrightarrow{\langle \epsilon \cdot \epsilon \cdot \epsilon\rangle} l[P'] \parallel h[P] \parallel l_1[P_1'] \parallel h_1[P_1] \parallel l_2[P_2']
\end{aligned}$$

Transition relation $\rightarrow$ as a fixpoint. Because of the format of rule Comp, which does not *prima facie* obey known SOS rule formats [21], or conform to the standard notion of transition system specification [5], the question remains of which relation the rules in Figures 4 and 5 define. Instead of trying to turn our rules into equivalent rules in an appropriate format, we answer this question directly, by providing a fixpoint definition for $\rightarrow$. We use the fixpoint construction introduced by Przymusinski for the three-valued semantics of logic programs [23].

Let $\sqsubseteq$ be the ordering on pairs of relations in $\mathcal{T} = \mathcal{G} \times \mathcal{K} \times \mathcal{L} \times \mathcal{K}$ defined as:

$$\langle R_1, R_2 \rangle \sqsubseteq \langle T_1, T_2 \rangle \iff R_1 \subseteq T_1 \wedge T_2 \subseteq R_2$$

As products of complete lattices, $(\mathcal{T}, \subseteq)$ and $(\mathcal{T}^2, \sqsubseteq)$ are complete lattices [10]. One can read the Comp rule in Figures 5 as the definition of an operator $\mathcal{F} : \mathcal{T}^2 \rightarrow \mathcal{T}^2$ which operates on pairs of sub and over-approximations of $\rightarrow$. Let $\rightarrow_0$ be the relation in $\mathcal{T}^2$ obtained as the least relation satisfying the rules in Figures 4 and 5, with rule Comp omitted. Operator $\mathcal{F}$ is then defined as follows:

$$\begin{aligned}
\mathcal{F}(R_1, R_2) &= (\rightarrow_0 \cup R_1 \cup \mathsf{r}(\rightarrow_0 \cup R_1, R_2), R_2 \cap \mathsf{r}(R_2, \rightarrow_0 \cup R_1)) \\
\mathsf{r}(R_1, R_2) &= \{t \in \mathcal{T} \mid t = (\Gamma, C_1 \parallel C_2, \langle \pi \cdot \sigma \cdot \omega \rangle, C_1' \parallel C_2') \\
&\qquad \wedge \mathtt{comp}(\Gamma, C_1, C_2, \pi, \sigma, \omega, C_1', C_2', R_1, R_2)\}
\end{aligned}$$

where the predicate `comp` is defined as follows:

$$\begin{aligned}
\mathtt{comp}(\Gamma, C_1, C_2, \pi, \sigma, \omega, C_1', C_2', R_1, R_2) \iff & \\
& \exists \pi_1, \pi_2, \sigma_1, \sigma_2, \omega_1, \omega_2, \Delta_1, \Delta_2, \Gamma', \\
& \Gamma' = \Gamma \cup \Gamma(C_1) \cup \Gamma(C_2) \\
& \wedge \pi = \pi_1 \cup \pi_2 \\
& \wedge \mathtt{fn}(\Gamma') \uplus \Delta_1 \uplus \Delta_2 = \mathcal{N} \\
& \wedge \sigma = \mathtt{seval}(\Gamma', \sigma_1 \cup \sigma_2) \\
& \wedge \omega = \mathtt{aeval}(\Gamma', \omega_1 \cup \omega_2) \\
& \wedge (\Gamma' \cup \Delta_1, C_1, \langle \pi_1 \cdot \sigma_1 \cdot \omega_1 \rangle, C_1') \in R_1 \\
& \wedge (\Gamma' \cup \Delta_2, C_2, \langle \pi_2 \cdot \sigma_2 \cdot \omega_2 \rangle, C_2') \in R_1 \\
& \wedge \Gamma' \triangleright (\!| C_1 \parallel C_2 |\!)_\pi \models_{R_2} \pi
\end{aligned}$$

where $\Gamma \triangleright C \models_R \{l_i : a_i \mid i \in I\}$ means that, for all $i \in I$, $\Gamma \triangleright C \models_R l_i : a_i$, and where $\Gamma \triangleright C \models_R l : a$ stands for:

$$\neg(\exists \pi, \sigma, \omega, V, C', (\Gamma, C, \langle \pi \cdot \{l : a\langle V \rangle\} \cup \sigma \cdot \omega \rangle, C') \in R)$$

The definition of the predicate `comp` mimics the definition of rule Comp, with all the conditions in the premises appearing as clauses in `comp`, but where the positive transition conditions in the premises are replaced by transitions in the sub-approximation R_1, and negative transition conditions (appearing in the $\Gamma' \triangleright (\!| C_1 \parallel C_2 |\!)_\pi \models \pi$ condition) are replaced by equivalent conditions with transitions not belonging to the over-approximation R_2. With the definitions above, if R_1 is a sub-approximation of $\rightarrow$, and R_2 is an over-approximation of

$\rightarrow$, then we have $R_1 \subseteq \pi_1(\mathcal{F}(R_1, R_2))$ and $\pi_2(\mathcal{F}(R_1, R_2)) \subseteq R_2$, where π_1, π_2 are the first and second projections. In other terms, given a pair of sub and over approximations of $\rightarrow$, $\mathcal{F}$ computes a pair of better approximations.

From this definition, if it is easy to show that $\mathcal{F}$ is order-preserving:

Lemma 1. *For all $R_1, T_1, R_2, T_2 \in \mathcal{T}^2$, if $(R_1, R_2) \sqsubseteq (T_1, T_2)$, then $\mathcal{F}(R_1, R_2) \sqsubseteq \mathcal{F}(T_1, T_2)$.*

Since $\mathcal{F}$ is order-preserving, it has a least fixpoint, $\mathcal{F}_* = (D, U)$, by the Knaster-Tarski theorem. We can then define $\rightarrow$ to be the first projection of $\mathcal{F}_*$, namely D. With the definition of $(\!|C|\!)_\pi$ we have adopted, and noting that it provides a form of stratification with the number of locations in a location graph with non-empty priority constraints, it is also possible to show that $\rightarrow = U$, meaning that $\rightarrow$ as just defined is *complete*, using the terminology in [27]. In fact, using the terminology in [27], we can prove the theorem below, whose proof we omit for lack of space:

Theorem 1. *The relation $\rightarrow$ as defined above is the least well-supported model of the rules in Figures 4 and 5. Moreover $\rightarrow$ is complete.*

4 Discussion

We discuss in this section the various features of the G-Kells framework and relevant related work.

Introductory example. Let's first revisit Example 3 to see how we can further model the behavior of its different components. We can add, for instance, a crash action to the virtual machine locations, which can be triggered by a process transition at a virtual machine location of the form $P \xrightarrow{\langle \epsilon \cdot \epsilon \cdot \epsilon \cdot \mathtt{kill}(*) \rangle} \mathsf{0}$ with the following evaluation function:

$$\mathtt{eval}_\omega(\Gamma, l, \mathtt{kill}(*)) = \{\mathtt{kill}(h) \mid \exists r,\, l.r \twoheadrightarrow h \in \Gamma\}$$

yielding, for instance, the following transition (where $\mathbf{u}$ includes all free names in $V_0[P] \parallel C[P_C] \parallel CC[P_{CC}]$):

$$\mathbf{u}, \{V_0.0 \twoheadrightarrow C, V_0.1 \twoheadrightarrow CC\} \triangleright V_0[P] \parallel C[P_C] \parallel CC[P_{CC}] \xrightarrow{\langle \epsilon \cdot \epsilon \cdot \epsilon \rangle} V_0[\mathsf{0}] \parallel \mathsf{0} \parallel \mathsf{0}$$

This crash behavior can be extended to an arbitrary location graph residing in a virtual machine, by first discovering the location graph inside a virtual machine via the `gquery` primitive, and then killing all locations in the graph as illustrated above.

Early style. Our operational semantics for location graphs is specified in an early style [24], with values in labels manifesting the results of successful communication. This allows us to remain oblivious to the actual forms of synchronization used. For instance, one could envisage pattern matching as in the ψ-calculus [3],

$$[\![l[P]]\!] = l[P]$$
$$[\![l[C_1, \ldots, C_n \star G]]\!] = l_1[C_1] \parallel l.1 \rightarrowtriangle l_1 \parallel \ldots \parallel l_n[C_n] \parallel l.n \rightarrowtriangle l_n \parallel l[[\![G]\!]]$$
$$[\![\mathsf{0}]\!] = \mathsf{0}$$
$$[\![\langle \pi \cdot l : a \cdot \sigma \rangle.G]\!] = \langle [\![\pi]\!] \cdot \{l : a\} \cdot [\![\sigma]\!] \rangle.[\![G]\!]$$
$$[\![G_1 \mid G_2]\!] = [\![G_1]\!] \mid [\![G_2]\!]$$
$$[\![\mu X.\, G]\!] = \mu X.\, [\![G]\!]$$
$$[\![\{l_i : a_i \mid i \in I\}]\!] = \{i : a_i \mid i \in I\}$$

$$\mathtt{eval}_\pi(\Gamma, l, i : a_i) = l.i : a_i \qquad \mathtt{eval}_\sigma(\Gamma, l, i : a_i) = l.i : a_i$$

Fig. 6. Encoding CAB in the G-Kells framework

or even bi-directional pattern matching: for instance we could have a process synchronization constraint $r : a\langle x, V\rangle$ matching an offered interaction $h : a\langle W, y\rangle$, which translate into matching located synchronization constraints $l.r : a\langle W, V\rangle$ and $h : a\langle W, V\rangle$.

Mobility vs higher-order. Our operational semantics comprises both mobility features with location binding, and higher-order features with swapping and higher-order interactions. One could wonder whether these features are all needed as primitives. For instance, one could argue that mobility features are enough to model higher-order phenomena as in the π-calculus [24]. Lacking at this point a behavioral theory for the G-Kells framework, we cannot answer the question definitely here. But we doubt that mobility via location binding is sufficient to faithfully encode higher-order communication. In particular, note that we have contexts (location graphs) that can distinguish the two cases via the ability to kill locations selectively.

Directed graphs vs acyclic directed graphs. Location graphs form directed graphs. One could wonder whether to impose the additional constraints that such graphs be acyclic. While most meaningul examples of ubiquitous systems and software structures can be modeled with acyclic directed graphs, our rules for location graphs function readily with arbitrary graphs. Enforcing the constraint that all evolutions of a location graph keep it acyclic does not seem necessary.

Relationship with CAB. The G-Kells model constitutes a conservative extension of CAB. A straightforwward encoding of CAB in the G-Kells framework can be defined as in Figure 6, with translated glues $[\![G]\!]$ defined with the same LTS, mutatis mutandis, as CAB glues. The following proposition is then an easy consequence of our definitions:

Proposition 1. *Let C be a CAB component. We have $C \xrightarrow{l:a} C'$ if and only if $[\![C]\!] \xrightarrow{\langle \epsilon \cdot \{l:a\} \cdot \epsilon \rangle} [\![C']\!]$.*

Graph constraints in rules. For simplicity, the evaluation functions `seval` and `aeval` have been defined above with only a simple graph constraint in the first clause of the `seval` definition. One can parameterize these definitions with additional graph constraints to enforce different policies. For instance, one could constrain the use of the swap, kill and and edge removal operations to locations dominating the target location by adding a constraint of the form $l \rightarrowtriangle^* h$ to each of the clauses in the definition of `aeval`, where $l \rightarrowtriangle^*_\Gamma h$ means that there exists a (possibly empty) chain $l.r \rightarrowtriangle l_1$, $l_1.r_1 \rightarrowtriangle l_2, \ldots, l_{n-1}.r_{n-1} \rightarrowtriangle l_n$, $l_n.r_n \rightarrowtriangle h$ linking l to h in the location graph Γ. Similar constraints could be added to rule AddE. Further constraints could be added to rule Gquery to further restrict the discovery of subgraphs, for instance, preventing nodes other than immediate children to be discovered.

Types and capabilities. The framework presented in this paper is an untyped one. However, introducing types similar to i/o types capabilities in the π-calculus [24] would be highly useful. For instance, edges of a location graph can be typed, perhaps with as simple a scheme as different colors to reflect different containment and visibility semantics, which can be exploited in the definition of evaluation functions to constrain effects and synchronization. In addition, location, role and channel names can be typed with capabilities constraining the transfer of rights from one location to another. For instance, transferring a location name l can come with the right to swap the behavior at l, but not with the right to kill l, or with the right to bind roles of l to locations, but not with the ability to swap the behavior at l. We believe these capabilities could be useful in enforcing encapsulation and access control policies.

Relation with the ψ-calculus framework and SCEL. We already remarked that our use of environments is reminiscent of the use of frames in the ψ-calculus framework [3]. An important difference with the ψ-calculus framework is the fact that we allow interactions to depend on constraints involving the global environment, in our case the structure of the location graph. Whether one can faithfully encode the G-Kells framework (with mild linguistic assumptions on processes) with the ψ-calculus framework remains to be seen.

On the other hand, it would seem worthwhile to pursue the extension of the framework presented here with ψ-calculus-like assertions. We wonder in particular what relation the resulting framework would have with the SCEL language for autonomous and adaptive systems [12]. The notion of *ensemble*, being assertion-based, is more fluid than our notion of location graph, but it does not have the ability to superimpose on ensembles the kind of control actions, such as swapping and killing, that the G-Kells framework allows.

Relation with SHR. The graph manipulation capabilities embedded in the G-Kells framework are reminiscent of synchronized hyperdege replacement (SHR) systems [18]. In SHR, multiple hyperedge replacements can be synchronized to yield an atomic transformation of the underlying hypergraph in conjunction with information exchange. Intuitively, it seems one can achieve much the same effects

with G-Kells: located effects can atomically build a new subgraph and modify the existing one, and they can be synchronized across multiple locations thanks to synchronization constraints. In contrast, SHR systems lack priorities and the internalization of hyperedge replacement rules (the equivalent of our processes) in graph nodes to account for inherent dynamic reconfiguration capabilities. We conjecture that SHR systems can be faithfully encoded in the G-Kells framework.

5 Conclusion

We have introduced the G-Kells framework to lift limitations in existing computational models for ubquitous and reconfigurable software systems, in particular the ability to describe dynamic structures with sharing, where different aggregates or composites can share components. Much work remains to be done, however. We first intend to develop the behavioral theory of our framework. Indeed we hope to develop a first-order bisimulation theory for the G-Kells framework, avoiding the difficulties inherent in mixing higher-order features with passivation described in [19]. We also need to formally compare G-Kells with several other formalisms, including SHR systems and the ψ-calculus framework. And we definitely need to develop a typed variant of the framework to exploit the rich set of capabilities that can be attached to location names.

Ackowledgements

Damien Pous suggested the move to an early style semantics. The paper was much improved thanks to comments by Ivan Lanese on earlier versions.

References

1. C. Baier, M. Sirjani, F. Arbab, and J. J. M. M. Rutten. Modeling component connectors in reo by constraint automata. *Sci. Comput. Program.*, 61(2), 2006.
2. F. Barbier, B. Henderson-Sellers, A. Le Parc, and J.M. Bruel. Formalization of the whole-part relationship in the unified modeling language. *IEEE Trans. Software Eng.*, 29(5), 2003.
3. J. Bengtson, M. Johansson, J. Parrow, and B. Victor. Psi-calculi: a framework for mobile processes with nominal data and logic. *Logical Methods in Computer Science*, 7(1), 2011.
4. S. Bliudze and J. Sifakis. A notion of glue expressiveness for component-based systems. In *CONCUR*, volume 5201 of *LNCS*. Springer, 2008.
5. R. N. Bol and J. F. Groote. The meaning of negative premises in transition system specifications. *J. ACM*, 43(5):863–914, 1996.
6. M. Bugliesi, G. Castagna, and S. Crafa. Access control for mobile agents: the calculus of boxed ambients. *ACM. Trans. Prog. Languages and Systems*, 26(1), 2004.
7. L. Cardelli and A. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1), 2000.

8. G. L. Cattani and P. Sewell. Models for name-passing processes: interleaving and causal. *Information and Computation*, 190(2), 2004.
9. R. Cleaveland, G. Lüttgen, and V. Natarajan. Priority in process algebra. In *Handbook of Process Algebra*. Elsevier, 2001.
10. B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order (2nd ed.)*. Cambridge University Press, 2002.
11. P.C. David, T. Ledoux, M. Léger, and T. Coupaye. Fpath and fscript: Language support for navigation and reliable reconfiguration of fractal architectures. *Annals of Telecommunications*, 64(1-2), 2009.
12. R. De Nicola, M. Loreti, R. Pugliese, and F. Tiezzi. A formal approach to autonomic systems programming: The SCEL language. *ACM Trans. on Autonomous and Adaptive Systems*, 9(2), 2014.
13. C. Di Giusto and J.-B. Stefani. Revisiting glues for component-based systems. In *COORDINATION 2011*, volume 6721 of *LNCS*. Springer, 2011.
14. G. L. Ferrari, D. Hirsch, I. Lanese, U. Montanari, and E. Tuosto. Synchronised hyperedge replacement as a model for service oriented computing. In *FMCO 2005*, volume 4111 of *LNCS*. Springer, 2005.
15. J. L. Fiadeiro and A. Lopes. A model for dynamic reconfiguration in service-oriented architectures. *Software and System Modeling*, 12(2), 2013.
16. D. Hirschkoff, T. Hirschowitz, D. Pous, A. Schmitt, and J.-B. Stefani. Component-Oriented Programming with Sharing: Containment is not Ownership. In *GPCE 2005*, volume 3676 of *LNCS*. Springer, 2005.
17. P. Kruchten. Architectural blueprints – The 4+1 view model of software architecture. *IEEE Software*, 12(6), 1995.
18. I. Lanese and U. Montanari. Mapping fusion and synchronized hyperedge replacement into logic programming. *Theory and Practice of Logic Programming*, 7(1-2), 2007.
19. S. Lenglet, A. Schmitt, and J.B. Stefani. Characterizing contextual equivalence in calculi with passivation. *Information and Computation*, 209(11), 2011.
20. R. Milner. *The Space and Motion of Communicating Agents*. Cambridge University Press, 2009.
21. M. R. Mousavi, M. A. Reniers, and J. F. Groote. SOS formats and meta-theory: 20 years after. *Theoretical Computer Science*, 373(3), 2007.
22. F. Oquendo. π-ADL: An Architecture Description Language based on the Higher-Order π-Calculus for Specifying Dynamic and Mobile Software Architectures. *ACM Software Engineering Notes*, 29(4), 2004.
23. T.C. Przymusinski. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae*, 13, 1990.
24. D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.
25. A. Schmitt and J.-B. Stefani. The Kell Calculus: A Family of Higher-Order Distributed Process Calculi. In *Global Computing*, volume 3267 of *LNCS*. Springer, 2005.
26. S. Tripakis, C. Stergiou, C. Shaver, and E. A. Lee. A modular formal semantics for ptolemy. *Mathematical Structures in Computer Science*, 23(4), 2013.
27. R. J. van Glabbeek. The meaning of negative premises in transition system specifications II. *J. Log. Algebr. Program.*, 60-61, 2004.
28. M. Wermelinger and J. L. Fiadeiro. A graph transformation approach to software architecture reconfiguration. *Sci. Comput. Program.*, 44(2), 2002.

Bisimulations up-to: beyond first-order transition systems

Jean-Marie Madiot[1], Damien Pous[1], and Davide Sangiorgi[2]

[1] ENS Lyon, Université de Lyon, CNRS, INRIA, France,
[2] Università di Bologna, Italy, INRIA

Abstract. The bisimulation proof method can be enhanced by employing *'bisimulations up-to'* techniques. A comprehensive theory of such enhancements has been developed for first-order (i.e., CCS-like) labelled transition systems (LTSs) and bisimilarity, based on the notion of compatible function for fixed-point theory.
We transport this theory onto languages whose bisimilarity and LTS go beyond those of first-order models. The approach consists in exhibiting fully abstract translations of the more sophisticated LTSs and bisimilarities onto the first-order ones. This allows us to reuse directly the large corpus of up-to techniques that are available on first-order LTSs. The only ingredient that has to be manually supplied is the compatibility of basic up-to techniques that are specific to the new languages. We investigate the method on the π-calculus, the λ-calculus, and a (call-by-value) λ-calculus with references.

1 Introduction

One of the keys for the success of bisimulation is its associated proof method, whereby to prove two terms equivalent, one exhibits a relation containing the pair and one proves it to be a bisimulation. The bisimulation proof method can be enhanced by employing relations called *'bisimulations up-to'* [14, 19, 20]. These need not be bisimulations; they are simply *contained in* a bisimulation. Such techniques have been widely used in languages for mobility such as π-calculus or higher-order languages such as the λ-calculus, or Ambients (e.g., [23, 15, 11]).

Several forms of bisimulation enhancements have been introduced: 'bisimulation up-to bisimilarity' [16] where the derivatives obtained when playing bisimulation games can be rewritten using bisimilarity itself; 'bisimulation up-to transitivity' where the derivatives may be rewritten using the up-to relation; 'bisimulation up-to-context' [21], where a common context may be removed from matching derivatives. Further enhancements may exploit the peculiarities of the definition of bisimilarity on certain classes of languages: e.g., the up-to-injective-substitution techniques of the π-calculus [7, 23], techniques for shrinking or enlarging the environment in languages with information hiding mechanisms (e.g., existential types, encryption and decryption constructs [1, 25, 24]), frame equivalence in the psi-calculi [17], or higher-order languages [12, 10]. Lastly, it is important to notice that one often wishes to use *combinations* of up-to techniques.

For instance, up-to context alone does not appear to be very useful; its strength comes out in association with other techniques, such as up-to bisimilarity or up-to transitivity.

The main problem with up-to techniques is proving their soundness (i.e. ensuring that any 'bisimulation up-to' is contained in bisimilarity). In particular, the proofs of complex combinations of techniques can be difficult or, at best, long and tedious. And if one modifies the language or the up-to technique, the entire proof has to be redone from scratch. Indeed the soundness of some up-to techniques is quite fragile, and may break when such variations are made. For instance, in certain models up-to bisimilarity may fail for weak bisimilarity, and in certain languages up-to bisimilarity and context may fail even if bisimilarity is a congruence relation and is strong (treating internal moves as any other move).

This problem has been the motivation for the development of a theory of enhancements, summarised in [19]. Expressed in the general fixed-point theory on complete lattices, this theory has been fully developed for both strong and weak bisimilarity, in the case of first-order labelled transition systems (LTSs) where transitions represent pure synchronisations among processes. In this framework, up-to techniques are represented using *compatible* functions, whose class enjoys nice algebraic properties. This allows one to derive complex up-to techniques algebraically, by composing simpler techniques by means of a few operators.

Only a small part of the theory has been transported onto other forms of transition systems, on a case by case basis. Transferring the whole theory would be a substantial and non-trivial effort. Moreover it might have limited applicability, as this work would probably have to be based on specific shapes for transitions and bisimilarity (a wide range of variations exist, e.g., in higher-order languages).

Here we explore a different approach to the transport of the theory of bisimulation enhancements onto richer languages. The approach consists in exhibiting fully abstract translations of the more sophisticated LTSs and bisimilarities onto first-order LTSs and bisimilarity. This allows us to import directly the existing theory for first-order bisimulation enhancements onto the new languages. Most importantly, the schema allows us to combine up-to techniques for the richer languages. The only additional ingredient that has to be provided manually is the soundness of some up-to techniques that are specific to the new languages. This typically includes the up-to context techniques, since those contexts are not first-order.

Our hope is that the method proposed here will make it possible to obtain a single formalised library about up-to techniques, that can be reused for a wide range of calculi: currently, all existing formalisations of such techniques in a proof assistant are specific to a given calculus: π-calculus [5, 4], the psi-calculi [17], or a miniML language [6].

We consider three languages: the π-calculus, the call-by-name λ-calculus, and an imperative call-by-value λ-calculus. Other calculi like the Higher-Order π-calculus can be handled in a similar way; we omit the details here for lack of space. We moreover focus on weak bisimilarity, since its theory is more delicate than that of strong bisimilarity. When we translate a transition system into a

first-order one, the grammar for the labels can be complex (e.g. include terms, labels, or contexts). What makes the system 'first-order' is that labels are taken as syntactic atomic objects, that may only be checked for syntactic equality. Note that full abstraction of the translation does not imply that the up-to techniques come for free: further conditions must be ensured. We shall see this with the π-calculus, where early bisimilarity can be handled but not the late one.

Forms of up-to context have already been derived for the languages we consider in this paper [11, 23, 22]. The corresponding soundness proofs are difficult (especially in λ-calculi), and require a mix of induction (on contexts) and coinduction (to define bisimulations). Recasting up-to context within the theory of bisimulation enhancements has several advantages. First, this allows us to combine this technique with other techniques, directly. Second, substitutivity (or congruence) of bisimilarity becomes a corollary of the compatibility of the up-to-context function (in higher-order languages these two kinds of proofs are usually hard and very similar). And third, this allows us to decompose the up-to context function into smaller pieces, essentially one for each operator of the language, yielding more modular proofs, also allowing, if needed, to rule out those contexts that do not preserve bisimilarity (e.g., input prefix in the π-calculus).

The translation of the π-calculus LTS into a first-order LTS follows the schema of abstract machines for the π-calculus (e.g., [26]) in which the issue of the choice of fresh names is resolved by ordering the names. Various forms of bisimulation enhancements have appeared in papers on the π-calculus or dialects of it. A translation of higher-order π-calculi into first-order processes has been proposed by Koutavas et al [8]. While the shape of our translations of λ-calculi is similar, our LTSs differ since they are designed to recover the theory of bisimulation enhancements. In particular, using the LTSs from [8] would lead to technical problems similar to those discussed in Remark 2. In the λ-calculus, limited forms of up-to techniques have been developed for applicative bisimilarity, where the soundness of the up-to context has still open problems [12, 11]. More powerful versions of up-to context exist for forms of bisimilarity on open terms (e.g., open bisimilarity or head-normal-form bisimilarity) [13]. Currently, the form of bisimilarity for closed higher-order terms that allows the richest range of up-to techniques is environmental bisimilarity [22, 9]. However, even in this setting, the proofs of combinations of up-to techniques are usually long and non-trivial. Our translation of higher-order terms to first-order terms is designed to recover environmental bisimilarity.

In Section 6, we show an example of how the wide spectrum of up-to techniques made available via our translations allows us to simplify relations needed in bisimilarity proofs, facilitating their description and reducing their size.

2 First-order bisimulation and up-to techniques

A *first-order Labelled Transition System*, briefly LTS, is a triple $(Pr, Act, \longrightarrow)$ where Pr is a non-empty set of states (or processes), Act is the set of *actions* (or *labels*), and $\longrightarrow \subseteq Pr \times Act \times Pr$ is the *transition relation*. We use P, Q, R to

range over the processes of the LTS, and μ to range over the labels in Act, and, as usual, write $P \xrightarrow{\mu} Q$ when $(P, \mu, Q) \in \longrightarrow$. We assume that Act includes a special action τ that represents an internal activity of the processes. We derive bisimulation from the notion of *progression* between relations.

Definition 1. *Suppose $\mathcal{R}, \mathcal{S}$ are relations on the processes of an LTS. Then $\mathcal{R}$* strongly progresses to *$\mathcal{S}$, written $\mathcal{R} \rightsquigarrow_{\mathbf{sp}} \mathcal{S}$, if $\mathcal{R} \subseteq \mathcal{S}$ and if $P \mathcal{R} Q$ implies:*

- *whenever $P \xrightarrow{\mu} P'$ there is Q' s.t. $Q \xrightarrow{\mu} Q'$ and $P' \mathcal{S} Q'$;*
- *whenever $Q \xrightarrow{\mu} Q'$ there is P' s.t. $P \xrightarrow{\mu} P'$ and $P' \mathcal{S} Q'$.*

A relation $\mathcal{R}$ is a strong bisimulation *if $\mathcal{R} \rightsquigarrow_{\mathbf{sp}} \mathcal{R}$; and* strong bisimilarity*, $\sim$, is the union of all strong bisimulations.*

To define weak progression we need weak transitions, defined as usual: first, $P \xrightarrow{\hat{\mu}} P'$ means $P \xrightarrow{\mu} P'$ or $\mu = \tau$ and $P = P'$; and $\stackrel{\hat{\mu}}{\Longrightarrow}$ is $\Longrightarrow \xrightarrow{\hat{\mu}} \Longrightarrow$ where $\Longrightarrow$ is the reflexive transitive closure of $\xrightarrow{\tau}$. *Weak progression*, $\mathcal{R} \rightsquigarrow_{\mathbf{wp}} \mathcal{S}$, and *weak bisimilarity*, $\approx$, are obtained from Definition 1 by allowing the processes to answer using $\stackrel{\hat{\mu}}{\Longrightarrow}$ rather than $\xrightarrow{\mu}$.

Below we summarise the ingredients of the theory of bisimulation enhancements for first-order LTSs from [19] that will be needed in the sequel. We use f and g to range over functions on relations over a fixed set of states. Each such function represents a potential up-to technique; only the *sound* functions, however, qualify as up-to techniques:

Definition 2. *A function f is* sound for $\sim$ *if $\mathcal{R} \rightsquigarrow_{\mathbf{sp}} f(\mathcal{R})$ implies $\mathcal{R} \subseteq \sim$, for all $\mathcal{R}$; similarly, f is* sound for $\approx$ *if $\mathcal{R} \rightsquigarrow_{\mathbf{wp}} f(\mathcal{R})$ implies $\mathcal{R} \subseteq \approx$, for all $\mathcal{R}$.*

Unfortunately, the class of sound functions does not enjoy good algebraic properties. As a remedy to this, the subset of *compatible* functions has been proposed. The concepts in the remainder of the section can be instantiated with both strong and weak bisimilarities; we thus use $\mathbf{p}$ to range over $\mathbf{sp}$ or $\mathbf{wp}$.

Definition 3. *We write $f \rightsquigarrow_{\mathbf{p}} g$ when $\mathcal{R} \rightsquigarrow_{\mathbf{p}} \mathcal{S}$ implies $f(\mathcal{R}) \rightsquigarrow_{\mathbf{p}} g(\mathcal{S})$ for all $\mathcal{R}$ and $\mathcal{S}$. A monotone function f on relations is* $\mathbf{p}$-compatible *if $f \rightsquigarrow_{\mathbf{p}} f$.*

In other terms [19], f is $\mathbf{p}$-compatible iff $f \circ \mathbf{p} \subseteq \mathbf{p} \circ f$ where $\mathbf{p}(\mathcal{S})$ is the union of all $\mathcal{R}$ such that $\mathcal{R} \rightsquigarrow_{\mathbf{p}} \mathcal{S}$ and $\circ$ denotes function composition. Note that $\mathcal{R} \rightsquigarrow_{\mathbf{p}} \mathcal{S}$ is equivalent to $\mathcal{R} \subseteq \mathbf{p}(\mathcal{S})$.

Lemma 1. *If f is $\mathbf{sp}$-compatible, then f is sound for $\sim$; if f is $\mathbf{wp}$-compatible, then f is sound for $\approx$.*

Simple examples of compatible functions are the identity function and the function mapping any relation onto bisimilarity (for the strong or weak case, respectively). The class of compatible functions is closed under function composition and union (where the union $\cup F$ of a set of functions F is the point-wise union

mapping $\mathcal{R}$ to $\bigcup_{f \in F} f(\mathcal{R})$), and thus under omega-iteration (where the omega-iteration f^ω of a function f maps $\mathcal{R}$ to $\bigcup_{n \in \mathbb{N}} f^n(\mathcal{R})$).

Other examples of compatible functions are typically contextual closure functions, mapping a relation into its closure w.r.t. a given set of contexts. For such functions, the following lemma shows that the compatibility of up-to-context implies substitutivity of (strong or weak) bisimilarity.

Lemma 2. *If f is* **sp***-compatible, then $f(\sim) \subseteq \sim$; similarly if f is* **wp***-compatible, then $f(\approx) \subseteq \approx$.*

Certain closure properties for compatible functions however only hold in the strong case. The main example is the *chaining operator* $\frown$, which implements relational composition:

$$f \frown g\ (\mathcal{R}) \triangleq f(\mathcal{R})\ g(\mathcal{R})$$

where $f(\mathcal{R})\ g(\mathcal{R})$ indicates the composition of the two relations $f(\mathcal{R})$ and $g(\mathcal{R})$. Using chaining we can obtain the compatibility of the function 'up to transitivity' mapping any relation $\mathcal{R}$ onto its reflexive and transitive closure $\mathcal{R}^\star$. Another example of **sp**-compatible function is 'up to bisimilarity' ($\mathcal{R} \mapsto \sim\mathcal{R}\sim$).

In contrast, in the weak case bisimulation up to bisimilarity is unsound. This is a major drawback in up-to techniques for weak bisimilarity, which can be partially overcome by resorting to the *expansion* relation $\gtrsim$ [3]. Expansion is an asymmetric refinement of weak bisimilarity whereby $P \gtrsim Q$ holds if P and Q are bisimilar and, in addition, Q is at least as efficient as P, in the sense that Q is capable of producing the same activity as P without ever performing more internal activities (the τ-actions); see Appendix A for its definition. Up-to-expansion yields a function ($\mathcal{R} \mapsto \gtrsim\mathcal{R}\lesssim$) that is **wp**-compatible. As a consequence, the same holds for the 'up-to expansion and contexts' function. More sophisticated up-to techniques can be obtained by carefully adjusting the interplay between visible and internal transitions, and by taking into account termination hypotheses [19].

Some further compatible functions are the functions **sp** and **wp** themselves (indeed a function f is **p**-compatible if $f \circ \mathbf{p} \subseteq \mathbf{p} \circ f$, hence trivially f can be replaced by **p** itself). Intuitively, the use of **sp** and **wp** as up-to techniques means that, in a diagram-chasing argument, the two derivatives need not be related; it is sufficient that the derivatives of such derivatives be related. Accordingly, we sometimes call functions **sp** and **wp** *unfolding* functions. We will use **sp** in the example in Section 6 and **wp** in Sections 4 and 5, when proving the **wp**-compatibility of the up to context techniques.

Last, note that to use a function f in combinations of up-to techniques, it is actually not necessary that f be **p**-compatible: for example proving that f progresses to $f \cup g$ and g progresses to g is enough, as then $f \cup g$ would be compatible. Extending this reasoning allows us to make use of 'second-order up-to techniques' to reason about compatibility of functions. When F is a set of functions, we say that F is **p**-*compatible up to* if for all f in F, it holds that $f \rightsquigarrow_{\mathbf{p}} (g \cup (\cup F))^\omega$ for a function g that has already been proven compatible. (We sometimes say that F is **p**-*compatible up to g*, to specify which compatible

function is employed.) Lemma 1 and 2 remain valid when 'f is compatible' is replaced by '$f \in F$ and F is compatible up to'.

Terminology We will simply say that a function is *compatible* to mean that it is both **sp**-compatible and **wp**-compatible; similarly for compatibility up to. In languages defined from a grammar, a context C is a term with numbered holes $[\cdot]_1, \ldots, [\cdot]_n$, and each hole $[\cdot]_i$ can appear any number of times in C.

3 The π-calculus

The syntax and operational semantics of the π-calculus are recalled in Appendix B. We consider the early transition system, in which transitions are of the forms

$$P \stackrel{ab}{\longmapsto}_\pi P' \qquad P \stackrel{\overline{a}b}{\longmapsto}_\pi P' \qquad P \stackrel{\overline{a}(b)}{\longmapsto}_\pi P' \ .$$

In the third transition, called bound output transition, name b is a binder for the free occurrences of b in P' and, as such, it is subject to α-conversion. The definition of bisimilarity takes α-conversion into account. The clause for bound output of strong early bisimilarity says ($\mathrm{fn}(Q)$ indicates the names free in Q):

– if $P \stackrel{\overline{a}(b)}{\longmapsto}_\pi P'$ and $b \notin \mathrm{fn}(Q)$ then $Q \stackrel{\overline{a}(b)}{\longmapsto}_\pi Q'$ for some Q' such that $P' \sim Q'$.

(The complete definition of bisimilarity is recalled in Appendix B). When translating the π-calculus semantics to a first-order one, α-conversion and the condition $b \notin \mathrm{fn}(Q)$ have to be removed. To this end, one has to force an agreement between two bisimilar process on the choice of the bound names appearing in transitions. We obtain this by considering *named processes* (c, P) in which c is bigger or equal to all names in P. For this to make sense we assume an enumeration of the names and use $\leq$ as the underlying order, and $c+1$ for name following c in the enumeration; for a set of names N, we also write $c \geq N$ to mean $c \geq a$ for all $a \in N$.

The rules below define the translation of the π-calculus transition system to a first-order LTS. In the first-order LTS, the grammar for labels is the same as that of the original LTS; however, for a named process (c, P) the only name that may be exported in a bound output is $c+1$; similarly only names that are below or equal to $c+1$ may be imported in an input transition. (Indeed, testing for all fresh names $b > c$ is unnecessary, doing it only for one ($b = c+1$) is enough.) This makes it possible to use the ordinary definition of bisimilarity for first-order LTS, and thus recover the early bisimilarity on the source terms.

$$\frac{P \stackrel{\tau}{\longmapsto}_\pi P'}{(c,P) \stackrel{\tau}{\longrightarrow} (c,P')} \qquad \frac{P \stackrel{ab}{\longmapsto}_\pi P'}{(c,P) \stackrel{ab}{\longrightarrow} (c,P')}\ b \leq c \qquad \frac{P \stackrel{\overline{a}b}{\longmapsto}_\pi P'}{(c,P) \stackrel{\overline{a}b}{\longrightarrow} (c,P')}\ b \leq c$$

$$\frac{P \stackrel{ab}{\longmapsto}_\pi P'}{(c,P) \stackrel{ab}{\longrightarrow} (b,P')}\ b = c+1 \qquad \frac{P \stackrel{\overline{a}(b)}{\longmapsto}_\pi P'}{(c,P) \stackrel{\overline{a}(b)}{\longrightarrow} (b,P')}\ b = c+1$$

We write π^1 for the first-order LTS derived from the above translation of the π-calculus. Although the labels of the source and target transitions have a similar shape, the LTS in π^1 is first-order because labels are taken as purely syntactic objects (without α-conversion). We write $\sim^{\mathrm{e}}$ and $\approx^{\mathrm{e}}$ for strong and weak early bisimilarity of the π-calculus.

Theorem 1. *Assume $c \geq \mathrm{fn}(P) \cup \mathrm{fn}(Q)$. Then we have: $P \sim^{\mathrm{e}} Q$ iff $(c, P) \sim (c, Q)$, and $P \approx^{\mathrm{e}} Q$ iff $(c, P) \approx (c, Q)$.*

The above full abstraction result allows us to import the theory of up-to techniques for first-order LTSs and bisimilarity, both in the strong and the weak case. We have however to prove the soundness of up-to techniques that are specific to the π-calculus. Function isub implements 'up to injective name substitutions':

$$\mathsf{isub}(\mathcal{R}) \triangleq \{((d, P\sigma), (d, Q\sigma)) \;\; \text{s.t. } (c, P)\ \mathcal{R}\ (c, Q),\ \mathrm{fn}(P\sigma) \cup \mathrm{fn}(Q\sigma) \leq d, \text{ and } \sigma \text{ is injective on } \mathrm{fn}(P) \cup \mathrm{fn}(Q) \;\; \} \quad .$$

A subtle drawback is the need of another function manipulating names, str, allowing us to replace the index c in a named process (c, P) with a lower one:

$$\mathsf{str}(\mathcal{R}) \triangleq \{((d, P), (d, Q)) \;\; \text{s.t. } (c, P)\ \mathcal{R}\ (c, Q) \text{ and } \mathrm{fn}(P, Q) \leq d \;\} \quad .$$

Lemma 3. *The set $\{\mathsf{isub}, \mathsf{str}\}$ is compatible up to.*

The up-to-context function is decomposed into a set of smaller context functions, called *initial* [19], one for each operator of the π-calculus. The only exception to this is the input prefix, since early bisimilarity in the π-calculus is not preserved by this operator. We write $\mathcal{C}_o$, $\mathcal{C}_\nu$, $\mathcal{C}_!$, $\mathcal{C}_|$, and $\mathcal{C}_+$ for these initial context functions, respectively returning the closure of a relation under the operators of output prefix, restriction, replication, parallel composition, and sum.

Definition 4. *If $\mathcal{R}$ is a relation on π^1, we define $\mathcal{C}_o(\mathcal{R})$, $\mathcal{C}_\nu(\mathcal{R})$, $\mathcal{C}_!(\mathcal{R})$, $\mathcal{C}_|(\mathcal{R})$ and $\mathcal{C}_+(\mathcal{R})$ by saying that whenever $(c, P)\ \mathcal{R}\ (c, Q)$,*

- $(c, \overline{a}b.P)\ \ \mathcal{C}_o(\mathcal{R})\ \ (c, \overline{a}b.Q)$, *for any a, b with $a, b \leq c$,*
- $(c, \nu a.P)\ \ \mathcal{C}_\nu(\mathcal{R})\ \ (c, \nu a.Q)$,
- $(c, !P)\ \ \mathcal{C}_!(\mathcal{R})\ \ (c, !Q)$;

and, whenever $(c, P_1)\ \mathcal{R}\ (c, Q_1)$ and $(c, P_2)\ \mathcal{R}\ (c, Q_2)$,

- $(c, P_1 \mid Q_1)\ \ \mathcal{C}_|(\mathcal{R})\ \ (c, P_2 \mid Q_2)$,
- $(c, P_1 + Q_1)\ \ \mathcal{C}_+(\mathcal{R})\ \ (c, P_2 + Q_2)$.

While bisimilarity in the π-calculus is not preserved by input prefix, a weaker rule holds (where $=$ can be $\sim^{\mathrm{e}}$ or $\approx^{\mathrm{e}}$):

$$\frac{P = Q \quad \text{and } P\{c/\!\!/b\} = Q\{c/\!\!/b\} \text{ for each } c \text{ free in } P, Q}{a(b).P = a(b).Q} \tag{1}$$

We define $\mathcal{C}_i$, the function for input prefix, accordingly: we have $(d, a(b).P)\ \mathcal{C}_i(\mathcal{R})\ (d, a(b).Q)$ if $a \leq d$ and $(d+1, P\{c/b\})\ \mathcal{R}\ (d+1, Q\{c/b\})$ for all $c \leq d+1$.

Theorem 2. *The set* $\{\mathcal{C}_o, \mathcal{C}_i, \mathcal{C}_\nu, \mathcal{C}_!, \mathcal{C}_|, \mathcal{C}_+\}$ *is* **sp***-compatible up to* $\mathsf{isub} \cup \mathsf{str}$.

Weak bisimilarity is not preserved by sums, only by guarded sums, whose function is $\mathcal{C}_{g+} \triangleq \mathcal{C}_+^\omega \circ (\mathcal{C}_o \cup \mathcal{C}_i)$.

Theorem 3. *The set* $\{\mathcal{C}_o, \mathcal{C}_i, \mathcal{C}_\nu, \mathcal{C}_!, \mathcal{C}_|, \mathcal{C}_{g+}\}$ *is* **wp***-compatible up to* $\mathsf{isub} \cup \mathsf{str} \cup \mathbf{b}$ *where* $\mathbf{b} = (\mathcal{R} \mapsto \sim\mathcal{R}\sim)$ *is 'up to bisimilarity'.*

The compatibility of these functions is not a logical consequence of the up to context results in the π-calculus; instead we prove them from scratch (see Appendix B), with the benefit of having a separate proof for each initial context.

As a byproduct of the compatibility of these initial context functions, and using Lemma 2, we derive the standard substitutivity properties of strong and weak early bisimilarity, including the rule (1) for input prefix.

Corollary 1. *In the* π*-calculus, relations* $\sim^{\mathrm{e}}$ *and* $\approx^{\mathrm{e}}$ *are preserved by the operators of output prefix, replication, parallel composition, restriction;* $\sim^{\mathrm{e}}$ *is also preserved by sum, whereas* $\approx^{\mathrm{e}}$ *is only preserved by guarded sums. Moreover, rule (1) is valid both for* $\sim^{\mathrm{e}}$ *and* $\approx^{\mathrm{e}}$.

Remark 1. Late bisimilarity makes use of transitions $P \overset{a(b)}{\longmapsto}_\pi P'$ where b is bound, the definition of bisimulation containing a quantification over names. To capture this bisimilarity in a first-order LTS we would need to have two transitions for the input $a(b)$: one to fire the input a, leaving b uninstantiated, and another to instantiate b. While such a translation does yield full abstraction for both strong and weak late bisimilarities, the decomposition of an input transition into two steps prevents us from obtaining the compatibility of up to context.

4 Call-by-name λ-calculus

To study the applicability of our approach to higher-order languages, we investigate the pure call-by-name λ-calculus, referred to as ΛN in the sequel.

We use M, N to range over the set Λ of λ-terms, and x, y, z to range over variables. The standard syntax of λ-terms, and the rules for call-by-name reduction, are recalled in Appendix C. We assume the familiar concepts of free and bound variables and substitutions, and identify α-convertible terms. The only values are the λ-abstractions $\lambda x.M$. In this section and in the following one, results and definitions are presented on closed terms; extension to open terms is made using closing abstractions (i.e., abstracting on all free variables). The reduction relation of ΛN is $\longmapsto_{\mathsf{n}}$, and $\Longmapsto_{\mathsf{n}}$ its reflexive and transitive closure.

As bisimilarity for the λ-calculus we consider *environmental bisimilarity* [22, 9], which allows a set of up-to techniques richer than Abramsky's applicative bisimilarity [2], even if the two notions actually coincide, together with contextual equivalence. Environmental bisimilarity makes a clear distinction between the tested terms and the environment. An element of an environmental bisimulation has, in addition to the tested terms M and N, a further component

$\mathcal{E}$, the environment, which expresses the observer's current knowledge. When an input from the observer is required, the arguments supplied are terms that the observer can build using the current knowledge; that is, terms obtained by composing the values in $\mathcal{E}$ using the operators of the calculus. An *environmental relation* is a set of elements each of which is of the form $(\mathcal{E}, M, N)$ or $\mathcal{E}$, and where M, N are closed terms and $\mathcal{E}$ is a relation on closed values. We use $\mathcal{X}, \mathcal{Y}$ to range over environmental relations. In a triple $(\mathcal{E}, M, N)$ the relation component $\mathcal{E}$ is the *environment*, and M, N are the *tested terms*. We write $M\ \mathcal{X}_{\mathcal{E}}\ N$ for $(\mathcal{E}, M, N) \in \mathcal{X}$. We write $\mathcal{E}^\star$ for the closure of $\mathcal{E}$ under contexts. We only define the weak version of the bisimilarity; its strong version is obtained in the expected way.

Definition 5. *An environmental relation $\mathcal{X}$ is an* environmental bisimulation *if*

1. *$M\ \mathcal{X}_{\mathcal{E}}\ N$ implies:*
   (a) if $M \longmapsto_{\mathsf{n}} M'$ then $N \Longmapsto_{\mathsf{n}} N'$ and $M'\ \mathcal{X}_{\mathcal{E}}\ N'$;
   (b) if $M = V$ then $N \Longmapsto_{\mathsf{n}} W$ and $\mathcal{E} \cup \{(V, W)\} \in \mathcal{X}$ (V and W are values);
   (c) the converse of the above two conditions, on N;
2. *if $\mathcal{E} \in \mathcal{X}$ then for all $(\lambda x.P, \lambda x.Q) \in \mathcal{E}$ and for all $(M, N) \in \mathcal{E}^\star$ it holds that $P\{^M/_x\}\ \mathcal{X}_{\mathcal{E}}\ Q\{^N/_x\}$.*

Environmental bisimilarity*, $\approx^{env}$, is the largest environmental bisimulation.*

For the translation of environmental bisimilarity to first-order, a few issues have to be resolved. For instance, an environmental bisimilarity contains both triples $(\mathcal{E}, M, N)$, and pure environments $\mathcal{E}$, which shows up in the difference between clauses (1) and (2) of Definition 5. Moreover, the input supplied to tested terms may be constructed using arbitrary contexts.

We write ΛN^1 for the first-order LTS resulting from the translation of ΛN. The states of ΛN^1 are sequences of λ-terms in which only the last one need not be a value. We use Γ and Δ to range over sequences of values only; thus (Γ, M) indicates a sequence of λ-values followed by M; and Γ_i is the i-th element in Γ.

For an environment $\mathcal{E}$, we write $\mathcal{E}_1$ for an ordered projection of the pairs in $\mathcal{E}$ on the first component, and $\mathcal{E}_2$ is the corresponding projection on the second component. In the translation, intuitively, a triple $(\mathcal{E}, M, N)$ of an environmental bisimulation is split into the two components $(\mathcal{E}_1, M)$ and $(\mathcal{E}_2, N)$. Similarly, an environment $\mathcal{E}$ is split into $\mathcal{E}_1$ and $\mathcal{E}_2$. We write $C[\Gamma]$ for the term obtained by replacing each hole $[\cdot]_i$ in C with the value Γ_i. The rules for transitions in ΛN^1 are as follows:

$$\frac{M \longmapsto_{\mathsf{n}} M'}{(\Gamma, M) \xrightarrow{\tau} (\Gamma, M')} \qquad \frac{\Gamma_i(C[\Gamma]) \longmapsto_{\mathsf{n}} M'}{\Gamma \xrightarrow{i,C} (\Gamma, M')} \tag{2}$$

The first rule says that if M reduces to M' in ΛN then M can also reduce in ΛN^1, in any environment. The second rule implements the observations in clause (2) of Definition 5: in an environment Γ (only containing values), any component Γ_i can be tested by supplying, as input, a term obtained by filling

a context C with values from Γ itself. The label of the transition records the position i and the context chosen. As the rules show, the labels of ΛN^1 include the special label τ, and can also be of the form i, C where i is a integer and C a context.

Theorem 4. *$M \approx^{env}_{\mathcal{E}} N$ iff $(\mathcal{E}_1, M) \approx (\mathcal{E}_2, N)$ and $\mathcal{E} \in \approx^{env}$ iff $\mathcal{E}_1 \approx \mathcal{E}_2$.*

(The theorem also holds for the strong versions of the bisimilarities.) Again, having established full abstraction with respect to a first-order transition system and ordinary bisimilarity, we can inherit the theory of bisimulation enhancements. We have however to check up-to techniques that are specific to environmental bisimilarity. A useful such technique is 'up to environment', which allows us to replace an environment with a larger one; $\mathsf{w}(\mathcal{R})$ is the smallest relation that includes $\mathcal{R}$ and such that, whenever $(V, \Gamma, M)\ \mathsf{w}(\mathcal{R})\ (W, \Delta, N)$ then also $(\Gamma, M)\ \mathsf{w}(\mathcal{R})\ (\Delta, N)$, where V and W are any values. (Here w stands for 'weakening' as, from Lemmas 2 and 4, if $(V, \Gamma, M) \approx (W, \Delta, N)$ then $(\Gamma, M) \approx (\Delta, N)$.)

Lemma 4. *Function w is compatible.*

Somehow dual to weakening is the strengthening of the environment, in which a component of an environment can be removed. However this is only possible if the component removed is 'redundant', that is, it can be obtained by gluing other pieces of the environment within a context; strengthening is captured by the following str function: $(\Gamma, C_v[\Gamma], M)\ \mathsf{str}(\mathcal{R})\ (\Delta, C_v[\Delta], N)$ whenever $(\Gamma, M)\ \mathcal{R}\ (\Delta, N)$ and C_v is a value context (i.e., the outermost operator is an abstraction). We derive the compatibility up to of str in Theorem 5.

For up-to context, we need to distinguish between arbitrary contexts and evaluation contexts. There are indeed substitutivity properties, and corresponding up-to techniques, that only hold for the latter contexts. A hole $[\cdot]_i$ of a context C is in a *redex position* if the context obtained by filling all the holes but $[\cdot]_i$ with values is an evaluation context. Below C ranges over arbitrary contexts, whereas E ranges over contexts whose first hole is in redex position.

$$\begin{aligned}
\mathcal{C}(\mathcal{R}) &\triangleq \big\{((\Gamma, C[\Gamma]), (\Delta, C[\Delta])) \qquad \text{s.t. } \Gamma\ \mathcal{R}\ \Delta \big\} \\
\mathcal{C}_{\mathsf{e}}(\mathcal{R}) &\triangleq \big\{((\Gamma, E[M, \Gamma]), (\Delta, E[N, \Delta])) \text{ s.t. } (\Gamma, M)\ \mathcal{R}\ (\Delta, N)\big\}
\end{aligned}$$

Theorem 5. *The set $\{\mathsf{str}, \mathcal{C}, \mathcal{C}_{\mathsf{e}}\}$ is $\mathbf{sp}$-compatible up to the identity function, and $\mathbf{wp}$-compatible up to $\mathbf{wp} \cup \mathbf{e}$ where $\mathbf{e} \triangleq (\mathcal{R} \mapsto \gtrsim \mathcal{R} \lesssim)$ is 'up to expansion'.*

For the proof, we establish the progression property separately for each function in $\{\mathsf{str}, \mathcal{C}, \mathcal{C}_{\mathsf{e}}\}$, using simple diagram-chasing arguments (together with induction on the structure of a context). Once more, the compatibility of the up to context functions entails also the substitutivity properties of environmental bisimilarity. In [22] the two aspects (substitutivity and up-to context) had to be proved separately, with similar proofs. Moreover the two cases of contexts (arbitrary contexts and evaluation contexts) had to be considered at the same time, within the same proof. Here, in contrast, the machinery of compatible function allows us to split the proof into two simpler proofs.

Remark 2. A transition system ensuring full abstraction as in Theorem 4 does not guarantee the compatibility of the up-to techniques specific to the language in consideration. For instance, a simpler and maybe more natural alternative to the second transition in (2) is the following one:

$$\frac{}{\Gamma \xrightarrow{i,C} (\Gamma, \Gamma_i(C[\Gamma]))} \tag{3}$$

With this rule, full abstraction holds, but up-to context is unsound: for any Γ and Δ, the singleton relation $\{(\Gamma, \Delta)\}$ is a bisimulation up to $\mathcal{C}$: indeed, using rule (3), the derivatives of the pair Γ, Δ are of the shape $\Gamma_i(C[\Gamma])$, $\Delta_i(C[\Delta])$, and they can be discarded immediately, up to the context $[\cdot]_i C$. If up-to context were sound then we would deduce that any two terms are bisimilar. (The rule in (2) prevents such a behaviour since it ensures that the tested values are 'consumed' immediately.)

5 Imperative call-by-value λ-calculus

In this section we study the addition of imperative features (higher-order references, that we call locations), to a call-by-value λ-calculus. It is known that finding powerful reasoning techniques for imperative higher-order languages is a hard problem. The language, ΛR, is a simplified variant of that in [10, 22]. The syntax of terms, values, and evaluation contexts, as well as the reduction semantics are given in Figure 1. A λ-term M is run in a *store*: a partial function from locations to closed values, whose domain includes all free locations of both M and its own co-domain. We use letters s, t to range over stores. New store locations may be created using the operator $\nu\ell\, M$; the content of a store location ℓ may be rewritten using $\mathsf{set}_\ell V$, or read using $\mathsf{get}_\ell V$ (the former instruction returns a value, namely the identity $I \triangleq \lambda x.x$, and the argument of the latter one is ignored). We denote the reflexive and transitive closure of $\longmapsto_{\mathsf{R}}$ by $\Longmapsto_{\mathsf{R}}$.

Note that in contrast with the languages in [10, 22], locations are not directly first-class values; the expressive power is however the same: a first-class location ℓ can always be encoded as the pair $(\mathsf{get}_\ell, \mathsf{set}_\ell)$.

We present the first-order LTS for ΛR, and then we relate the resulting strong and weak bisimilarities directly with contextual equivalence (the reference equivalence in λ-calculi). Alternatively, we could have related the first-order bisimilarities to the environmental bisimilarities of ΛR, and then inferred the correspondence with contextual equivalence from known results about environmental bisimilarity, as we did for ΛN.

We write $(s; M) \downarrow$ when M is a value; and $(s; M) \Downarrow$ if $(s; M) \Longmapsto_{\mathsf{R}} \downarrow$. For the definition of contextual equivalence, we distinguish the cases of values and of arbitrary terms, because they have different substitutivity properties: values can be tested in arbitrary contexts, while arbitrary terms must be tested only in evaluation contexts. As in [22], we consider contexts that do not contain free locations (they can contain bound locations). We refer to [22] for more details on these aspects.

$$M ::= x \mid MM \mid \nu\ell\, M \mid V \qquad V ::= \lambda x.M \mid \mathsf{set}_\ell \mid \mathsf{get}_\ell \qquad E ::= [\cdot] \mid EV \mid ME$$

$$\frac{}{(s;(\lambda x.M)V) \longmapsto_{\mathsf{R}} (s; M\{V/x\})} \qquad \frac{\ell \notin \mathsf{dom}(s)}{(s;\nu\ell\, M) \longmapsto_{\mathsf{R}} (s[\ell \mapsto I]; M)}$$

$$\frac{\ell \in \mathsf{dom}(s)}{(s;\mathsf{get}_\ell V) \longmapsto_{\mathsf{R}} (s; s[\ell])} \qquad \frac{\ell \in \mathsf{dom}(s)}{(s;\mathsf{set}_\ell V) \longmapsto_{\mathsf{R}} (s[\ell \mapsto V]; I)}$$

$$\frac{(s;M) \longmapsto_{\mathsf{R}} (s';M')}{(s;E[M]) \longmapsto_{\mathsf{R}} (s';E[M'])}$$

Fig. 1. The imperative λ-calculus

Definition 6. – *For values V, W, we write $(s;V) \equiv (t;W)$ when $(s;C[V])\Downarrow$ iff $(t;C[W])\Downarrow$, for all location-free context C.*
– *For terms M and N, we write $(s;M) \equiv (t;N)$ when $(s;E[M])\Downarrow$ iff $(t;E[N])\Downarrow$, for all location-free evaluation context E.*

We now define ΛR^1, the first-order LTS for ΛR. The states and the transitions for ΛR^1 are similar to those for the pure λ-calculus of Section 4, with the addition of a component for the store. The two transitions (2) of call-by-name λ-calculus become:

$$\frac{(s;M) \longmapsto_{\mathsf{R}} (s';M')}{(s;\Gamma,M) \xrightarrow{\tau} (s';\Gamma,M')} \qquad \frac{\Gamma' = \Gamma, \mathsf{getset}(r) \quad (s \uplus r[\Gamma'];\Gamma_i(C[\Gamma'])) \longmapsto_{\mathsf{R}} (s';M')}{(s;\Gamma) \xrightarrow{i,C,\mathtt{cod}(r)} (s';\Gamma',M')}$$

The first rule is the analogous of the first rule in (2). The important differences are on the second rule. First, since we are *call-by-value*, C now ranges over $\mathbb{C}_v$, the set of *value contexts* (i.e., contexts of the form $\lambda x.C'$) without free locations. Moreover, since we are now *imperative*, in a transition we must permit the creation of new locations, and a term supplied by the environment should be allowed to use them. In the rule, the new store is represented by r (whose domain has to be disjoint from that of s). Correspondingly, to allow manipulation of these locations from the observer, for each new location ℓ we make set_ℓ and get_ℓ available, as an extension of the environment; in the rule, these are collectively written $\mathsf{getset}(r)$, and Γ' is the extended environment. Finally, we must initialise the new store, using terms that are created out of the extended environment Γ'; that is, each new location ℓ is initialised with a term $D_\ell[\Gamma']$ (for $D_\ell \in \mathbb{C}_v$). Moreover, the contexts D_ℓ chosen must be made visible in the label of the transition. To take care of these aspects, we view r as a *store context*, a tuple of assignments $\ell \mapsto D_\ell$. Thus the initialisation of the new locations is written $r[\Gamma']$; and, denoting by $\mathtt{cod}(r)$ the tuple of the contexts D_ℓ in r, we add $\mathtt{cod}(r)$ to the label of the transition. Note also that, although C and D_ℓ are location-free, their holes may be instantiated with terms involving the set_ℓ and get_ℓ operators, and these allow manipulation of the store.

Once more, on the (strong and weak) bisimilarities that are derived from this first-order LTS we can import the theory of compatible functions and bisimulation enhancements. Concerning additional up-to functions, specific to ΛR, the functions w, str, $\mathcal{C}$ and $\mathcal{C}_\mathsf{e}$ are adapted from Section 4 in the expected manner—contexts C_v, C and E must be location-free. A further function for ΛR is store, which manipulates the store by removing locations that do not appear elsewhere (akin to garbage collection); thus, $\mathsf{store}(\mathcal{R})$ is the set of all pairs

$$((s \uplus r[\Gamma']; \Gamma', M),\ (t \uplus r[\Delta']; \Delta', N))$$

such that $(s; \Gamma, M)\ \mathcal{R}\ (t; \Delta, N)$, and with $\Gamma' = \Gamma, \mathsf{getset}(r)$ and $\Delta' = \Delta, \mathsf{getset}(r)$.

Lemma 5. *The set* $\{\mathsf{w}, \mathsf{str}, \mathcal{C}_\mathsf{e}, \mathsf{store}, \mathcal{C}\}$ *is* **sp***-compatible up to the identity function and is* **wp***-compatible up to* $\mathbf{wp} \cup \mathbf{e}$.

The techniques $\mathcal{C}$ and $\mathcal{C}_\mathsf{e}$ allow substitutivity under location-free contexts, from which we can derive the soundness part of Theorem 6.

Theorem 6. $(s; M) \equiv (t; N)$ *iff* $(s; M) \approx (t; N)$.

Proof (sketch). Soundness ($\Leftarrow$) follows from congruence by $\mathcal{C}_\mathsf{e}$ (Lemmas 5 and 2) and completeness ($\Rightarrow$) is obtained by standard means. See Appendix D for details.

Note that substitutivity of bisimilarity is restricted either to values ($\mathcal{C}$), or to evaluation contexts ($\mathcal{C}_\mathsf{e}$). The following lemma provides a sufficient condition for a given law between arbitrary terms to be preserved by arbitrary contexts.

Lemma 6. *Let* $\asymp$ *be any of the relations* $\sim, \approx$, *and* $\gtrsim$. *Suppose* L, R *are* ΛR *terms with* $(s; \Gamma, L) \asymp (s; \Gamma, R)$ *for all environments* Γ *and stores* s. *Then also* $(s; \Gamma, C[L]) \asymp (s; \Gamma, C[R])$, *for any store* s, *environment* Γ *and context* C.

Proof (sketch). We first prove a simplified result in which C is an evaluation context, using techniques $\mathcal{C}_\mathsf{e}$ and store. We then exploit this partial result together with up-to expansion to derive the general result. See Appendix D for more details.

We use this lemma at various places in the example we cover in Section 6. For instance we use it to replace a term $N_1 \triangleq (\lambda x.E[x])M$ (with E an evaluation context) with $N_2 \triangleq E[M]$, under an arbitrary context. Such a property is delicate to prove, even for closed terms, because the evaluation of M could involve reading from a location of the store that itself could contain occurrences of N_1 and N_2.

6 An example

We conclude by discussing an example from [10]. It consists in proving a law between terms of ΛR extended with integers, operators for integer addition and

subtraction, and a conditional—those constructs are straightforward to accommodate in the presented framework. For readability, we also use the standard notation for store assignment, dereferencing and sequence: $(\ell := M) \triangleq \mathsf{set}_\ell M$, $!\ell \triangleq \mathsf{get}_\ell I$, and $M; N \triangleq (\lambda x.N)M$ where x does not appear in N. The two terms are the following ones:

- $M \triangleq \lambda g.\nu \ell\ \ell := 0; g(\mathsf{incr}_\ell); \mathtt{if}\ !\ell\ \mathtt{mod}\ 2 = 0\ \mathtt{then}\ I\ \mathtt{else}\ \Omega$
- $N \triangleq \lambda g.g(F); I$,

where $\mathsf{incr}_\ell \triangleq \lambda z.\ell := !\ell + 2$, and $F \triangleq \lambda z.I$. Intuitively, those two terms are weakly bisimilar because the location bound by ℓ in the first term will always contain an even number.

This example is also considered in [22] where it is however modified to fit the up-to techniques considered in that paper. The latter are less powerful than those available here thanks to the theory of up-to techniques for first-order LTSs (e.g., up to expansion is not considered in [22]—its addition to environmental bisimulations is non-trivial, having stores and environments as parameters).

We consider two proofs of the example. In comparison with the proof in [22]: (i) we handle the original example from [10], and (ii) the availability of a broader set of up-to techniques and the possibility of freely combining them allows us to work with smaller relations. In the first proof we work up to the store (through the function store) and up to expansion—two techniques that are not available in [22]. In the second proof we exploit the up-to-transitivity technique of Section 2, which is only sound for strong bisimilarity, to further reduce the size of the relation we work with.

First proof. We first employ Lemma 6 to reach a variant similar to that of [22]: we make a 'thunk' out of the test in M, and we make N look similar. More precisely, let $\mathsf{test}_\ell \triangleq \lambda z.\mathtt{if}\ !\ell\ \mathtt{mod}\ 2 = 0\ \mathtt{then}\ I\ \mathtt{else}\ \Omega$, we first prove that

- $M \approx M' \triangleq \lambda g.\nu \ell\ \ell := 0; g(\mathsf{incr}_\ell); \mathsf{test}_\ell I$, and
- $N \approx N' \triangleq \lambda g.g(F); FI$.

It then suffices to prove that $M' \approx N'$, which we do using the following relation:

$$\mathcal{R} \triangleq \left\{ \left(s, M', (\mathsf{incr}_\ell, \mathsf{test}_\ell)_{\ell \in \tilde{\ell}}\right), \left(\emptyset, N', (F, F)_{\ell \in \tilde{\ell}}\right) \ \text{ s.t. } \forall \ell \in \tilde{\ell}, \quad s(\ell) \text{ is even} \right\} .$$

The initial pair of terms is generalised by adding any number of private locations, since M' can use itself to create more of them. Relation $\mathcal{R}$ is a weak bisimulation up to store, $\mathcal{C}$ and expansion. More details can be found in Appendix E.

Second proof. Here we also preprocess the terms using Lemma 6, to add a few artificial internal steps to N, so that we can carry out the reminder of the proof using strong bisimilarity, which enjoys more up-to techniques than weak bisimilarity:

- $M \approx M' \triangleq \lambda g.\nu \ell\ \ell := 0; g(\mathsf{incr}_\ell); \mathsf{test}_\ell I$,
- $N \approx N'' \triangleq \lambda g.I; I; g(\mathsf{incr}_0); \mathsf{test}_0 I$.

where incr_0 and test_0 just return I on any input, taking the same number of internal steps as incr_ℓ and test_ℓ. We show that $M' \sim N''$ by proving that the following relation $\mathcal{R}$ is a strong bisimulation *up to unfolding, store, weakening, strengthening, transitivity and context* (a technique unsound in the weak case):

$$\mathcal{S} \triangleq \{(M', N'')\} \cup \{(\ell \mapsto 2n, \mathsf{incr}_\ell, \mathsf{test}_\ell), (\emptyset, \mathsf{incr}_0, \mathsf{test}_0) \;\text{ s.t. }\; n \in \mathbb{N}\}$$

This relation uses a single location; there is one pair for each integer that can be stored in the location. In the diagram-chasing arguments for $\mathcal{S}$, essentially a pair of derivatives is proved to be related under the function

$$\mathbf{sp} \circ \mathbf{sp} \circ \mathsf{star} \circ (\mathsf{str} \cup \mathsf{store} \cup \mathcal{C} \cup \mathsf{w})^\omega$$

where $\mathsf{star} : \mathcal{R} \mapsto \mathcal{R}^\star$ is the reflexive-transitive closure function. (Again, we refer to Appendix E for more details.)

The difference between the relation $\mathcal{R}$ in the first proof and the proofs in [10, 22] is that $\mathcal{R}$ only requires locations that appear free in the tested terms; in contrast, the relations in [10, 22] need to be closed under all possible extensions of the store, including extensions in which related locations are mapped onto arbitrary context-closures of related values. We avoid this thanks to the up-to store function. The reason why, both in [10, 22] and in the first proof above, several locations have to be considered is that, with bisimulations akin to environmental bisimulation, the input for a function is built using the values that occur in the candidate relation. In our example, this means that the input for a function can be a context-closure of M and N; hence uses of the input may cause several evaluations of M and N, each of which generates a new location. In this respect, it is surprising that our second proof avoids multiple allocations (the candidate relation $\mathcal{S}$ only mentions one location). This is due to the massive combination of up-to techniques whereby, whenever a new location is created, a double application of up to context (the 'double' is obtained from up-to transitivity) together with some administrative work (given by the other techniques) allows us to absorb the location.

Acknowledgement

The authors acknowledge support from the ANR projects 2010-BLAN-0305 PiCoq and 12IS02001 PACE.

References

1. M. Abadi and A.D. Gordon. A bisimulation method for cryptographic protocols. In Chris Hankin, editor, *ESOP'98*, volume 1381 of *LNCS*, pages 12–26. Springer, 1998.
2. S. Abramsky. The lazy lambda calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1989.

3. S. Arun-Kumar and M. Hennessy. An efficiency preorder for processes. *Acta Informatica*, 29:737–760, 1992.
4. K. Chaudhuri, M. Cimini, and D. Miller. Formalization of the bisimulation-up-to technique and its meta theory. Draft, 2014.
5. D. Hirschkoff. A full formalisation of pi-calculus theory in the calculus of constructions. In *TPHOLs*, volume 1275 of *LNCS*, pages 153–169. Springer, 1997.
6. C.-K. Hur, G. Neis, D. Dreyer, and V. Vafeiadis. The power of parameterization in coinductive proof. In *POPL*, pages 193–206. ACM, 2013.
7. A. Jeffrey and J. Rathke. Towards a theory of bisimulation for local names. In *LICS*, pages 56–66, 1999.
8. V. Koutavas and M. Hennessy. First-order reasoning for higher-order concurrency. *Computer Languages, Systems & Structures*, 38(3):242–277, 2012.
9. V. Koutavas, P. B. Levy, and E. Sumii. From applicative to environmental bisimulation. *Electr. Notes Theor. Comput. Sci.*, 276:215–235, 2011.
10. V. Koutavas and M. Wand. Small bisimulations for reasoning about higher-order imperative programs. In *POPL'06*, pages 141–152. ACM, 2006.
11. S.B. Lassen. Relational reasoning about contexts. In *Higher-order operational techniques in semantics*, pages 91–135. Cambridge University Press, 1998.
12. S.B. Lassen. *Relational Reasoning about Functions and Nondeterminism*. PhD thesis, Department of Computer Science, University of Aarhus, 1998.
13. S.B. Lassen. Bisimulation in untyped lambda calculus: Böhm trees and bisimulation up to context. *Electr. Notes Theor. Comput. Sci.*, 20:346–374, 1999.
14. M. Lenisa. *Themes in Final Semantics*. Ph.D. thesis, Università di Pisa, 1998.
15. M. Merro and F. Zappa Nardelli. Behavioral theory for mobile ambients. *J. ACM*, 52(6):961–1023, 2005.
16. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
17. J.Å. Pohjola and J. Parrow. Bisimulation up-to techniques for psi-calculi. Draft, 2014.
18. D. Pous. *Techniques modulo pour les bisimulations*. Phd thesis, École Normale Supérieure de Lyon, February 2008.
19. D. Pous and D. Sangiorgi. Enhancements of the bisimulation proof method. In *Advanced Topics in Bisimulation and Coinduction*. Cambridge University Press, 2012.
20. J. Rot, M. Bonsangue, and J. Rutten. Coalgebraic bisimulation-up-to. In *SOFSEM'13*, volume 7741 of *LNCS*, pages 369–381. Springer, 2013.
21. D. Sangiorgi. On the bisimulation proof method. *J. of MSCS*, 8:447–479, 1998.
22. D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. *ACM Trans. Program. Lang. Syst.*, 33(1):5, 2011.
23. D. Sangiorgi and D. Walker. *The Pi-Calculus: a theory of mobile processes*. Cambridge University Press, 2001.
24. E. Sumii and B. C. Pierce. A bisimulation for dynamic sealing. *Theor. Comput. Sci.*, 375(1-3):169–192, 2007.
25. E. Sumii and B. C. Pierce. A bisimulation for type abstraction and recursion. *J. ACM*, 54(5), 2007.
26. N.D. Turner. *The polymorphic pi-calculus: Theory and Implementation*. PhD thesis, Department of Computer Science, University of Edinburgh, 1996.

A First-order bisimulation and up-to techniques

Definition 7 (weak bisimilarity). *A relation $\mathcal{R}$ on processes is a* weak bisimulation *if whenever $P \mathcal{R} Q$:*

- *if $P \xrightarrow{\mu} P'$ then $Q \stackrel{\hat{\mu}}{\Longrightarrow} Q'$ and $P' \mathcal{R} Q'$;*
- *if $Q \xrightarrow{\mu} Q'$ then $P \stackrel{\hat{\mu}}{\Longrightarrow} P'$ and $P' \mathcal{R} Q'$.*

We write $\approx$ for the largest weak bisimulation, and call it weak bisimilarity.

Definition 8 (expansion). *A relation $\mathcal{R}$ on processes is an* expansion relation *if whenever $P \mathcal{R} Q$:*

- *if $P \xrightarrow{\mu} P'$ then $Q \xrightarrow{\hat{\mu}} Q'$ and $P' \mathcal{R} Q'$.*
- *if $Q \xrightarrow{\mu} Q'$ then $P \stackrel{\mu}{\Longrightarrow} P'$ and $P' \mathcal{R} Q'$;*

We write $\gtrsim$ for the largest expansion relation, and simply call it expansion.

B The π-calculus

The syntax of the π-calculus is the following:

$$P ::= 0 \mid a(b).P \mid \overline{a}b.P \mid P|P \mid \boldsymbol{\nu} a\, P \mid !P$$

(other operators, such as matching and mismatching, could be added). The operational semantics is described by the rules for $\longmapsto_\pi$ below. We assume that α-convertible terms are identified. The grammar of labels is $\mu ::= \tau \mid ab \mid \overline{a}b \mid \overline{a}(b)$.

$$\text{OUT}\ \frac{}{\overline{a}b.P \overset{\overline{a}b}{\longmapsto}_\pi P} \qquad \text{INP}\ \frac{}{a(b).P \overset{ac}{\longmapsto}_\pi P\{c/b\}} \qquad \text{SUM-L}\ \frac{P \overset{\mu}{\longmapsto}_\pi P'}{P + Q \overset{\mu}{\longmapsto}_\pi P'}$$

$$\text{PAR-L}\ \frac{P \overset{\mu}{\longmapsto}_\pi P'}{P \mid Q \overset{\mu}{\longmapsto}_\pi P' \mid Q}\ \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$$

$$\text{COMM-L}\ \frac{P \overset{\overline{a}b}{\longmapsto}_\pi P' \qquad Q \overset{ab}{\longmapsto}_\pi Q'}{P \mid Q \overset{\tau}{\longmapsto}_\pi P' \mid Q'}$$

$$\text{CLOSE-L}\ \frac{P \overset{\overline{a}(b)}{\longmapsto}_\pi P' \qquad Q \overset{ab}{\longmapsto}_\pi Q'}{P \mid Q \overset{\tau}{\longmapsto}_\pi \boldsymbol{\nu} b\, (P' \mid Q')}\ b \notin \mathrm{fn}(Q) \qquad \text{RES}\ \frac{P \overset{\mu}{\longmapsto}_\pi P'}{\boldsymbol{\nu} a\, P \overset{\mu}{\longmapsto}_\pi \boldsymbol{\nu} a\, P'}\ a \notin \mathrm{n}(\mu)$$

$$\text{OPEN}\ \frac{P \overset{\overline{a}b}{\longmapsto}_\pi P'}{\boldsymbol{\nu} b\, P \overset{\overline{a}(b)}{\longmapsto}_\pi P'}\ a \neq b \qquad \text{REP}\ \frac{P \mid !P \overset{\mu}{\longmapsto}_\pi P'}{!P \overset{\mu}{\longmapsto}_\pi P'}$$

Definition 9 (Bisimilarity in π). *A relation $\mathcal{R}$ is a strong early bisimulation in the π-calculus if, whenever $P \mathcal{R} Q$:*

1. *if* $P \overset{\overline{a}(b)}{\longmapsto}_\pi P'$ *and* $b \notin \mathrm{fn}(Q)$ *then* $Q \overset{\overline{a}(b)}{\longmapsto}_\pi Q'$ *for some* Q' *such that* $P' \mathcal{R} Q'$,
2. *if* $P \overset{\mu}{\longmapsto}_\pi P'$ *and* μ *is not a bound output, then* $Q \overset{\mu}{\longmapsto}_\pi Q'$ *for some* Q' *such that* $P' \mathcal{R} Q'$,
3. *the converse of (1) and (2), on* Q.

Early bisimilarity, $\sim^{\mathrm{e}}$*, is the union of all early bisimulations.*

The weak version of early bisimilarity, *weak early bisimilarity*, written $\approx^{\mathrm{e}}$, is obtained in the standard way: the transition $Q \overset{\overline{a}(b)}{\longmapsto}_\pi Q'$ in clause (1) is replaced by $Q \overset{\overline{a}(b)}{\Longmapsto}_\pi Q'$; and similarly the transition $Q \overset{\mu}{\longmapsto}_\pi Q'$ in (2) is replaced by $Q \overset{\hat{\mu}}{\Longrightarrow} Q'$.

Proof excerpts for Theorem 2 and and 3: Theorem 2 is proven modularly, by proving a progression for each initial context function. The progressions for Theorem 3 are similar, except general choice is not compatible in the weak case, and we need one more up-to technique for case of the replication. We write $\mathbf{p}$ when the progression stands for both $\mathbf{sp}$ and $\mathbf{wp}$. The proofs largely follows the ones from [18], the differences accounting for the cases when extrusion is involved. In some of those cases we use the notation $N \triangleq (\mathsf{str} \circ \mathcal{C}_\nu) \cup \mathsf{id}$.

- $\mathcal{C}_o \rightsquigarrow_{\mathbf{p}} \mathsf{id}$,
- $\mathcal{C}_i \rightsquigarrow_{\mathbf{p}} \mathsf{str}$,
- $\mathcal{C}_\nu \rightsquigarrow_{\mathbf{p}} \mathcal{C}_\nu \cup \mathsf{isub}$,
- $\mathcal{C}_+ \rightsquigarrow_{\mathbf{sp}} \mathsf{id}$,
- $\mathcal{C}_{g+} \rightsquigarrow_{\mathbf{wp}} \mathsf{str}$,
- $\mathcal{C}_| \rightsquigarrow_{\mathbf{p}} N \circ \mathcal{C}_|$,
- $\mathcal{C}_! \rightsquigarrow_{\mathbf{sp}} \mathcal{C}_|^\omega \circ N \circ \mathcal{C}_|^\omega \circ (\mathcal{C}_! \cup \mathsf{str}))$,
- $\mathcal{C}_! \rightsquigarrow_{\mathbf{wp}} \mathcal{C}_|^\omega \circ N \circ \mathcal{C}_|^\omega \circ (\mathcal{C}_! \cup \mathsf{str})) \cup \mathcal{C}_|^\omega \circ (\mathsf{id} \cup \mathcal{C}_!)$.

In the last two cases we see twice the use of $\mathcal{C}_|^\omega$, while only one is necessary in CCS. This is because in the π-calculus, Lemma 5.17 from [18] characterising the transitions from a replicated process, becomes the following: if $!P \xrightarrow{\alpha} Q$ then one of the following holds:

- $Q = !P \,|\, P_0 \,|\, P \,|\, \ldots \,|\, P$ with $P \xrightarrow{\alpha} P_0$, or when $\alpha = \tau$ it can be as well:
- $Q = !P \,|\, P_0 \,|\, P \,|\, \ldots \,|\, P \,|\, P_1 \,|\, P \,|\, \ldots \,|\, P$ with $P \xrightarrow{\overline{a}b} P_i$ and $P \xrightarrow{ab} P_{1-i}$ or
- $Q = (\nu b)(!P \,|\, P_0 \,|\, P \,|\, \ldots \,|\, P \,|\, P_1) \,|\, P \,|\, \ldots \,|\, P$ with $P \xrightarrow{\overline{a}(b)} P_i$ and $P \xrightarrow{ab} P_{1-i}$,

the last case being unnecessary in the case of CCS.

C The λ-calculus

The set Λ of pure λ-terms is defined by:

$$M, N ::= x \mid \lambda x.M \mid MN$$

We write Λ^0 for the subset of closed terms. The *call-by-name reduction relation* $\longmapsto_{\mathsf{n}}$ is the least relation over Λ^0 that is closed under the following rules.

$$\frac{}{(\lambda x.M)N \longmapsto_{\mathsf{n}} M\{^{N}\!/_{x}\}} \qquad \frac{M \longmapsto_{\mathsf{n}} M'}{MN \longmapsto_{\mathsf{n}} M'N}$$

We write $\Longmapsto_{\mathsf{n}}$ for the reflexive and transitive closure of $\longmapsto_{\mathsf{n}}$. The values are the terms of the form $\lambda x.M$. In call-by-name *evaluation contexts* are described by the following grammar:

$$C := CM \mid [\cdot]$$

(Symbol C is used also for arbitrary contexts; it will be explicitly indicated when C refers to evaluation contexts.)

D Imperative call-by-value λ-calculus

Here we give more details on a few results for the imperative λ-calculus ΛR. Specifically, first the relationship between contextual equivalence in ΛR and bisimilarity on the target first-order LTS (Theorem 6 of the main text); then Lemma 6 of the main text, and then Theorem 7 that is used in examples (as an instantiation of Lemma 6).

Lemma 7. *If* $(s; M) \approx (t; N)$ *then* $(s; M) \equiv (t; N)$.

Proof. Let E be an evaluation context. Since $\mathcal{C}_{\mathsf{e}}$ is compatible up to (Lemma 5) by Lemma 2 we know $\approx$ is a $\mathcal{C}_{\mathsf{e}}$-congruence, hence $(s; E[M]) \approx (t; E[N])$.

Suppose now $(s; E[M])$ reaches a value. Then we can derive a $\overset{i,C,\mathtt{cod}(r)}{\Longrightarrow}$ transition from it, and by weak bisimulation we can derive the same transition from $(t; E[N])$, meaning it also reaches a value. This means $(s; E[M])\Downarrow$ implies $(t; E[N])\Downarrow$ and we conclude by symmetry.

Lemma 8. *If* $(s; M) \equiv (t; N)$ *then* $(s; M) \approx (t; N)$.

Proof. We prove a slightly stronger result, that the relation $\mathcal{R}$ below is a weak bisimulation (E ranges over location-free evaluation contexts).

$$\mathcal{R} \triangleq \big\{((s; \Gamma, M), (t; \Delta, N)) \text{ s.t. } \forall E\ (s; E[M, \Gamma])\Downarrow \text{ iff } (t; E[N, \Delta])\Downarrow\big\} \qquad (4)$$

Case 1: τ transition. When $(s; M) \longmapsto_{\mathsf{R}} (s'; M)$ we can easily see that $(s; E[M, \Gamma])\Downarrow$ iff $(s'; E[M', \Gamma])\Downarrow$ by determinism of $\longmapsto_{\mathsf{R}}$, so we have in fact $(s'; \Gamma, M')\ \mathcal{R}\ (t; \Delta, N)$ and the transition $\xrightarrow{\tau}$ is caught up with no transition at all.

Case 2: $i, C, \mathtt{cod}(r)$ transition from $(s; \Gamma, M)$ to $(s'; \Gamma'', M')$. (verifying $(*)$ below) means that M is a value V and thus $(s; M)\Downarrow$. By choosing $E = [\cdot]_1$ and $u = \emptyset$ in (4) we know that $(t; N)\Downarrow$ and thus $(t; N) \Longmapsto_{\mathsf{R}} (t'; W)$ for some value W.

Hence, we easily get the weak transition $(t; \Delta, N) \overset{i,C,\mathsf{cod}(r)}{\Longrightarrow} (t''; \Delta'', N')$ through $(t'; \Delta, W)$, verifying $(**)$ below.

$$\Gamma' = \Gamma, V \qquad \Gamma'' = \Gamma', \mathsf{getset}(r) \qquad (r[\Gamma''] \uplus s; \Gamma_i'(C[\Gamma''])) \longmapsto_{\mathsf{R}} (s'; M') \qquad (*)$$
$$\Delta' = \Delta, W \qquad \Delta'' = \Delta', \mathsf{getset}(r) \qquad (r[\Delta''] \uplus t'; \Delta_i'(C[\Delta''])) \longmapsto_{\mathsf{R}} (t''; N') \qquad (**)$$

We prove now $(s'; \Gamma'', M') \mathcal{R} (t''; \Delta'', N')$. Let E be any location-free evaluation context, we prove:

$$(s'; E[M', \Gamma'']) \Downarrow \quad \text{iff} \quad (t''; E[N', \Delta'']) \Downarrow \quad . \tag{5}$$

Backtracking one step using $(*)$ and $(**)$ it is enough to prove

$$(r[\Gamma''] \uplus s; E[\Gamma_i'(C[\Gamma'']), \Gamma'']) \Downarrow \quad \text{iff} \quad (r[\Gamma''] \uplus t'; E[\Delta_i'(C[\Delta'']), \Delta'']) \Downarrow \tag{6}$$

which we do by exhibiting an evaluation context F for which we already have (by instantiating with F the definition of $\mathcal{R}$) the equation (7) below, and that each member of (6) is a derivative of the corresponding member of (7).

$$(s; F[M, \Gamma]) \Downarrow \quad \text{iff} \quad (t; F[N, \Delta]) \Downarrow \tag{7}$$

For that we choose

$$F \triangleq \ \mathtt{let}\ x = [\cdot]_1\ \mathtt{in}\ \nu\ell_1 \ldots \nu\ell_n\ \ell_1 := C_1^\bullet;\ \ \ldots; \ell_n := C_n^\bullet;\ E^\bullet[[\cdot]_i^\bullet(C^\bullet), -]$$

where r is the collection of $\ell_i \mapsto C_i$ and we write $D^\bullet$ for a context D where the holes destined to get_{ℓ_i} and set_{ℓ_i} are already filled, and the holes destined to get the values V and W are filled with x.

The lemma below is Lemma 6 of the main text.

Lemma 9. *Let $\asymp$ be any of the relations $\sim, \approx$, and $\gtrsim$. Suppose L, R are ΛR terms with $(s; \Gamma, L) \asymp (s; \Gamma, R)$ for all environments Γ and stores s. Then also $(s; \Gamma, C[L]) \asymp (s; \Gamma, C[R])$, for any store s, environment Γ and context C.*

We give the proof for $\gtrsim$ as it is the most general case. Also remark that the last Γ is not necessary, as it can be encoded into the C.

The proof goes as follows: (1) we first prove a simplified result in which the context C is an evaluation context, using techniques $\mathcal{C}_{\mathsf{e}}$ and store. (2) We then exploit (1) to derive another partial result where C is a context whose holes are *not* in evaluation position, and achieve the proof using up to expansion and (1) when a hole is freed.

These classes of contexts are enough to cover all cases and the proofs (1) and (2) focus on very different parts of the problem. This separation is necessary: side effects of the store would make quite convoluted a naive bisimulation candidate, on which case analyses prove difficult.

Lemma 10 handles (1) and Lemma 11 handles (2).

Lemma 10. *Suppose that for all s and Γ, we have $(s;\Gamma,L) \gtrsim (s;\Gamma,R)$. Then for all s, Γ and evaluation context F with free locations, $(s;\Gamma,F[L]) \gtrsim (s;\Gamma,F[R])$.*

Proof. Let A be the list of set_ℓ and get_ℓ for all location ℓ in F. Then it is easy to get F' from F such that $F = F'[-,A]$ and F' is location-free. By hypothesis we know $(s;\Gamma,A,L) \gtrsim (s;\Gamma,A,R)$ on which we apply precongruence for evaluation contexts $\mathcal{C}_\mathsf{e}$ to get $(s;\Gamma,A,F'[L,A]) \gtrsim (s;\Gamma,A,F'[R,A])$. By weakening w we get $(s;\Gamma,F'[L,A]) \gtrsim (s;\Gamma,F'[R,A])$.

Lemma 11. *Let L, R be ΛR terms with $(s;\Gamma,L) \gtrsim (s;\Gamma,R)$ for all environment Γ and store s. Then Suppose C is a multi hole context, such that no hole is in evaluation position in C. Then for all store s we know $(s;C[L]) \gtrsim (s;C[R])$.*

Proof. Let $\mathcal{R}$ relate each configuration $((\ell \mapsto C_v^\ell[L])_\ell; \tilde{C}_v[L], C[L])$ to the one where R replaces L: $((\ell \mapsto C_v^\ell[R])_\ell; \tilde{C}_v[R], C[R])$ for all $(C_v^\ell)_\ell$ and $\tilde{C}_v$ families of value contexts (of the form $\lambda x.C'$), and C context without any hole in evaluation position. For short we write s_L, s_R, Γ_L, and Γ_R the corresponding stores and environments. We run simultaneously the transitions from both sides $(s_L;\Gamma_L,C[L])$ and $(s_R;\Gamma_R,C[R])$ as they have always the same shape.

We prove $\mathcal{R}$ is an expansion relation up to expansion. We rely on the fact that L and R will never be run directly in this proof.

Case 1: τ action. Since in L in $C[L]$ (and R in $C[R]$) is not in evaluation position, both sides will do the same kind of transition, being completely oblivious to L/R. The resulting configurations will be $(s'_L;\Gamma_L,C'[L])$ and $(s'_R;\Gamma_R,C'[R])$. (Even if a set_ℓ or a get_ℓ is involved, L/R part may go to or from the store, but this will keep the same shape.)

The only part of the invariant of the relation that is not maintained is that L/R may appear in evaluation position, if $C'[L] = E[L,L]$ (where $[\cdot]_1$ is in evaluation position and $[\cdot]_2$ may appear everywhere).

In this case, we remark that $F \triangleq E[-,L]$ is an evaluation context, on which we can apply Lemma 10 to have $(s'_L;\Gamma_L,E[L,L]) \gtrsim (s'_L;\Gamma_L,E[R,L])$ and now since R is not in evaluation position, the context $C_2 = E[R,-]$ is a context with no hole in evaluation position, hence $(s'_L;\Gamma_L,E[R,L])\ \mathcal{R}\ (s'_R;\Gamma_R,E[R,R])$ and we have closed the diagram.

Note that it may happen that even if $E[L,-]$ doesn't have holes in evaluation position, $E[R,-]$ does. In this case, we just use Lemma 10 while there are still such holes, and the progression to $\gtrsim\mathcal{R}$ still holds ($\gtrsim$ is transitive).

Case 2: visible action. First, $C[L]$ is a value iff $C[R]$ is a value so they have the same visible actions of the form $i, D, \mathtt{cod}(r)$. We end up in the same shape of configurations we had for the τ transition above, and proceed the same to close the diagram.

Finally we have proven that $\mathcal{R}$ progresses to $\gtrsim\mathcal{R}$ (expansion up to expansion). In the strong case, we would have proven $\mathcal{R}$ progresses to $\sim\mathcal{R}$, and in the weak case we would have proven that it progresses to both $\approx\mathcal{R}$ *and* $\mathcal{R}\approx$, which is necessary because in the weak case, one can use "up to $\approx$" only when $\approx$ is not on the same side as the challenge.

In the following $\mathcal{R}^+$ is the transitive closure of $\mathcal{R}$. We prove here Theorem 7 (using Lemma 12) since we use (simple instances of) it in Section 6.

Lemma 12. *Suppose that E and E' are evaluation contexts and that for all value V value and store s, we have $(s; E[V]) \longmapsto_{\mathsf{R}}^{+} (s; E'[V])$. Then for all environment Γ and store s, we have $(s; \Gamma, E[M]) \gtrsim (s; \Gamma, E'[M])$.*

Proof. For a given Γ we consider $\mathcal{R} = \{(s; \Gamma, E[M]), (s; \Gamma, E'[M]) \mid s, M\}$ and the transitions from both sides:

1. when M is not a value, $(s; M) \longmapsto_{\mathsf{R}} (s'; M')$ and the only transition from both sides is a τ staying knowingly in the relation.
2. (left to right) when $M = V$ by hypothesis $(s; \Gamma, E[V]) \xrightarrow{\tau}{}^{+} s(; \Gamma, E'[V])$ so the first transition from the left-hand side is a τ. We use up to expansion to reach $(s; \Gamma, E'[V])$ which is equal to the right-hand side, and conclude up to reflexivity.
3. (right to left) when $M = V$ and the right-hand side makes some transition $(s; \Gamma, E'[V]) \xrightarrow{\alpha} (s'; \Gamma', N')$ we know in fact that $(s; \Gamma, E[V]) \xrightarrow{\tau}{}^{+} (s; \Gamma, E'[V])$ so $(s; \Gamma, E'[V]) \Longrightarrow\!\!\xrightarrow{\alpha} (s'; \Gamma', N')$ and we conclude again up to reflexivity.

We proved $\mathcal{R}$ is an expansion relation up to expansion and reflexivity.

Remark 3. To get to Theorem 7 we combine (in Lemma 6) proofs for evaluation contexts (Lemma 12) and for non-evaluation contexts (Lemma 11). This separation is critical, as handling all contexts together would yield a much bigger and error-prone bisimulation candidate as L and R in Lemma 11 would be replaced by all intricate combinations of E and E'.

Remark 4. In the proofs leading to Theorem 7 we universally quantify over contexts several times, but we use up to context techniques only a few times. This makes sense, as those are arbitrary contexts with locations containing arbitrary terms that are not necessarily values; we needed tight control over them, and the resulting fine-tuned proof can now be used as a black box.

Remark 5. To see why separating the proof into Lemma 10 Lemma 11 is necessary one must go through several naive steps when expanding the candidate relation relating $(\emptyset; C[(\lambda x.E[x])M])$ to $(\emptyset; C[E[M]])$.

- there can be a location ℓ both in C and M. For instance, C could be $\mathsf{set}_\ell(C_v); C_2$ where C_v is a value context, so the store must be able to contain $s = (\ell \mapsto C_v[(\lambda x.E[x])M])$ on the left, where it contains $s' = (\ell \mapsto C_v[E[M]])$ on the right.
- then C can be $[\cdot]$ so we must be able to compare $(s; (\lambda x.E[x])M)$ to $(s'; E[M])$ which calls on how to relate $(s; M)$ to $(s'; M)$, for instance it implies proving that either both or none reach a value, which we don't know yet, because that is similar to what we already intended to prove (we get into a circular argument).

Theorem 7. *Suppose that E and E' are evaluation contexts and that for all value V value and store s, we have $(s; E[V]) \longmapsto_{\mathsf{R}}^{*} (s; E'[V])$. Then for all environment s and context C, we have $(s; C[E[M]]) \gtrsim (s; C[E'[M]])$.*

Proof. Consequence of Lemma 12 and Lemma 6.

E Example from Section 6

Continuing from Section 6, we show that the relation

$$\mathcal{R} = \left\{ \left(s; M', (\mathsf{incr}_\ell, \mathsf{test}_\ell)_{\ell \in \tilde{\ell}}\right), \left(\emptyset; N', (F, F)_{\ell \in \tilde{\ell}}\right) \;\text{ s.t. } \forall \ell \in \tilde{\ell}, \quad s(\ell) \text{ is even} \right\}$$

is a weak bisimulation up to store, $\mathcal{C}$ and expansion. We write $(s; \Gamma_{\tilde{\ell}})$ for the left-hand side of a pair in $\mathcal{R}$ and $(\emptyset; \Delta_{\tilde{\ell}})$ for the right-hand side.

Consider a transition $1, C, \mathtt{cod}(r)$ from M' and N'. We write below Γ' for $\Gamma_{\tilde{\ell}}, \mathsf{getset}(r)$ and Δ' for $\Delta_{\tilde{\ell}}, \mathsf{getset}(r)$.

- $(s; \Gamma_{\tilde{\ell}}) \xrightarrow{1, C, \mathtt{cod}(r)} (s \uplus r[\Gamma']; \Gamma', \nu\ell\ \ell := 0; C[\Gamma'](\mathsf{incr}_\ell); \mathsf{test}_\ell I)$
- $(\emptyset; \Delta_{\tilde{\ell}}) \xrightarrow{1, C, \mathtt{cod}(r)} (r[\Delta']; \Delta', C[\Delta'](F); FI)$

In the first line, we make the configuration run two τ transitions, so that $\nu\ell$ and $\ell := 0$ get executed. Now we have a new store $s' = s \uplus (\ell \mapsto 0)$ ($s'(\ell)$ is even, so in this respect we stay in the bisimulation candidate).

Now the main term is $C[\Gamma'](\mathsf{incr}_\ell); \mathsf{test}_\ell I)$ which can be rewritten to $D[\Gamma_{\tilde{\ell}.\ell}, \mathsf{getset}(r)]$ for some context D. On the right-hand side $C[\Delta'](F); FI$ can be rewritten to $D[\Delta_{\tilde{\ell}.\ell}, \mathsf{getset}(r)]$ as well. By construction $(s'; \Gamma_{\tilde{\ell}.\ell})\ \mathcal{R}\ (\emptyset; \Delta_{\tilde{\ell}.\ell})$ hence

$$(s' \uplus r[\Gamma']; \Gamma_{\tilde{\ell}.\ell}, \mathsf{getset}(r)) \quad \mathsf{store}(\mathcal{R}) \quad (r[\Delta']; \Delta_{\tilde{\ell}.\ell}, \mathsf{getset}(r)) \,.$$

Now we apply $\mathcal{C}$ with the context D, then the weakening w to remove incr_ℓ and test_ℓ to reach the pair we wanted.

Now that we handled M' and N', let us look at any transition $i, C, \mathtt{cod}(r)$ coming from some incr_ℓ (and F on the other side). It will result in I on both sides (the argument C is discarded), with $s(\ell)$ being updated to $s(\ell)+2$ which stays in the relation. The store is augmented with $r[\Gamma']$ and $r[\Delta']$ and the environment with $\mathsf{getset}(r)$ which can be safely removed by the store technique as we did before. The same is done when a test_ℓ is run: both sides reduce to I, the argument is discarded, and the r part of the transition is garbage-collected.

We now present some details for the second proof of the example. We show that the relation

$$\mathcal{S} = \{(M', N'')\} \cup \{(\ell \mapsto 2n; \mathsf{incr}_\ell, \mathsf{test}_\ell), (\emptyset; \mathsf{incr}_0, \mathsf{test}_0) \;\text{ s.t. } n \in \mathbb{N}\}$$

is a strong bisimulation *up to unfolding, store, weakening, strengthening, transitivity and context.*

This up-to technique, unsound in the weak case (transitivity is unsound), is powerful enough to make the bisimulation considerably smaller. Proving that the second member of $\mathcal{S}$ progresses to itself (up to store) is straightforward. We focus on the following transitions from M' and N'':

$$\begin{aligned}(\emptyset, M') &\xrightarrow{1,C,\mathtt{cod}(r)} (\ r[\Gamma];\ \ \Gamma, \nu\ell\ \ell := 0; C[\Gamma](\mathsf{incr}_\ell); \mathsf{test}_\ell I\ ) \triangleq H_1\\ (\emptyset, N'') &\xrightarrow{1,C,\mathtt{cod}(r)} (\ r[\Delta];\ \ \Delta, I; I; \qquad C[\Delta](\mathsf{incr}_0); \mathsf{test}_0 I\ ) \triangleq H_2\end{aligned}$$

where $\Gamma = M', \mathsf{getset}(r)$ and $\Delta = N'', \mathsf{getset}(r)$. We use $\mathbf{sp}$ as an up-to technique[3] twice to run two steps of reduction on both sides:

$$H_1 \xrightarrow{\tau}\xrightarrow{\tau} H_1' \qquad \text{and} \qquad H_2 \xrightarrow{\tau}\xrightarrow{\tau} H_2' \ .$$

This way we trigger $\nu\ell$ and $\ell := 0$ and obtain two configurations H_1' and H_2' that can be related using a few up-to functions:

$$\begin{array}{rl} & (\ r[\Gamma] \uplus (\ell \mapsto 0);\ \ \Gamma, C[\Gamma](\mathsf{incr}_\ell); \mathsf{test}_\ell I\ ) = H_1'\\ \mathsf{w}(\mathcal{C}(\mathsf{store}(\mathsf{str}(\mathcal{S})))) & (\ r[\Gamma];\ \ \Gamma, C[\Gamma](\mathsf{incr}_0); \mathsf{test}_0 I\ )\\ \mathcal{C}(\mathsf{store}(\mathcal{S})) & (\ r[\Delta];\ \ \Delta, C[\Delta](\mathsf{incr}_0); \mathsf{test}_0 I\ ) = H_2' \ .\end{array}$$

We detail below how we go from the first to the second line. We write $\Gamma_\ell \triangleq \mathsf{incr}_\ell, \mathsf{test}_\ell$ and $\Gamma_0 \triangleq \mathsf{incr}_0, \mathsf{test}_0$.

$$\begin{array}{lcl}
(\ell \mapsto 0; \Gamma_\ell) & \mathcal{S} & (\emptyset; \Gamma_0)\\
(\ell \mapsto 0; \Gamma_\ell, M') & \mathsf{str}(-) & (\emptyset; \Gamma_0, M')\\
(r[\Gamma] \uplus \ell \mapsto 0; \Gamma_\ell, \Gamma) & \mathsf{store}(-) & (r[\Gamma]; \Gamma_0, \Gamma)\\
(r[\Gamma] \uplus \ell \mapsto 0; \Gamma_\ell, \Gamma, C[\Gamma](\mathsf{incr}_\ell); \mathsf{test}_\ell I)) & \mathcal{C}(-) & (r[\Gamma]; \Gamma_0, \Gamma, C[\Gamma](\mathsf{incr}_0); \mathsf{test}_0 I)\\
(r[\Gamma] \uplus \ell \mapsto 0; \quad \Gamma, C[\Gamma](\mathsf{incr}_\ell); \mathsf{test}_\ell I)) & \mathsf{w}(-) & (r[\Gamma]; \quad \Gamma, C[\Gamma](\mathsf{incr}_0); \mathsf{test}_0 I)
\end{array}$$

Going from the second to the third line is easier:

$$\begin{array}{lcl}
(\emptyset; M') & \mathcal{S} & (\emptyset; N'')\\
(r[\Gamma]; \Gamma) & \mathsf{store}(\mathcal{S}) & (r[\Delta]; \Delta)\\
(r[\Gamma]; \Gamma, C[\Gamma](\mathsf{incr}_0); \mathsf{test}_0 I) & \mathcal{C}(\mathsf{store}(\mathcal{S})) & (r[\Delta]; \Delta, C[\Delta](\mathsf{incr}_0); \mathsf{test}_0 I)
\end{array}$$

Finally we proved that $H_1\ f(\mathcal{S})\ H_2$ where $f = \mathbf{sp} \circ \mathbf{sp} \circ \mathsf{star} \circ (\mathsf{str} \cup \mathsf{store} \cup \mathcal{C} \cup \mathsf{w})^\omega$ is a compatible function, and hence $\mathcal{S} \rightsquigarrow_{\mathbf{sp}} f(\mathcal{S}) \cup \mathsf{store}(\mathcal{S})$ (not forgetting the second member of $\mathcal{S}$).

To conclude, $\mathcal{S}$, as a strong bisimulation up to (unfolding, store, weakening, strengthening, transitivity and context), is included in $\sim$.

[3] If $\xrightarrow{\tau}$ is deterministic then $(\xrightarrow{\tau} \mathcal{R} \xleftarrow{\tau}) \subseteq \mathbf{sp}(\mathcal{R})$.

Coinduction up to in a fibrational setting

Filippo Bonchi, Daniela Petrisan, Damien Pous, Jurriaan Rot

Coinduction Up-To in a Fibrational Setting *

Filippo Bonchi Daniela Petrişan
Damien Pous †
LIP, CNRS, INRIA, ENS Lyon,
Université de Lyon, UMR 5668
{filippo.bonchi,daniela.petrisan,damien.pous}
@ens-lyon.fr

Jurriaan Rot ‡
LIACS - Leiden University, CWI
j.c.rot@liacs.leidenuniv.nl

Abstract

Bisimulation up-to enhances the coinductive proof method for bisimilarity, providing efficient proof techniques for checking properties of different kinds of systems. We prove the soundness of such techniques in a fibrational setting, building on the seminal work of Hermida and Jacobs. This allows us to systematically obtain up-to techniques not only for bisimilarity but for a large class of coinductive predicates modelled as coalgebras. By tuning the parameters of our framework, we obtain novel techniques for unary predicates and nominal automata, a variant of the GSOS rule format for similarity, and a new categorical treatment of weak bisimilarity.

Categories and Subject Descriptors F.3 [*Logics and meanings of programs*]; F.4 [*Mathematical logic and formal languages*]

General Terms Theory.

Keywords fibrations, coinductive predicates, bisimulation up-to, GSOS, up-to techniques, similarity, bialgebras, nominal automata.

1. Introduction

1.1 Coinduction up-to

The rationale behind coinductive up-to techniques is the following. Suppose you have a characterisation of an object of interest as a greatest fixed-point. For instance, behavioural equivalence in CCS is the greatest fixed-point of a monotone function B on relations, describing the standard bisimulation game. This means that to prove two processes equivalent, it suffices to exhibit a relation R that relates them, and which is a B-*invariant*, i.e., $R \subseteq B(R)$. Such a task can however be painful or inefficient, and one could prefer to exhibit a relation which is only a B-invariant *up to some function* A, i.e., $R \subseteq B(A(R))$.

Not every function A can safely be used: A should be *sound* for B, meaning that any B-invariant up to A should be contained in a B-invariant. Instances of sound functions for behavioural equivalence in process calculi usually include transitive closure, context closure and congruence closure. The use of such techniques dates back to Milner's work on CCS [21]; a famous example of an unsound technique is that of weak bisimulation up to weak bisimilarity. Since then, coinduction up-to proved useful, if not essential, in numerous proofs about concurrent systems (see [25] for a list of references); it has been used to obtain decidability results [8], and more recently to improve standard automata algorithms [7].

The theory underlying these techniques was first developed by Sangiorgi [27]. It was then reworked and generalised by one of the authors to the abstract setting of complete lattices [24, 25]. The key observation there is that the notion of soundness is not compositional: the composition of two sound functions is not necessarily sound itself. The main solution to this problem consists in restricting to *compatible* functions, a subset of the sound functions which enjoys nice compositionality properties and contains most of the useful techniques.

An illustrative example of the benefits of a modular theory is the following: given a signature Σ, consider the *congruence closure* function, that is, the function Cgr mapping a relation R to the smallest congruence containing R. This function has proved to be useful as an up-to technique for language equivalence of non-deterministic automata [7]. It can be decomposed into small pieces as follows: $Cgr = Trn \circ Sym \circ Ctx \circ Rfl$, where Trn is the transitive closure, Sym is the symmetric closure, Rfl is the reflexive closure, and Ctx is the context closure associated to Σ. Since compatibility is preserved by composition (among other operations), the compatibility of Cgr follows from that of its smaller components. In turn, transitive closure can be decomposed in terms of relational composition, and context closure can be decomposed in terms of the smaller functions that close a relation with respect to Σ one symbol at a time. Compatibility of such functions can thus be obtained in a modular way.

A key observation in the present work is that when we move to a coalgebraic presentation of the theory, compatible functions generalise to functors equipped with a distributive law (Section 3).

1.2 Fibrations and coinductive predicates

Coalgebras are a tool of choice for describing state based systems: given a functor F determining its type (e.g., labelled transition systems, automata, streams), a system is just an F-coalgebra (X, ξ). When F has a final coalgebra (Ω, ω), this gives a canonical notion of behavioural equivalence [17]:

$$\begin{array}{ccc} X & \xrightarrow{[\![\cdot]\!]} & \Omega \\ {\scriptstyle\xi}\downarrow & & \downarrow{\scriptstyle\omega} \\ FX & \xrightarrow{F[\![\cdot]\!]} & F\Omega \end{array}$$

two states $x, y \in X$ are equivalent if they are mapped to the same element in the final coalgebra.

When the functor F preserves weak pullbacks—which we shall assume throughout this introductory section for the sake of

---

* Extended version of the paper with the same title, to appear in Proc. *CSL-LICS 2014*, July 14–18, 2014, Vienna, Austria, and to be available at http://dx.doi.org/10.1145/2603088.2603149

† The first three authors acknowledge support from the ANR projects 2010-BLAN-0305 PiCoq and 12IS02001 PACE.

‡ The research of this author has been funded by the Netherlands Organisation for Scientific Research (NWO), CoRE project, dossier number: 612.063.920.

simplicity—behavioural equivalence can be characterised coinductively using Hermida-Jacobs bisimulations [14, 30]: given an F-coalgebra (X, ξ), behavioural equivalence is the largest B-invariant for a monotone function B on Rel_X, the poset of binary relations over X. This function B can be decomposed as

$$B \triangleq \xi^* \circ \mathsf{Rel}(F)_X : \mathsf{Rel}_X \to \mathsf{Rel}_X$$

Let us explain the notations used here. We consider the category Rel whose objects are relations $R \subseteq X^2$ and morphisms from $R \subseteq X^2$ to $S \subseteq Y^2$ are maps from X to Y sending pairs in R to pairs in S. For each set X the poset Rel_X of binary relations over X is a subcategory of Rel, also called the fibre over X. The functor F has a canonical lifting to Rel, denoted by $\mathsf{Rel}(F)$. This lifting restricts to a functor $\mathsf{Rel}(F)_X : \mathsf{Rel}_X \to \mathsf{Rel}_{FX}$, which in this case is just a monotone function between posets. The monotone function $\xi^* : \mathsf{Rel}_{FX} \to \mathsf{Rel}_X$ is the *inverse image* of the coalgebra ξ mapping a relation $R \subseteq (FX)^2$ to $(\xi \times \xi)^{-1}(R)$.

To express other predicates than behavioural equivalence, one can take arbitrary *liftings* of F to Rel, different from the canonical one. Any lifting $\overline{F}$ yields a functor B defined as

$$B \triangleq \xi^* \circ \overline{F}_X : \mathsf{Rel}_X \to \mathsf{Rel}_X \qquad (\dagger)$$

The final coalgebra, or greatest fixed-point for such a B is called a *coinductive predicate* [13, 14]. By taking appropriate $\overline{F}$, one can obtain, for instance, various behavioural preorders: similarity on labelled transition systems (LTSs), language inclusion on automata, or lexicographic ordering of streams.

This situation can be further generalised using *fibrations*. We refer the reader to the first chapter of [16] for a gentle introduction, or to Section 2 for succinct definitions. The functor $p : \mathsf{Rel} \to \mathsf{Set}$ mapping a relation $R \subseteq X^2$ to its support set X is a fibration, where the inverse image ξ^* is just the *reindexing functor* of ξ. By choosing a different fibration than Rel, one can obtain coinductive characterisations of objects that are not necessarily binary relations, e.g., unary predicates like divergence, ternary relations, or metrics.

Our categorical generalisation of compatible functions provides a natural extension of this fibrational framework with a systematic treatment of up-to techniques: we provide functors (i.e., monotone functions in the special case of the Rel fibration) that are compatible with those functors B corresponding to coinductive predicates.

For instance, when the chosen lifting $\overline{F}$ is a *fibration map*, the functor corresponding to a technique called "up to behavioural equivalence" is compatible (Theorem 1). The canonical lifting of a functor is always such a fibration map, so that when F is the functor for LTSs, we recover the soundness of the very first up-to technique from the literature, namely "bisimulation up to bisimilarity" [21]. One can also check that another lifting of this same functor but in another fibration yields the divergence predicate, and is a fibration map. We thus obtain the validity of the "divergence up to bisimilarity" technique.

1.3 Bialgebras and up to context

Another important class of techniques comes into play when considering systems with an algebraic structure on the state space (e.g., the syntax of a process calculus). A minimal requirement for such systems usually is that behavioural equivalence should be a congruence. In the special case of bisimilarity on LTSs, several rule formats have been proposed to ensure such a congruence property [1]. At the categorical level, the main concept to study such systems is that of *bialgebras*. Assume two endofunctors T, F related by a distributive law $\lambda : TF \Rightarrow FT$. A λ-bialgebra consists in a triple (X, α, ξ) where (X, α) is a T-algebra, (X, ξ) is an F-coalgebra, and a diagram involving λ commutes. It is well known that in such a bialgebra, behavioural equivalence is a congruence with respect to T [31]. This is actually a generalisation of the fact that bisimilarity is a congruence for all GSOS specifications [3]: GSOS specifications are in one-to-one correspondence with distributive laws between the appropriate functors [2, 31].

This congruence result can be strengthened into a compatibility result [26]: in any λ-bialgebra, the contextual closure function that corresponds to T is compatible for behavioural equivalence. By moving to fibrations, we generalise this result so that we can obtain up to context techniques for arbitrary coinductive predicates: unary predicates like divergence, by using another fibration than Rel; but also other relations than behavioural equivalence, like the behavioural preorders mentioned above, or weak bisimilarity.

The technical device we need to establish this result is that of *bifibrations*, fibrations p whose opposite functor p^{op} is also a fibration. We keep the running example of the Rel fibration for the sake of clarity; the results are presented in full generality in the remaining parts of the paper. In such a setting, any morphism $f : X \to Y$ in Set has a *direct image* $\coprod_f : \mathsf{Rel}_X \to \mathsf{Rel}_Y$. Now given an algebra $\alpha : TX \to X$ for a functor T on Set, any lifting $\overline{T}$ of T gives rise to a functor on the fibre above X, defined dually to $(\dagger)$:

$$C \triangleq \coprod_\alpha \circ \overline{T}_X : \mathsf{Rel}_X \to \mathsf{Rel}_X \qquad (\ddagger)$$

When we take for $\overline{T}$ the canonical lifting of T in Rel, then C is the contextual closure function corresponding to the functor T. We shall see that we sometimes need to consider variations of the canonical lifting to obtain a compatible up-to technique (e.g., up to "monotone" contexts for checking language inclusion of weighted automata—Section 5.1).

Now, starting from a λ-bialgebra (X, α, ξ), and given two liftings $\overline{T}$ and $\overline{F}$ of T and F, respectively, the question is whether the above functor C is compatible with the functor B defined earlier in $(\dagger)$. The simple condition we give in this paper is the following: the distributive law $\lambda : TF \Rightarrow FT$ should lift to a distributive law $\overline{\lambda} : \overline{T}\,\overline{F} \Rightarrow \overline{F}\,\overline{T}$ (Theorem 2).

This condition is always satisfied in the bifibration Rel, when $\overline{T}$ and $\overline{F}$ are the canonical liftings of T and F. Thus we obtain as a corollary the compatibility of bisimulation of up to context in λ-bialgebras, which is the main result from [26]—soundness was previously observed by Lenisa et al. [19, 20] and then Bartels [2].

The present work allows us to go further in several directions, as illustrated below.

1.4 Contributions and Applications

The main contribution of this paper is the abstract framework developed in Section 4; it allows us to derive the soundness of a wide range of both novel and well-established up-to techniques for arbitrary coinductive predicates. Sections 5 and 6 are devoted to several such applications, which we describe now.

When working in the predicate fibration on Set, one can characterise some formulas from modal logic as coinductive predicates (see [9] for an account of coalgebraic modal logic). Our framework allows us to introduce up-to techniques in this setting: we consider the formula $\nu x.\langle\tau\rangle x$ in Section 5.2, and we provide a technique called "divergence up to left contexts and behavioural equivalence". We use it to prove divergence of a simple process using a finite invariant, while the standard method requires an infinite one.

One can also change the base category: by considering the fibration of equivariant relations over nominal sets, we show how to obtain up-to techniques for language equivalence of non-deterministic nominal automata [4]. In Section 5.3, these techniques allow us to prove the equivalence of two nominal automata using an orbit-finite relation, where the standard method would require an infinite one (recall that the determinisation of a nominal automaton is not necessarily orbit-finite).

Another benefit of the presented theory is modularity w.r.t. the liftings chosen to define coinductive predicates: two liftings can be composed, and we give sufficient conditions for deriving compatible functors for the composite lifting out of compatible functors for its sub-components (Section 6). We give two examples of such a situation: similarity, and weak bisimilarity on LTSs.

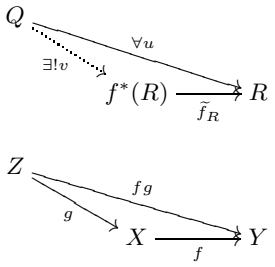
By using Hughes and Jacobs' definition of similarity [15], we obtain that for "up to context" to be compatible it suffices to start from a *monotone* distributive law (Section 6.1). In the special case of LTSs, this monotonicity condition amounts to the *positive GSOS* rule format [12]: GSOS [3] without negative premises.

In Section 6.2 we propose a novel characterisation of weak bisimilarity on LTSs, that fits into our framework. This allows us to give a generic condition for "up to context" to be compatible (and hence weak bisimilarity to be a congruence). In particular, this condition rules out the sum operation from CCS, which is well known not to preserve weak bisimilarity.

2. Preliminaries

We refer the reader to [16] for background on fibrations and recall here basic definitions.

Definition 1. *A functor $p\colon \mathcal{E} \to \mathcal{B}$ is called a* fibration *when for every morphism $f\colon X \to Y$ in $\mathcal{B}$ and every R in $\mathcal{E}$ with $p(R) = Y$ there exists a map $\widetilde{f_R}\colon f^*(R) \to R$ such that $p(\widetilde{f_R}) = f$ satisfying the universal property: For all maps $g : Z \to X$ in $\mathcal{B}$ and $u\colon Q \to R$ in $\mathcal{E}$ sitting above fg (i.e., $p(u) = fg$) there is a unique map $v\colon Q \to f^*(R)$ such that $u = \widetilde{f_R}v$ and $p(v) = g$.*



For X in $\mathcal{B}$ we denote by $\mathcal{E}_X$ the *fibre* above X, i.e., the subcategory of $\mathcal{E}$ with objects mapped by p to X and arrows sitting above the identity on X.

A map $\widetilde{f}$ as above is called a *Cartesian lifting* of f and is unique up to isomorphism. If we make a choice of Cartesian liftings, the association $R \mapsto f^*(R)$ gives rise to the so-called *reindexing functor* $f^*\colon \mathcal{E}_Y \to \mathcal{E}_X$.

The fibrations considered in this paper are *bicartesian* (both $\mathcal{E}$ and $\mathcal{B}$ have a bicartesian structure strictly preserved by p) and *split*, i.e., the reindexing functors behave well with respect to composition and identities: $(1_X)^* = 1_{\mathcal{E}_X}$ and $(f \circ g)^* = g^* \circ f^*$.

A functor $p\colon \mathcal{E} \to \mathcal{B}$ is called a *bifibration* if both $p\colon \mathcal{E} \to \mathcal{B}$ and $p^{op}\colon \mathcal{E}^{op} \to \mathcal{B}^{op}$ are fibrations. A fibration $p\colon \mathcal{E} \to \mathcal{B}$ is a bifibration if and only if each reindexing functor $f^*\colon \mathcal{E}_Y \to \mathcal{E}_X$ has a left adjoint $\coprod_f \dashv f^*$, see [16, Lemma 9.1.2].

Example 1. *Let* Pred *be the category of predicates: objects are pairs of sets (P, X) with $P \subseteq X$ and morphisms $f\colon (P, X) \to (Q, Y)$ are arrows $f\colon X \to Y$ that can be restricted to $f\big|_P\colon P \to Q$.*

Similarly, we can consider the category Rel *whose objects are pairs of sets (R, X) with $R \subseteq X^2$ and morphisms $f\colon (R, X) \to (S, Y)$ are arrows $f\colon X \to Y$ such that $f \times f$ can be restricted to $f \times f\big|_R\colon R \to S$.*

The functors mapping predicates, respectively, relations to their underlying sets are bifibrations. The fibres Pred_X and Rel_X sitting above X are the posets of subsets of X, respectively relations on X, ordered by inclusion. The reindexing functors are given by inverse image and their left adjoints by direct image.

Given fibrations $p\colon \mathcal{E} \to \mathcal{B}$ and $p'\colon \mathcal{E}' \to \mathcal{B}$ and $F\colon \mathcal{B} \to \mathcal{B}$, we call $\overline{F}\colon \mathcal{E} \to \mathcal{E}'$ a *lifting* of F when $p'\overline{F} = Fp$. Notice that a lifting $\overline{F}$ restricts to a functor between the fibres $\overline{F}_X\colon \mathcal{E}_X \to \mathcal{E}_{FX}$. When the subscript X is clear from the context we will omit it.

A *fibration map* between $p\colon \mathcal{E} \to \mathcal{B}$ and $p'\colon \mathcal{E}' \to \mathcal{B}$ is a pair $(\overline{F}, F)$ such that $\overline{F}$ is a lifting of F that preserves the Cartesian liftings: $(Ff)^*\overline{F} = \overline{F}f^*$ for any $\mathcal{B}$-morphism f. We denote by $\mathsf{Fib}(\mathcal{B})$ the category of fibrations with base $\mathcal{B}$.

Example 2. *A* Set*-endofunctor T has a* canonical *relation lifting* $\mathsf{Rel}(T)\colon \mathsf{Rel} \to \mathsf{Rel}$. *Represent $R \in \mathsf{Rel}_X$ as a jointly mono span $X \leftarrow R \rightarrow X$ and apply T. Then $\mathsf{Rel}(T)(R)$ is obtained by factorising the induced map $TR \to TX \times TX$. When T preserves weak pullbacks, $(\mathsf{Rel}(T), T)$ is a fibration map (see e.g. [15]).*

3. Compatible Functors

Given two monotone functions $A, B\colon \mathcal{C} \to \mathcal{C}$ on a complete lattice $\mathcal{C}$, A is said to be B-compatible if $AB \subseteq BA$. In [25, Theorem 6.3.9], it is shown that any B-compatible function A is sound, that is, it can be used as an up-to technique: every B-invariant up to A is included in a B-invariant.

This result is an instance of a more general fact which holds in any category $\mathcal{C}$ with countable coproducts and for any pair of endofunctors A, B equipped with a distributive law $\gamma\colon AB \Rightarrow BA$. Indeed, following the proof of [2, Theorem 3.8], for any BA-coalgebra ξ (that is a B-invariant up to A) one can find a B-coalgebra ζ (that is a B-invariant) making the next diagram commutative.

$$\begin{array}{ccc} X & \xrightarrow{\kappa_0} & A^\omega X \\ {\scriptstyle\xi}\downarrow & & \downarrow{\scriptstyle\zeta} \\ BAX & \xrightarrow[B\kappa_1]{} & BA^\omega X \end{array}$$

(Here A^ω denotes the coproduct $\coprod_{i<\omega} A^i$ of all finite iterations of A and κ_0, κ_1 are the injections of X and AX respectively, into $A^\omega X$. Alternatively, we can replace the countable coproduct A^ω by the free monad on A, assuming the latter exists. In this case, the result is an instance of the generalized powerset construction [28].)

Similarly, that compatible functions preserve bisimilarity [25, Lemma 6.4.3] is an instance of the well-known fact [31] that a final B-coalgebra νB lifts to a final γ-bialgebra for $\gamma\colon AB \Rightarrow BA$. When $\mathcal{C}$ is a lattice, this entails that $A(\nu B) \subseteq \nu B$. For instance, if B is a predicate for bisimilarity and A is the congruence closure function, we obtain that bisimilarity is a congruence whenever the congruence closure function is compatible.

As discussed in the Introduction, the main interest in compatible functions comes from their nice compositionality properties. This leads us to define compatibility of arbitrary functors of type $\mathcal{C} \to \mathcal{C}'$ rather than just endofunctors.

Definition 2. *Consider two endofunctors $B\colon \mathcal{C} \to \mathcal{C}$ and $B'\colon \mathcal{C}' \to \mathcal{C}'$. We say that a functor $A\colon \mathcal{C} \to \mathcal{C}'$ is* (B, B')-compatible *when there exists a natural transformation $\gamma\colon AB \Rightarrow B'A$.*

Notice that the pair (A, γ) is a morphism between endofunctors B and B' in the sense of [20]. Since the examples dealt with in this paper involve only poset fibrations, we will omit the natural transformation γ from the notation. Moreover, given an endofunctor $B : \mathcal{C} \to \mathcal{C}$, we will simply write that $A : \mathcal{C}^n \to \mathcal{C}^m$ is B-compatible, when A is (B^n, B^m)-compatible.

This definition makes it possible to use the internal notions of product and pairing to emphasise the compositionality aspect. For

instance, coproduct becomes a compatible functor by itself, rather than a way to compose compatible functors.

Proposition 1. *Compatible functors are closed under the following constructions:*

(i) composition: if A is (B,C)-compatible and A' is (C,D)-compatible, then $A' \circ A$ is (B,D)-compatible;
(ii) pairing: if $(A_i)_{i\in\iota}$ are (B,C)-compatible, then $\langle A_i\rangle_{i\in\iota}$ is (B,C^ι)-compatible;
(iii) product: if A is (B,C)-compatible and A' is (B',C')-compatible, then $A\times A'$ is $(B\times B', C\times C')$-compatible;

Moreover, for an endofunctor $B\colon \mathcal{C}\to\mathcal{C}$,

(iv) the identity functor $Id\colon \mathcal{C}\to\mathcal{C}$ is B-compatible;
(v) the constant functor to the carrier of any B-coalgebra is B-compatible, in particular the final one if it exists;
(vi) the coproduct functor $\coprod\colon \mathcal{C}^\iota\to\mathcal{C}$ is (B^ι, B)-compatible.

4. Up-to Techniques in a Fibration

Throughout this section we fix a bifibration $p\colon \mathcal{E}\to\mathcal{B}$, an endofunctor $F\colon \mathcal{B}\to\mathcal{B}$, a lifting $\overline{F}\colon \mathcal{E}\to\mathcal{E}$ of F and a coalgebra $\xi\colon X\to FX$. Intuitively, the studied system lives in the base category $\mathcal{B}$ while its properties live in $\mathcal{E}_X$, the fibre above X. We thus instantiate the category $\mathcal{C}$ from the previous section with $\mathcal{E}_X$.

As explained in the Introduction (†), we discuss proof techniques for the properties modelled as final coalgebras of the functor $\xi^*\circ\overline{F}_X\colon \mathcal{E}_X\to\mathcal{E}_X$, that we refer hereafter as $\overline{F}_\xi$. In Rel, when $\overline{F}$ is the canonical lifting, $\mathsf{Rel}(F)_\xi$-coalgebras are exactly the Hermida-Jacobs bisimulations [14].

To obtain sound techniques for $\overline{F}_\xi$, it suffices to find $\overline{F}_\xi$-compatible endofunctors on $\mathcal{E}_X$. We provide such functors by giving conditions on the lifting $\overline{F}$, abstracting away from the coalgebra ξ at hand.

4.1 Compatibility of Behavioural Equivalence Closure

The most basic technique is up to behavioural equivalence, a prime example of which is Milner's up to bisimilarity [21], where a relation R is mapped into $\sim R\sim$. If f is the unique morphism from ξ to a final F-coalgebra (assumed to exist), behavioural equivalence is the kernel of f. This leads us to consider the functor

$$Bhv = f^*\circ\coprod_f : \mathcal{E}_X\to\mathcal{E}_X.$$

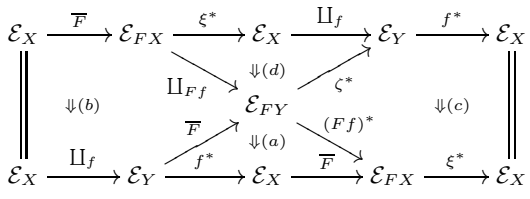
For the fibrations $\mathsf{Pred}\to\mathsf{Set}$ and $\mathsf{Rel}\to\mathsf{Set}$ the functor Bhv maps a predicate, respectively a relation, to its closure under behavioural equivalence. The compatibility of Bhv is an instance of:

Theorem 1. *Suppose that $(\overline{F},F)$ is a fibration map. For any F-coalgebra morphism $f\colon (X,\xi)\to(Y,\zeta)$, the functor $f^*\circ\coprod_f$ is $\overline{F}_\xi$-compatible.*

Proof sketch. We exhibit a natural transformation

$$f^*\circ\coprod_f\circ(\xi^*\circ\overline{F})\Rightarrow(\xi^*\circ\overline{F})\circ f^*\circ\coprod_f$$

obtained by pasting the 2-cells $(a),(b),(c),(d)$ in the following diagram:



(a) Since $(\overline{F},F)$ is a fibration map we have that
$$\overline{F}f^* = (Ff)^*\overline{F}$$
(b) is a consequence of Lemma 3 in Appendix B.
(c) is a natural isomorphism and comes from the fact that f is a coalgebra map and the fibration is split.
(d) is obtained from (c) using the counit of $\coprod_f \dashv f^*$ and the unit of $\coprod_{Ff}\dashv (Ff)^*$.

(Note that this proof decomposes into a proof that $\coprod_f$ is $(\overline{F}_\xi,\overline{F}_\zeta)$-compatible, by pasting (b) and (d), and a proof that f^* is $(\overline{F}_\zeta,\overline{F}_\xi)$-compatible, by pasting (a) and (c). These two independent results can be composed by Proposition 1(i) to obtain the theorem.) □

Corollary 1. *If F is a Set-functor preserving weak pullbacks then the behavioural equivalence closure functor Bhv is $\mathsf{Rel}(F)_\xi$-compatible.*

Proof. $(\mathsf{Rel}(F),F)$ is a fibration map whenever F preserves weak pullbacks (see e.g. [15]). □

From Theorem 1 we also derive the soundness of up-to Bhv for unary predicates: the *monotone predicate liftings* used in coalgebraic modal logic [9] are fibration maps [17], thus the hypothesis of Theorem 1 are satisfied.

4.2 Compatibility of Equivalence Closure

In this section we show that compatibility of equivalence closure can be modularly derived from compatibility of reflexive, symmetric and transitive closures. For the latter it suffices to prove that relational composition is compatible. Composition of relations can be expressed in a fibrational setting, by considering the category $\mathsf{Rel}\times_{\mathsf{Set}}\mathsf{Rel}$ obtained as a pullback of the fibration $\mathsf{Rel}\to\mathsf{Set}$ along itself:

$$\begin{array}{ccc} \mathsf{Rel}\times_{\mathsf{Set}}\mathsf{Rel} & \longrightarrow & \mathsf{Rel} \\ \downarrow & \lrcorner & \downarrow \\ \mathsf{Rel} & \longrightarrow & \mathsf{Set} \end{array}$$

Then relational composition is a functor $\otimes\colon \mathsf{Rel}\times_{\mathsf{Set}}\mathsf{Rel}\to\mathsf{Rel}$ mapping $R,S\subseteq X\times X$ to their composition. As we will see as a corollary of Proposition 2, when proving compatibility of relational composition with respect to $\overline{F}_\xi$ we can abstract away from the coalgebra ξ and simply use that $\otimes$ is a morphism of endofunctors from $\overline{F}^2$ to $\overline{F}$.

Compatibility of symmetric and reflexive closures can be proved following the same principle. This leads us to consider for an arbitrary fibration $\mathcal{E}\to\mathcal{B}$ its n-fold product in the category $\mathsf{Fib}(\mathcal{B})$, denoted by $\mathcal{E}^{\times_{\mathcal{B}}n}\to\mathcal{B}$. The objects in $\mathcal{E}^{\times_{\mathcal{B}}n}$ are tuples of objects in $\mathcal{E}$ belonging to the same fibre. This product is computed fibrewise, that is, $\mathcal{E}^{\times_{\mathcal{B}}n}_X = \mathcal{E}^n_X$. For $n{=}0$ we have $\mathcal{E}^0=\mathcal{B}$.

Hereafter, we are interested in functors $G\colon \mathcal{E}^{\times_{\mathcal{B}}n}\to\mathcal{E}$ that are liftings of the identity functor on $\mathcal{B}$: for each X in $\mathcal{B}$ we have functors $G_X\colon \mathcal{E}^n_X\to\mathcal{E}_X$. Then relational composition is just an instance of G for $n{=}2$.

Proposition 2. *Let $G\colon \mathcal{E}^{\times_{\mathcal{B}}n}\to\mathcal{E}$ be a lifting of the identity, with a natural transformation $G\overline{F}^n\Rightarrow\overline{F}G$. Then G_X is $\overline{F}_\xi$-compatible.*

We list now several applications of the proposition for the fibration $\mathsf{Rel}\to\mathsf{Set}$.

($n{=}0$) Let $Rfl\colon \mathsf{Set}\to\mathsf{Rel}$ be the functor mapping each set X to Δ_X, the identity relation on X. Rfl_X is $\overline{F}_\xi$-compatible if
$$\Delta_{FX}\subseteq\overline{F}\Delta_X. \qquad (*)$$

(n=1) Let $Sym\colon \mathsf{Rel} \to \mathsf{Rel}$ be the functor mapping each relation $R \subseteq X^2$ to its converse $R^{-1} \subseteq X^2$. Sym_X is $\overline{F}_\xi$-compatible if for all relations $R \subseteq X^2$

$$\overline{F}(R)^{-1} \subseteq \overline{F}(R^{-1}). \qquad (**)$$

(n=2) Let $\otimes\colon \mathsf{Rel} \times_{\mathsf{Set}} \mathsf{Rel} \to \mathsf{Rel}$ be the relational composition functor. Then $\otimes_X$ is $\overline{F}_\xi$-compatible if for all $R, S \subseteq X^2$

$$\overline{F}R \otimes \overline{F}S \subseteq \overline{F}(R \otimes S) \qquad (***)$$

If moreover $T_1, T_2\colon \mathsf{Rel}_X \to \mathsf{Rel}_X$ are two $\overline{F}_\xi$-compatible functors, their pointwise composition $T_1 \otimes T_2 = \otimes_X \circ \langle T_1, T_2\rangle$ is $\overline{F}_\xi$-compatible by Proposition 1 (i,ii).

The transitive closure functor Trn is obtained from $\otimes$ in a modular way:

$$Trn = \coprod_{i\geq 0} (-)^i\colon \mathcal{E}_X \to \mathcal{E}_X$$

where $(-)^0 = \mathrm{Id}$ and $(-)^{i+1} = \mathrm{Id} \otimes (-)^i$. Using Proposition 1 we get

Corollary 2. *If F is a* Set*-functor then the reflexive and symmetric closure functors Rfl_X and Sym_X are $\mathsf{Rel}(F)_\xi$-compatible. Moreover, if F preserves weak pullbacks, then the transitive closure functor Trn_X is $\mathsf{Rel}(F)_\xi$-compatible.*

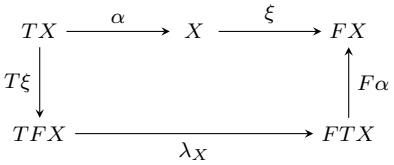
Proof. The above conditions $(*)$ and $(**)$ always hold for the canonical lifting $\overline{F} = \mathsf{Rel}(F)$; $(***)$ holds for $\mathsf{Rel}(F)$ when F preserves weak pullbacks. □

By compositionality (Proposition 1), one can then deduce compatibility of the equivalence closure functor: this functor can be defined as $Eqv \triangleq Trn \circ (\mathrm{Id} + Sym + Rfl)$, where $+$ denotes binary coproduct.

When $\overline{F}_\xi$ has a final coalgebra S, one can define a "self closure" $\mathcal{E}_X$-endofunctor $Slf = \widetilde{S} \otimes \mathrm{Id} \otimes \widetilde{S}$, where $\widetilde{S} : \mathcal{E}_X \to \mathcal{E}_X$ is the constant to S functor. Thanks to Proposition 1, the functor Slf is $\overline{F}_\xi$-compatible whenever $(***)$ holds. When F preserves weak pullbacks and $\overline{F}$ is instantiated to the canonical lifting $\mathsf{Rel}(F)$, Slf coincides with Bhv since S is just behavioural equivalence in this case. If instead we consider the lifting that yields weak bisimilarity (to be defined in Section 6.2), Slf corresponds to a technique called "weak bisimulation up to weak bisimilarity", while Bhv corresponds to "weak bisimulation up to (strong) bisimilarity".

4.3 Compatibility of Contextual Closure

For defining contextual closure, we assume that the state space of the coalgebra is equipped with an algebraic structure. More precisely, we fix a bialgebra for a distributive law $\lambda\colon TF \Rightarrow FT$, that is, a triple (X, α, ξ), where $\alpha : TX \to X$ is a an algebra and $\xi\colon X \to FX$ is a coalgebra such that the next diagram commutes:



Theorem 2. *Let $\overline{T}, \overline{F}\colon \mathcal{E} \to \mathcal{E}$ be liftings of T and F. If $\overline{\lambda}\colon \overline{T}\,\overline{F} \Rightarrow \overline{F}\,\overline{T}$ is a natural transformation sitting above λ, then $\coprod_\alpha \circ \overline{T}$ is $\overline{F}_\xi$-compatible.*

Proof sketch. We exhibit a natural transformation

$$(\coprod_\alpha \circ \overline{T}) \circ (\xi^* \circ \overline{F}) \Rightarrow (\xi^* \circ \overline{F}) \circ (\coprod_\alpha \circ \overline{T}) \ .$$

This is achieved in Figure 1 by pasting five natural transformations, obtained as follows:

(a) is the counit of the adjunction $\coprod_{\lambda_X} \dashv \lambda_X^*$.
(b) comes from $\overline{\lambda}$ being a lifting of λ.
(c) comes from the bialgebra condition, the fibration being split, and the units and counits of the adjunctions $\coprod_\alpha \dashv \alpha^*$, $\coprod_{F\alpha} \dashv (F\alpha)^*$, and $\coprod_{\lambda_X} \dashv \lambda_X^*$.
(d) arises since $\overline{T}$ is a lifting of T, using the universal property of the Cartesian lifting $(T\xi)^*$.
(e) comes from $\overline{F}$ being a lifting of F, combined with the unit and counit of the adjunction $\coprod_\alpha \dashv \alpha^*$.

(Note that like for Theorem 1, this proof actually decomposes into a proof that $\overline{T}$ is $(\overline{F}_\xi, (T\xi)^* \circ \lambda_X^*)$-compatible, and a proof that $\coprod_\alpha$ is $((T\xi)^* \circ \lambda_X^*, \overline{F}_\xi)$-compatible.) □

When the fibration at issue is $\mathsf{Rel} \to \mathsf{Set}$ and $\overline{T}$ is the canonical lifting $\mathsf{Rel}(T)$, one can easily check that $\coprod_\alpha \circ \mathsf{Rel}(T)$ applied to a relation R gives exactly its *contextual closure* as described in [26]. For this reason, we abbreviate $\coprod_\alpha \circ \mathsf{Rel}(T)$ to Ctx. When moreover $\overline{F}$ is the canonical lifting $\mathsf{Rel}(F)$, we get:

Corollary 3 ([26, Theorem 4])**.** *If F, T are* Set*-functors and (X, α, ξ) is a bialgebra for $\lambda\colon TF \Rightarrow FT$. The contextual closure functor Ctx is $\mathsf{Rel}(F)_\xi$-compatible.*

Proof. The canonical lifting $\mathsf{Rel}(-)$ is a 2-functor [17, Exercise 4.4.6]. Therefore $\overline{\lambda} = \mathsf{Rel}(\lambda)$ fulfils the assumption of Theorem 2. □

Our interest in Theorem 2 is not restricted to prove compatibility of up to Ctx. By taking non canonical liftings of T, one derives novel and effective up-to techniques, such as the *monotone contextual closure* and the *left-contextual closure* defined in Sections 5.1 and 5.2. In order to apply Theorem 2 for situations when either $\overline{T}$ or $\overline{F}$ is not the canonical relation lifting, one has to exhibit a $\overline{\lambda}$ sitting above λ. In Rel, such a $\overline{\lambda}$ exists if and only if for all relations $R \subseteq X^2$, the restriction of $\lambda_X \times \lambda_X$ to $\overline{T}\,\overline{F}R$ corestricts to $\overline{F}\,\overline{T}R$. A similar condition has to be checked for $\mathsf{Pred} \to \mathsf{Set}$.

4.4 Abstract GSOS

For several applications, it is convenient to consider natural transformations of a slightly different type $\lambda\colon T(F \times \mathrm{Id}) \Rightarrow F\mathtt{T}$, where $\mathtt{T}$ is the free monad over T. These are called *abstract GSOS specifications* since, as shown in [31], they generalise GSOS rules to any behaviour endofunctor F. Each such λ induces a distributive law $\lambda^\dagger\colon \mathtt{T}(F \times \mathrm{Id}) \Rightarrow (F \times \mathrm{Id})\mathtt{T}$ of the monad $\mathtt{T}$ over the copointed functor $F \times \mathrm{Id}$, whose bialgebras are the objects of our interest (see Appendix B.3). In order to prove compatibility via Theorem 2, one should exhibit a $\overline{\lambda^\dagger}$ sitting above $\lambda^\dagger$. The following lemma simplifies such a task.

Lemma 1. *Let $\overline{\lambda}\colon \overline{T}(\overline{F} \times \overline{\mathrm{Id}}) \Rightarrow \overline{F}\,\overline{\mathtt{T}}$ be a natural transformation sitting above $\lambda\colon T(F \times \mathrm{Id}) \Rightarrow F\mathtt{T}$. Then there exists a $\overline{\lambda^\dagger}\colon \overline{\mathtt{T}}(\overline{F} \times \overline{\mathrm{Id}}) \Rightarrow (\overline{F} \times \overline{\mathrm{Id}})\overline{\mathtt{T}}$ sitting above $\lambda^\dagger\colon \mathtt{T}(F \times \mathrm{Id}) \Rightarrow (F \times \mathrm{Id})\mathtt{T}$.*

For a bialgebra $(X, \alpha, \langle \xi, id\rangle)$, the existence of $\overline{\lambda^\dagger}$ ensures, via Theorem 2, compatibility w.r.t. $(\overline{F} \times \mathrm{Id})_{\langle \xi, id\rangle}$, which is not exactly $\overline{F}_\xi$. However, this difference is harmless in poset fibrations: coalgebras for the two functors coincide, and for any pointed functor A compatible with $(\overline{F} \times \mathrm{Id})_{\langle \xi, id\rangle}$, every $\overline{F}_\xi$-invariant up to A is also an $(\overline{F} \times \mathrm{Id})_{\langle \xi, id\rangle}$-invariant up to A.

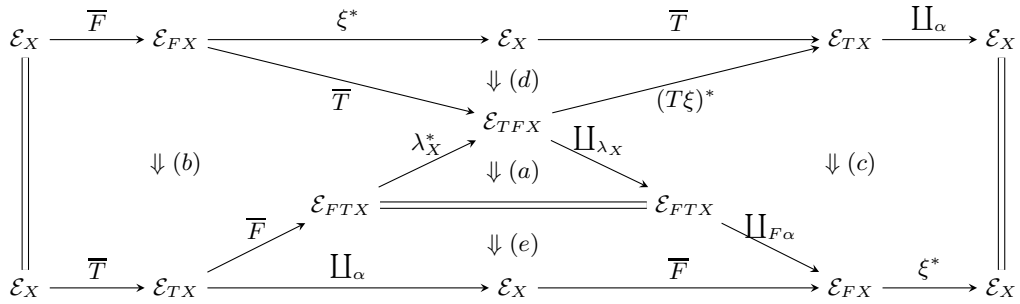


Figure 1. Compatibility of contextual closure in a fibration

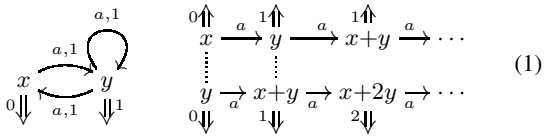
5. Examples

5.1 Inclusion of weighted automata

To illustrate how to instantiate the above framework, we consider weighted automata. We first give a short description of their coalgebraic treatment [6]. For a semiring $\mathtt{S}$ and a set X, we denote by $\mathtt{S}_\omega^X$ the set of functions $f\colon X \to \mathtt{S}$ with finite support. These functions can be thought of as linear combinations $\sum_{x\in X} f(x)\cdot x$, and in fact $\mathtt{S}_\omega^- : \mathsf{Set} \to \mathsf{Set}$ is the monad sending each set X to the free semi-module generated by X.

A *weighted automaton* over a semiring $\mathtt{S}$ with alphabet A is a pair $(X, \langle o, t\rangle)$, where X is a set of states, $o\colon X \to \mathtt{S}$ is an output function associating to each state its output weight and $t\colon X \to (\mathtt{S}_\omega^X)^A$ is a weighted transition relation. Denoting by F the functor $\mathtt{S}\times(-)^A$, weighted automata are thus coalgebras for the composite functor $F\mathtt{S}_\omega^-$. By the generalised powerset construction [28], they induce bialgebras for the functor F, the monad $\mathtt{S}_\omega^-$, and the distributive law $\lambda\colon \mathtt{S}_\omega^- F \Rightarrow F\mathtt{S}_\omega^-$ given for all sets X by $\lambda_X(\sum r_i(s_i,\varphi_i)) = \langle \sum r_i s_i, \lambda a.\sum r_i\varphi_i(a)\rangle$. Indeed every $(X,\langle o,t\rangle)$ induces a bialgebra $(\mathtt{S}_\omega^X, \mu, \langle o^\sharp, t^\sharp\rangle)$ where μ is the multiplication of $\mathtt{S}_\omega^-$ and $\langle o^\sharp, t^\sharp\rangle\colon \mathtt{S}_\omega^X \to \mathtt{S}\times(\mathtt{S}_\omega^X)^A$ is the linear extension of $\langle o,t\rangle$, defined as $(F\mu)\circ\lambda\circ(\mathtt{S}_\omega^{\langle o,t\rangle})$.

For a concrete example we take the semiring $\mathtt{R}^+$ of positive real numbers. A weighted automaton is depicted on the left below: arrows $x \xrightarrow{a,r} y$ mean that $t(x)(a)(y) = r$ and arrows $x \overset{r}{\Rightarrow}$ mean that $o(x) = r$.



(1)

On the right is depicted (part of) the corresponding bialgebra: states are elements of $(\mathtt{R}^+)_\omega^X$ (hereafter denoted by v, w), arrows $v \xrightarrow{a} w$ mean that $t^\sharp(v)(a) = w$ and arrows $v \overset{r}{\Rightarrow}$ mean that $o^\sharp(v) = r$.

Whenever $\mathtt{S}$ carries a partial order $\leq$, one can take the following lifting $\overline{F}\colon \mathsf{Rel} \to \mathsf{Rel}$ of F defined for $R \subseteq X^2$ by:

$$\{((r,\varphi),(s,\psi)) \mid r \leq s \ \wedge\ \forall a.\varphi(a)\ R\ \psi(a)\} \subseteq (FX)^2.$$

Then the functor $\overline{F}_{\langle o^\sharp, t^\sharp\rangle} = \langle o^\sharp, t^\sharp\rangle^* \circ \overline{F}\colon \mathsf{Rel}_X \to \mathsf{Rel}_X$ maps a relation $R \subseteq X^2$ into

$$\{(x,y) \mid o^\sharp(x) \leq o^\sharp(y) \wedge \forall a.t^\sharp(x)(a)\ R\ t^\sharp(y)(a)\}\ .$$

The carrier of a final $\overline{F}_{\langle o^\sharp, t^\sharp\rangle}$-coalgebra is a relation, denoted by $\precsim$, which we call *inclusion*: when $\mathtt{S}$ is the Boolean semiring, it coincides with language inclusion of non-deterministic automata.

For any two $v, w \in \mathtt{S}_\omega^X$, one can prove that $v \precsim w$ by exhibiting a $\overline{F}_{\langle o^\sharp, t^\sharp\rangle}$-invariant relating them. These invariants are usually infinite, since there are infinitely many reachable states in a bialgebra $\mathtt{S}_\omega^X$, even for finite X. This is the case when trying to check $x \precsim y$ in (1): we should relate infinitely many reachable states.

In order to obtain finite proofs, we exploit the algebraic structure of bialgebras and employ an up to context technique. To this end, we use the canonical lifting of the monad $\mathtt{S}_\omega^-$, defined for all $R \subseteq X^2$ as

$$\mathsf{Rel}(\mathtt{S}_\omega^-)(R) = \left\{ \left(\sum r_i x_i,\ \sum r_i y_i \right) \mid x_i\ R\ y_i \right\}$$

We prove that the endofunctor $Ctx = \coprod_\mu \circ \mathsf{Rel}(\mathtt{S}_\omega^-)$ is $\overline{F}_{\langle o^\sharp, t^\sharp\rangle}$-compatible by Theorem 2: it suffices to check that for any relation R on X, the restriction of $\lambda_X \times \lambda_X$ to $\mathsf{Rel}(\mathtt{S}_\omega^-)\overline{F}(R)$ corestricts to $\overline{F}\mathsf{Rel}(\mathtt{S}_\omega^-)(R)$. This is the case when for all $n_1, m_1, n_2, m_2 \in \mathtt{S}$ such that $n_1 \leq m_1$ and $n_2 \leq m_2$, we have (a) $n_1 + n_2 \leq m_1 + m_2$ and (b) $n_1 \cdot n_2 \leq m_1 \cdot m_2$. These two conditions are satisfied, e.g., in the Boolean semiring or in $\mathtt{R}^+$ and thus, in these cases, we can prove inclusion of automata using $\overline{F}_{\langle o^\sharp, t^\sharp\rangle}$-invariants up to Ctx. For example, in (1), the relation $R = \{(x,y),(y,x{+}y)\}$ is a $\overline{F}_{\langle o^\sharp, t^\sharp\rangle}$-invariant up to Ctx (to check this, just observe that $(x{+}y, x{+}2y) \in Ctx(R)$). This finite relation thus proves $x \precsim y$.

Unfortunately, condition (b) fails for the semiring $\mathtt{R}$ of (all) real numbers. Nevertheless, our framework allows us to define another up-to technique, which we call "up to *monotone* contextual closure". It is obtained by composing $\coprod_\mu$ and a non-canonical lifting of $\mathtt{R}_\omega^-$:

$$\overline{\mathtt{R}_\omega^-}(R) = \left\{ \left(\sum r_i x_i, \sum r_i y_i \right) \mid \begin{array}{l} r_i \geq 0 \Rightarrow x_i\ R\ y_i \\ r_i < 0 \Rightarrow y_i\ R\ x_i \end{array} \right\}$$

The restriction of $\lambda_X \times \lambda_X$ to $\overline{\mathtt{R}_\omega^-}\overline{F}(R)$ corestricts to $\overline{F}\overline{\mathtt{R}_\omega^-}(R)$. Therefore, by Theorem 2, the monotone contextual closure is $\overline{F}_{\langle o^\sharp, t^\sharp\rangle}$-compatible.

5.2 Divergence of processes

Up-to techniques can be instrumental in proving unary predicates. We take the fibration $\mathsf{Pred} \to \mathsf{Set}$ and we focus on the *divergence* predicate $\nu u.\langle\tau\rangle u$ defined on LTSs. The latter are coalgebras $\xi\colon X \to F(X)$ for the Set-functor $FX = \mathcal{P}_\omega(L \times X)$, where $L = \{a, \overline{a}, b, \overline{b} \ldots, \tau\}$ is a set of labels containing a special symbol τ and $\mathcal{P}_\omega$ is the *finite* powerset functor. We lift F to $\overline{F}^{\langle\tau\rangle}\colon \mathsf{Pred} \to \mathsf{Pred}$, defined for all sets X as

$$\overline{F}^{\langle\tau\rangle}_X(P \subseteq X) = \{S \in FX \mid \exists(\tau, x) \in S, x \in P\}.$$

The final $\overline{F}^{\langle\tau\rangle}_\xi$-coalgebra consists precisely of all the states in X satisfying $\nu u.\langle\tau\rangle u$. Hence, to prove that a state p diverges, it suffices to exhibit an $\overline{F}^{\langle\tau\rangle}_\xi$-invariant containing p.

When the LTS is specified by some process algebra, such invariants might be infinite. Suppose for instance that we have a parallel operator defined by the following GSOS rules and their symmetric counterparts:

$$\frac{x \xrightarrow{l} x'}{x|y \xrightarrow{l} x'|y} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{\overline{a}} y'}{x|y \xrightarrow{\tau} x'|y'} \ .$$

Consider the processes $p \xrightarrow{a} p|p$ and $q \xrightarrow{\overline{a}} q$. To prove that $p|q$ diverges, any invariant should include all the states that are on the infinite path $p|q \xrightarrow{\tau} (p|p)|q \xrightarrow{\tau} \dots$.

Instead, an intuitive proof would go as follows: assuming that $p|q$ diverges one has to prove that the τ successor $(p|p)|q$ also diverges. Rather than looking further for the τ-successors of $(p|p)|q$, observe that

(a) since $p|q$ diverges by hypothesis, then also $(p|q)|p$ diverges, and

(b) since $(p|q)|p$ is bisimilar (i.e., behavioural equivalent) to $(p|p)|q$, then also $(p|p)|q$ diverges.

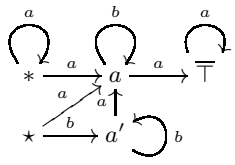
Formally, (b) corresponds to using the functor Bhv from Section 4.1. For (a) we define the *left contextual closure* functor as $Ctx^{\ell}(P \subseteq X) = \{(\dots(x|y_1)|\dots)|y_n \mid x \in P,\ y_i \in X\}$. Indeed, it is easy to see that $P = \{p|q\}$ is an $\overline{F}_{\xi}^{\langle\tau\rangle}$-invariant up to $Bhv \circ Ctx^{\ell}$, i.e, $P \subseteq \overline{F}_{\xi}^{\langle\tau\rangle} \circ Bhv \circ Ctx^{\ell}(P)$.

In order to prove soundness of this "up to behavioural equivalence and left contextual closure", we show compatibility of Bhv and Ctx^{ℓ} separately. For the former, we note that $\overline{F}^{\langle\tau\rangle}$ is defined exactly as in coalgebraic modal logic [9, 13] and thus $(\overline{F}^{\langle\tau\rangle}, F)$ is a fibration map: Theorem 1 applies. The functor Ctx^{ℓ} is defined just as Ctx, but instead of the canonical lifting of the endofunctor for binary operations $T(X) = X \times X$ we use the predicate lifting $\overline{T}(P \subseteq X) = P \times X \subseteq TX$. The conditions of Lemma 1 are met for the distributive law given by the above GSOS rules (see Appendix C). The functor Ctx^{ℓ} can be seen to be the composition $\coprod_{\mu} \circ \overline{\mathtt{T}}$ where $\overline{\mathtt{T}}$ is the free monad on $\overline{T}$ and μ is the multiplication of $\mathtt{T}$. We can thus apply Theorem 2 and obtain its compatibility.

5.3 Equivalence of nominal automata

Nominal automata and variants [4] have been considered as a means of studying languages over infinite alphabets, but also for the operational semantics of process calculi [22]. We refer the reader to [23] for background on the category Nom of nominal sets. These are sets equipped with actions of the group of permutations on a countable set $\mathtt{A}$ of names, satisfying an additional finite support condition.

Consider the nominal automaton below. The part reachable from state $*$ corresponds to [5, Example I.1].



It is important to specify how to read this drawing: the represented nominal automaton has as state space the orbit-finite nominal set $\{*\} + \{\star\} + \mathtt{A} + \mathtt{A}' + \{\top\}$, where $\mathtt{A}'$ is a copy of $\mathtt{A}$. It suffices in this case to give only one representative of each of the five orbits: we span all the transitions and states of the automaton by applying all possible finite permutations to those explicitly written. For example, the transition $a \xrightarrow{c} a$ is obtained from $a \xrightarrow{b} a$ by applying the transposition $(b\ c)$ to the latter.

With this semantics in mind, one can see that the state $*$ accepts the language of words in the alphabet $\mathtt{A}$ where some letter appears twice: it reads a word in $\mathtt{A}$, then it nondeterministically guesses that the next letter will appear a second time and verifies that this is indeed the case. The state $\star$ accepts the same language, in a different way: it reads a first letter, then guesses if this letter will be read again, or, if a distinct letter—nondeterministically chosen—will appear twice.

Formally, nominal automata are $F\mathcal{P}_{\omega}$-coalgebras $\langle o, t\rangle$ where $F\colon \mathsf{Nom} \to \mathsf{Nom}$ is given by $FX = 2 \times X^{\mathtt{A}}$ and the monad $\mathcal{P}_{\omega}$ is the finitary version of the power object functor in the category of nominal sets (mapping a nominal set to its finitely-supported orbit-finite subsets). In our example, $o(a) = 0$ and $t(a)$ is the following map:

$$t(a) = \begin{cases} b \mapsto \{a\} & b \# a \\ a \mapsto \{\top\} \end{cases}$$

By the generalised powerset construction [28], $\langle o, t\rangle$ induces a deterministic nominal automata, which is a bialgebra on $\mathcal{P}_{\omega}(X)$ with the algebraic structure given by union. To prove that $*$ and $\star$ accept the same language, we should play the bisimulation game in the determinisation of the automaton. However, the latter has *infinitely* many orbits and a rather complicated structure. A bisimulation constructed like this will thus have infinitely many orbits. Instead, we can show that the orbit-finite relation spanned by the four pairs

$$(\{*\}, \{\star\}),\ (\{a\}, \{a, a'\}),\ (\{\top\}, \{a, \top\}),\ (\{*\}, \mathtt{A}')$$

is a bisimulation up to congruence (w.r.t. union).

The soundness of this technique is established in Appendix D using the fibration $\mathsf{Rel}(\mathsf{Nom}) \to \mathsf{Nom}$ of equivariant relations. We derive the compatibility of contextual closure using Theorem 2, and compatibility of the transitive, symmetric, and reflexive closures using Proposition 2. Compatibility of congruence closure follows from Proposition 1(i).

6. Compositional Predicates

In this section we consider a structured way of defining coinductive predicates, by composing lifted functors. Assume a fibration $p\colon \mathcal{E} \to \mathcal{B}$ and a functor $\otimes\colon \mathcal{E} \times_{\mathcal{B}} \mathcal{E} \to \mathcal{E}$. Given two liftings $\overline{F_1}, \overline{F_2}\colon \mathcal{E} \to \mathcal{E}$ of the same endofunctor F on $\mathcal{B}$, one can then define a *composite* lifting $\otimes \circ \langle \overline{F_1}, \overline{F_2}\rangle$, which we denote by $\overline{F_1} \otimes \overline{F_2}$. We will instantiate this to the fibration $\mathsf{Rel} \to \mathsf{Set}$ with relational composition for $\otimes$, to define simulation and weak bisimulation as coinductive predicates.

One advantage of this approach is that the compatibility of up-to-context can be proved in a modular way.

Theorem 3. *Let $\overline{T}$ be a lifting of T having a $\gamma\colon \overline{T} \otimes \Rightarrow \otimes \overline{T}^2$ above $\mathrm{Id}\colon T \Rightarrow T$. Let both $\overline{F_1}$ and $\overline{F_2}$ be liftings of F. If $\lambda_1\colon \overline{T}\,\overline{F_1} \Rightarrow \overline{F_1}\,\overline{T}$ and $\lambda_2\colon \overline{T}\,\overline{F_2} \Rightarrow \overline{F_2}\,\overline{T}$ sit above the same $\lambda\colon TF \Rightarrow FT$, then there exists $\overline{\lambda}\colon \overline{T}(\overline{F_1} \otimes \overline{F_2}) \Rightarrow (\overline{F_1} \otimes \overline{F_2})\overline{T}$ above λ.*

Notice that the canonical lifting $\mathsf{Rel}(T)$ always satisfies the first hypothesis of the theorem when $\otimes$ is relational composition.

6.1 Simulation up-to

We recall simulations for coalgebras as introduced in [15]. An endofunctor F on Set is said to be *ordered* if it factors through the forgetful functor from Pre (the category of preorders) to Set: this means that for every X, FX is equipped with a preorder $\sqsubseteq_{FX}$. An ordered functor gives rise to a *constant* relation lifting $\overline{\sqsubseteq}$ of F defined as $\overline{\sqsubseteq}(R \subseteq X^2) = \sqsubseteq_{FX}$. Then the *lax relation lifting*

$\mathsf{Rel}(F)^{\sqsubseteq}$ is defined as

$$\mathsf{Rel}(F)^{\sqsubseteq} = \overline{\sqsubseteq} \otimes \mathsf{Rel}(F) \otimes \overline{\sqsubseteq}$$

where $\otimes$ is relational composition. For a coalgebra $\xi\colon X \to FX$, the coalgebras for the endofunctor $\xi^* \circ \mathsf{Rel}(F)^{\sqsubseteq}_X$—which we denote as $\mathsf{Rel}(F)^{\sqsubseteq}_\xi$—are called *simulations*; the final one is called *similarity*. We list two examples of ordered functors and their associated notion of simulations, and refer to [15] for many more.

Example 3. *For weighted automata on a semiring $\mathbb{S}$ equipped with a partial order $\leq$, the functor $FX = \mathbb{S} \times X^A$ is ordered with $\sqsubseteq_{FX}$ defined as $(s,\phi) \sqsubseteq_{FX} (r,\psi)$ iff $s \leq r$ and $\phi = \psi$. It is immediate to see that $\mathsf{Rel}(F)^{\sqsubseteq}$ coincides with the lifting $\overline{F}$ defined in Section 5.1.*

For LTSs, the functor $FX = \mathcal{P}_\omega(A \times X)$ is ordered with subset inclusion $\subseteq$. In this case a simulation is a relation $R \subseteq X^2$ such that for all $(x,y) \in R$: if $x \xrightarrow{a} x'$ then there exists y' such that $x' \xrightarrow{a} y'$ and $x' R y'$.

An ordered functor F is called *stable* if $(\mathsf{Rel}(F)^{\sqsubseteq}, F)$ is a fibration map [15]. Since polynomial functors are stable, as well as the one for LTSs [15], the following results hold for the coalgebras in Example 3.

Proposition 3. *If F is a stable ordered functor, then Bhv, Slf, and Trn are $\mathsf{Rel}(F)^{\sqsubseteq}_\xi$-compatible.*

Proof. Compatibility of *Bhv* comes from Theorem 1. Compatibility of *Slf* and *Trn* comes from Proposition 2: stable functors satisfy $(***)$ [15, Lemma 5.3]. □

We proceed to consider the compatibility of up to context, for which we assume an abstract GSOS specification $\lambda\colon T(F \times \mathrm{Id}) \Rightarrow F\mathtt{T}$. By Theorem 3, proving compatibility w.r.t. $\mathsf{Rel}(F)^{\sqsubseteq}_\xi$ is reduced to proving compatibility w.r.t. its components $\mathsf{Rel}(F)$ and $\overline{\sqsubseteq}$. For the former, compatibility comes immediately from the proof of Corollary 3. For the latter, we need to assume that the abstract GSOS specification is *monotone*, i.e, such that for any set X, the restriction of $\lambda_X \times \lambda_X$ to $\mathsf{Rel}(T)(\sqsubseteq_{FX} \times \Delta_X)$ corestricts to $\sqsubseteq_F \mathtt{T}X$. If T is a polynomial functor representing a signature, then this means that for any operator σ (of arity n) we have

$$\frac{b_1 \sqsubseteq_{FX} c_1 \quad \dots \quad b_n \sqsubseteq_{FX} c_n}{\lambda_X(\sigma(\mathbf{b},\mathbf{x})) \sqsubseteq_F \mathtt{T}X\ \lambda_X(\sigma(\mathbf{c},\mathbf{x}))}$$

where $\mathbf{b},\mathbf{x} = (b_1,x_1),\dots,(b_n,x_n)$ with $x_i \in X$ and similarly for $\mathbf{c},\mathbf{x}$. If $\sqsubseteq$ is the order on the functor for LTSs, monotonicity corresponds to the *positive GSOS* format [12] which, as expected, is GSOS [3] without negative premises. Monotonicity turns out to be precisely the condition needed to apply Lemma 1, yielding

Proposition 4. *Let λ be a monotone abstract GSOS specification and $(X,\alpha,\langle\xi,id\rangle)$ be a $\lambda^\dagger$-bialgebra. Then Ctx is $(\mathsf{Rel}(F)^{\sqsubseteq} \times \mathrm{Id})_{\langle\xi,id\rangle}$-compatible.*

6.2 Weak bisimulation-up-to

A *weak bisimulation* is a relation $R \subseteq X^2$ on the states of an LTS such that for every pair $(x,y) \in R$: (1) if $x \xrightarrow{l} x'$ then $y \stackrel{l}{\Rightarrow} y'$ with $(x',y') \in R$ and (2) if $y \xrightarrow{l} y'$ then $x \stackrel{l}{\Rightarrow} x'$ with $(x',y') \in R$. Here $\to$ and $\Rightarrow$ are two LTSs, i.e., coalgebras for the functor $FX = \mathcal{P}_\omega(L \times X)$, and $\Rightarrow$ is the *saturation* [21] of $\to$. Weak bisimilarity can alternatively be reduced to strong bisimilarity on $\Rightarrow$, but the associated proof method is rather tedious. To remain faithful to the above definition, we define weak bisimulations via the following lifting of $F \times F$:

$$\overline{F \times F} = \rho \otimes \mathsf{Rel}(F \times F)^{[\supseteq\subseteq]} \ ,$$

where ρ is the constant functor defined as $\rho(R \subseteq X^2) = \{((U,V),(V,U)) \mid U,V \in FX\}$ and $\mathsf{Rel}(F \times F)^{[\supseteq\subseteq]}$ is the lax relation lifting of $F \times F$ for the ordering $(U_1,V_1)[\supseteq\subseteq](U_2,V_2)$ iff $U_2 \subseteq U_1$ and $V_1 \subseteq V_2$.

For an intuition, observe that an $F \times F$-coalgebra is a pair $\langle\xi_1,\xi_2\rangle\colon X \to FX \times FX$ of LTSs that we denote with $\to_1$ and $\to_2$. An invariant for $\mathsf{Rel}(F \times F)^{[\supseteq\subseteq]}_{\langle\xi_1,\xi_2\rangle}$ is a relation $R \subseteq X^2$ such that for each $(x,y) \in R$: (1) if $y \xrightarrow{l}_1 y'$ then $x \xrightarrow{l}_1 x'$ with $x' R y'$, and (2) if $x \xrightarrow{l}_2 x'$ then $y \xrightarrow{l}_2 y'$ with $x' R y'$. Composing with ρ "flips" the LTSs $\to_1$ and $\to_2$: an invariant for $\overline{F \times F}_{\langle\xi_1,\xi_2\rangle}$ is now an $R \subseteq X^2$ such that: (1) if $y \xrightarrow{l}_1 y'$ then $x \xrightarrow{l}_2 x'$ with $x' R y'$, and (2) if $x \xrightarrow{l}_1 x'$ then $y \xrightarrow{l}_2 y'$ with $x' R y'$. It is easy to see that for $\langle\xi_1,\xi_2\rangle = \langle\to,\Rightarrow\rangle$, coalgebras for $\overline{F \times F}_{\langle\xi_1,\xi_2\rangle}$ are weak bisimulations and the final coalgebra is weak bisimilarity.

In Appendix E, we show that $(\overline{F \times F}, F)$ is a fibration map and by Theorem 1 we now obtain the following.

Corollary 4. *Bhv is $\overline{F \times F}_{\langle\xi_1,\xi_2\rangle}$-compatible.*

For $\langle\xi_1,\xi_2\rangle = \langle\to,\Rightarrow\rangle$, behavioural equivalence is simply strong bisimilarity. Consequently, Corollary 4 actually gives the compatibility of weak bisimulation up to strong bisimilarity [25]. One could wish to use up to *Slf* or up to *Trn* for weak bisimulations. However, the condition $(***)$ from Section 4.2 fails, and indeed, weak bisimulations up to weak bisimilarity or up to transitivity are not sound [25].

For up to context, we use Theorem 3 to reduce compatibility w.r.t. $\overline{F \times F}$ to compatibility w.r.t. ρ and $\mathsf{Rel}(F \times F)^{[\supseteq\subseteq]}$ (for which we can reuse the result of the previous section).

Proposition 5. *Let $\lambda\colon T(F \times \mathrm{Id}) \Rightarrow F\mathtt{T}$ be a positive GSOS specification and $(X,\alpha,\langle\xi_1,id\rangle)$ and $(X,\alpha,\langle\xi_2,id\rangle)$ be two $\lambda^\dagger$-bialgebras then Ctx is $(\overline{F \times F} \times \mathrm{Id})_{\langle\xi_1,\xi_2,id\rangle}$-compatible.*

The above proposition requires both $\to$ and $\Rightarrow$ to be models [1] of the same positive GSOS specification λ. This means that the rules of λ should be sound for both $\to$ and $\Rightarrow$. For instance, in the case of CCS, $\Rightarrow$ is not a model of λ because the rule for nondeterministic choice is not sound for $\Rightarrow$. Nevertheless, we can use our framework to prove the compatibility of weak bisimulation up to contextual closure w.r.t. the remaining operators.

7. Directions for future work

Our nominal automata example leads us to expect that the framework introduced in this paper will lend itself to obtaining a clean theory of up-to techniques for name-passing process calculi. For instance, we would like to understand whether the congruence rule format proposed by Fiore and Staton [11] can fit in our setting: this would provide general conditions under which up-to techniques related to name substitution are sound in such calculi.

Another interesting research direction is suggested by the divergence predicate we studied in Section 5.2. Other formulas of (coalgebraic) modal logic [9] can be expressed by taking different predicate liftings, and yield different families of compatible functors. This suggests a connection with the proof systems in [10, 29]: we can regard proofs in those systems as invariants up to some compatible functors. By using our framework and the logical distributive laws of [18], we hope to obtain a systematic way to derive or enhance such proof systems, starting from a given abstract GSOS specification.

Acknowledgments

We are grateful to the anonymous reviewers for their constructive comments, and in particular to the one who noticed that our no-

tion of compatible functor was just an instance of morphisms of endofunctors. We would also like to thank Alexander Kurz and Alexandra Silva for the stimulating discussions that eventually led to the example on nominal automata; Marcello Bonsangue, Tom Hirschowitz and Henning Kerstan for comments on preliminary versions of the paper; Ichiro Hasuo for the inspiring talk at Bellairs Workshop on Coalgebras.

References

[1] L. Aceto, W. Fokkink, and C. Verhoef. Structural operational semantics. In *Handbook of Process Algebra*, pages 197–292. Elsevier, 2001.

[2] F. Bartels. Generalised coinduction. *MSCS*, 13(2):321–348, 2003.

[3] B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced. In *POPL*, pages 229–239. ACM, 1988.

[4] M. Bojanczyk, B. Klin, and S. Lasota. Automata with group actions. In *LICS*, pages 355–364, 2011.

[5] M. Bojanczyk, B. Klin, S. Lasota, and S. Torunczyk. Turing machines with atoms. In *LICS*, pages 183–192, 2013.

[6] F. Bonchi, M. Bonsangue, M. Boreale, J. Rutten, and A. Silva. A coalgebraic perspective on linear weighted automata. *Inf. and Comp.*, 211:77–105, 2012.

[7] F. Bonchi and D. Pous. Checking NFA equivalence with bisimulations up to congruence. In *POPL*, pages 457–468. ACM, 2013.

[8] D. Caucal. Graphes canoniques de graphes algébriques. *ITA*, 24:339–352, 1990.

[9] C. Cîrstea, A. Kurz, D. Pattinson, L. Schröder, and Y. Venema. Modal logics are coalgebraic. *Comput. J.*, 54(1):31–41, 2011.

[10] M. Dam. Compositional proof systems for model checking infinite state processes. In *CONCUR*, volume 962 of *LNCS*, pages 12–26. Springer, 1995.

[11] M. Fiore and S. Staton. A congruence rule format for name-passing process calculi. *Inf. and Comp.*, 207(2):209–236, 2009.

[12] M. Fiore and S. Staton. Positive structural operational semantics and monotone distributive laws. In *CMCS*, page 8, 2010.

[13] I. Hasuo, K. Cho, T. Kataoka, and B. Jacobs. Coinductive predicates and final sequences in a fibration. In *MFPS*, 2013.

[14] C. Hermida and B. Jacobs. Structural induction and coinduction in a fibrational setting. *Inf. and Comp.*, 145:107–152, 1997.

[15] J. Hughes and B. Jacobs. Simulations in coalgebra. *TCS*, 327(1-2):71–108, 2004.

[16] B. Jacobs. *Categorical Logic and Type Theory*. Elsevier, 1999.

[17] B. Jacobs. Introduction to coalgebra. Towards mathematics of states and observations, 2014. Draft.

[18] B. Klin. Bialgebraic operational semantics and modal logic. In *LICS*, pages 336–345. IEEE, 2007.

[19] M. Lenisa. From set-theoretic coinduction to coalgebraic coinduction: some results, some problems. *ENTCS*, 19:2–22, 1999.

[20] M. Lenisa, J. Power, and H. Watanabe. Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads. *ENTCS*, 33:230–260, 2000.

[21] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[22] U. Montanari and M. Pistore. History-dependent automata: An introduction. In *SFM*, LNCS, pages 1–28. Springer, 2005.

[23] A. M. Pitts. *Nominal Sets*. Cambridge University Press, 2013.

[24] D. Pous. Complete lattices and up-to techniques. In *APLAS*, volume 4807 of *LNCS*, pages 351–366. Springer, 2007.

[25] D. Pous and D. Sangiorgi. Enhancements of the bisimulation proof method. In *Advanced Topics in Bisimulation and Coinduction*, pages 233–289. Cambridge University Press, 2012.

[26] J. Rot, F. Bonchi, M. Bonsangue, D. Pous, J. Rutten, and A. Silva. Enhanced coalgebraic bisimulation. *To appear in MSCS*, 2014. Available at `http://www.liacs.nl/~jrot/up-to.pdf`.

[27] D. Sangiorgi. On the bisimulation proof method. *MSCS*, 8:447–479, 1998.

[28] A. Silva, F. Bonchi, M. Bonsangue, and J. Rutten. Generalizing the powerset construction, coalgebraically. In *FSTTCS*, pages 272–283, 2010.

[29] A. Simpson. Sequent calculi for process verification: Hennessy-Milner logic for an arbitrary GSOS. *JLAP*, 60–61:287–322, 2004.

[30] S. Staton. Relating coalgebraic notions of bisimulation. *LMCS*, 7(1), 2011.

[31] D. Turi and G. D. Plotkin. Towards a mathematical operational semantics. In *LICS*, pages 280–291. IEEE, 1997.

A. Proofs for Section 3

The following Proposition generalises the compositionality results for compatible functions on lattices, see [24] or [25, Proposition 6.3.11].

Proposition 1. *Compatible functors are closed under the following constructions:*

(i) composition: if A is (B,C)-compatible and A' is (C,D)-compatible, then $A' \circ A$ is (B,D)-compatible;
(ii) pairing: if $(A_i)_{i\in\iota}$ are (B,C)-compatible, then $\langle A_i\rangle_{i\in\iota}$ is (B,C^ι)-compatible;
(iii) product: if A is (B,C)-compatible and A' is (B',C')-compatible, then $A \times A'$ is $(B\times B', C\times C')$-compatible;

Moreover, for an endofunctor $B\colon \mathcal{C} \to \mathcal{C}$,

(iv) the identity functor $Id\colon \mathcal{C} \to \mathcal{C}$ is B-compatible;
(v) the constant functor to the carrier of any B-coalgebra is B-compatible, in particular the final one if it exists;
(vi) the coproduct functor $\coprod\colon \mathcal{C}^\iota \to \mathcal{C}$ is (B^ι, B)-compatible.

Proof. (i) Given $\gamma : AB \Rightarrow CA$ and $\gamma' : A'C \Rightarrow DA'$ we obtain

$$A'AB \xRightarrow{A'\gamma} A'CA \xRightarrow{\gamma' A} DA'A$$

(ii) Given natural transformations $\gamma_i : A_iB \Rightarrow CA_i$ for all $i \in \iota$ we obtain a natural transformation

$$\begin{array}{ccc} \langle A_i\rangle_{i\in\iota} B & & C^\iota\langle A_i\rangle_{i\in\iota} \\ \| & & \| \\ \langle A_iB\rangle_{i\in\iota} & \xRightarrow{\langle\gamma_i\rangle_{i\in\iota}} & \langle CA_i\rangle_{i\in\iota} \end{array}$$

(iii) Given $\gamma : AB \Rightarrow CA$ and $\gamma' : A'B' \Rightarrow C'A'$ we construct $\gamma \times \gamma' : (A \times A')(B \times B') \Rightarrow (C \times C')(A \times A')$.

Items (iv), (v) and (vi) are trivial. For example, the latter is immediate using the universal property of the coproduct. □

B. Proofs for Section 4

The next simple Lemma about liftings in fibrations will be used throughout this appendix, e.g., to prove Proposition 2, but also Theorem 2.

Lemma 2. *Let $p : \mathcal{E} \to \mathcal{B}$ and $p' : \mathcal{E}' \to \mathcal{B}$ be two fibrations and assume $\overline{T} : \mathcal{E} \to \mathcal{E}'$ is the lifting of a functor $T : \mathcal{B} \to \mathcal{B}$. Consider a $\mathcal{B}$-morphism $f : X \to Y$. Then there exists a natural transformation:*

$$\theta : \overline{T} \circ f^* \Rightarrow (Tf)^* \circ \overline{T} : \mathcal{E}_Y \to \mathcal{E}'_{TX}.$$

Proof. In order to define θ_R for some R in $\mathcal{E}_Y$, we use the universal property of the Cartesian lifting $\widetilde{Tf}_{\overline{T}(R)}$. In a diagram:

$$\begin{array}{ccc} \overline{T}(f^*(R)) & & \\ \theta_R \,\vdots & \searrow^{\overline{T}(\widetilde{f}_R)} & \\ (Tf)^*(\overline{T}R) & \xrightarrow[\widetilde{Tf}_{\overline{T}R}]{} & \overline{T}R \end{array} \qquad (2)$$

$$TX \xrightarrow{Tf} TY$$

□

Lemma 3. *Let $p : \mathcal{E} \to \mathcal{B}$ be a bifibration and assume $\overline{F} : \mathcal{E} \to \mathcal{E}$ is the lifting of a functor $F : \mathcal{B} \to \mathcal{B}$. Consider a $\mathcal{B}$-morphism $f : X \to Y$. Then there exists a natural transformation:*

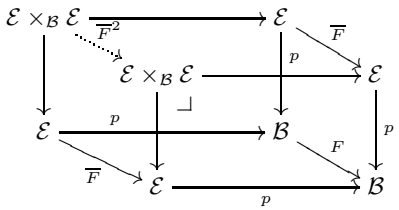
$$\rho : \textstyle\coprod_{Ff} \circ \overline{F} \Rightarrow \overline{F} \circ \coprod_f : \mathcal{E}_X \to \mathcal{E}_{FY}.$$

Proof. The proof uses the universal property of the opcartesian liftings. Equivalently, from Lemma 2 we have a natural transformation $\overline{F}f^* \Rightarrow (Ff)^*\overline{F}$. Taking the adjoint transpose via $\coprod_{Ff} \dashv (Ff)^*$ we get a natural transformation $\coprod_{Ff} \overline{F}f^* \Rightarrow \overline{F}$. A further adjoint transpose via the adjunction $\coprod_f \dashv f^*$ yields the desired $\rho : \coprod_{Ff}\overline{F} \Rightarrow \overline{F}\coprod_f$. □

B.1 Proofs for Section 4.2

In this section we prove Proposition 2. For the sake of clarity we explain how $\overline{F}^n$ is defined for $n = 2$. Recall that $\mathcal{E} \times_\mathcal{B} \mathcal{E}$ is obtained as a pullback of p along p in Cat.

For a lifting $\overline{F}$ of F, the functor $\overline{F}^2$ makes the next diagram commute.



This means that on each fibre we have

$$\overline{F}^n_X = (\overline{F}_X)^n : \mathcal{E}^n_X \to \mathcal{E}^n_{FX}.$$

As a consequence of Lemma 2 we obtain:

Lemma 4. *Let $p : \mathcal{E} \to \mathcal{B}$ and assume $G : \mathcal{E}^{\times_\mathcal{B} n} \to \mathcal{E}$ is a lifting of the identity on $\mathcal{B}$. If $f : X \to Y$ is a $\mathcal{B}$-morphism, there is a canonical natural transformation*

$$\theta : G(f^*)^n \Rightarrow f^*G : \mathcal{E}_Y \to \mathcal{E}_{GX}.$$

Proof. This is an instance of Lemma 2 for $T = \mathrm{Id}$ and $\overline{T} = G$. We also use that the Cartesian lifting of a $\mathcal{B}$-morphism f in $\mathcal{E}^{\times_\mathcal{B} n}$ is $(f^*)^n$, where f^* is the Cartesian lifting in $\mathcal{E}$. □

Proposition 2. *Let $G\colon \mathcal{E}^{\times_\mathcal{B} n} \to \mathcal{E}$ be a lifting of the identity on $\mathcal{B}$ such that there exists a natural transformation $G\overline{F}^n \Rightarrow \overline{F}G$. Then G_X is $\overline{F}_\xi$-compatible.*

Proof. Consider the natural transformation obtained as the composition

$$G_X(\xi^*)^n(\overline{F})^n \Rightarrow \xi^* G_X(\overline{F})^n \Rightarrow \xi^*\overline{F}G_X$$

and use that $(\xi^* \circ \overline{F})^n = (\xi^*)^n \circ (\overline{F})^n$. The first natural transformation comes from Lemma 4 applied for ξ. □

B.2 Proofs for Section 4.3

In the next Theorem we only use that the fibration $p : \mathcal{E} \to \mathcal{B}$ is a bifibration and is split. [1]

Theorem 2. *Let $\overline{T}, \overline{F}\colon \mathcal{E} \to \mathcal{E}$ be liftings of T and F. If $\overline{\lambda}\colon \overline{T}\,\overline{F} \Rightarrow \overline{F}\,\overline{T}$ is a natural transformation sitting above λ, then $\coprod_\alpha \circ \overline{T}$ is $\overline{F}_\xi$-compatible.*

[1] The Beck-Chevalley condition is not required for the functors $\coprod_f$.

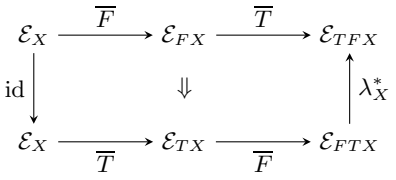
Proof. We exhibit a natural transformation

$$\coprod_\alpha \circ \overline{T} \circ \xi^* \circ \overline{F} \Rightarrow \xi^* \circ \overline{F} \circ \coprod_\alpha \circ \overline{T}.$$

This is achieved in Figure 1 by pasting five natural transformations, obtained as follows:

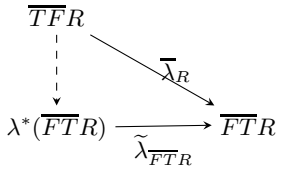
(a) is the counit of the adjunction $\coprod_{\lambda_X} \dashv \lambda_X^*$.
(b) comes from $\overline{\lambda}$ being a lifting of λ, see Lemma 5.
(c) comes from the bialgebra condition, the fibration being split, and the units and counits of the adjunctions $\coprod_\alpha \dashv \alpha^*$, $\coprod_{F\alpha} \dashv (F\alpha)^*$, and $\coprod_{\lambda_X} \dashv \lambda_X^*$. See Lemma 6.
(d) arises since $\overline{T}$ is a lifting of T, using the universal property of the Cartesian lifting $(T\xi)^*$, see Lemma 2.
(e) comes from $\overline{F}$ being a lifting of F, combined with the unit and counit of the adjunction $\coprod_\alpha \dashv \alpha^*$, see Lemma 3. □

Lemma 5. *Consider a fibration $p : \mathcal{E} \to \mathcal{B}$, two $\mathcal{B}$-endofunctors F, T with corresponding liftings $\overline{T}, \overline{F}$. Assume $\lambda : TF \Rightarrow FT$ is a natural transformation and $\overline{\lambda} : \overline{T}\overline{F} \Rightarrow \overline{F}\overline{T}$ sits above λ. Then there exists a 2-cell as in the diagram below:*



(3)

Proof. For $R \in \mathcal{E}_{FTX}$ the R-component of the required natural transformation is the dashed line in the diagram below and is obtained using the universal property of the Cartesian lifting of λ_X.

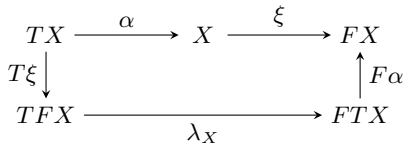


(4)

$$TFX \xrightarrow{\lambda_X} FTX$$

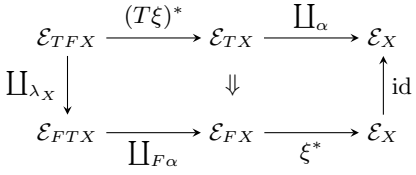
The naturality in R can be easily checked and is a consequence of the uniqueness of the factorisation. □

Lemma 6. *Given (X, α, ξ) an λ-bialgebra as in (5)*



(5)

and $p : \mathcal{E} \to \mathcal{B}$ a split fibration, there exists a 2-cell



(6)

Proof. We obtain the required natural transformation as the composite of the natural transformations of (7) below.

$$\begin{array}{ll}
\coprod_\alpha \circ (T\xi)^* & \\
\Downarrow & (\coprod_\lambda \dashv \lambda^*) \\
\coprod_\alpha \circ (T\xi)^* \circ \lambda^* \circ \coprod_\lambda & \\
\Downarrow & (\coprod_{F\alpha} \dashv (F\alpha)^*) \\
\coprod_\alpha \circ (T\xi)^* \circ \lambda^* \circ (F\alpha)^* \circ \coprod_{F\alpha} \circ \coprod_\lambda & \\
\Downarrow & (\text{bialg}) \\
\coprod_\alpha \circ \alpha^* \circ \xi^* \circ \coprod_{F\alpha} \circ \coprod_\lambda & \\
\Downarrow & (\coprod_\alpha \dashv \alpha^*) \\
\xi^* \circ \coprod_{F\alpha} \circ \coprod_\lambda &
\end{array} \tag{7}$$

Except for the third one, these 2-cells are obtained from the units or counits of the adjunctions recalled on the right column. The third natural transformation is actually an isomorphism and arises from (X, α, ξ) being a bialgebra and the fibration being split. □

B.3 Proofs for Section 4.4

In this section we will prove Lemma 1. First we recall some basic facts on the free monad $\mathtt{T}$ over a functor T on some category $\mathcal{C}$.

Assuming T has free algebras over any X in $\mathcal{C}$ one can show that the free monad $\mathtt{T}$ over T exists. We can define $\mathtt{T}X$ as the free T-algebra on X, or equivalently, as the initial algebra for the functor $X + T(-)$. Thus for each X in $\mathcal{C}$ one has an isomorphism

$$[\eta_X, \kappa_X] \colon X + T\mathtt{T}X \to \mathtt{T}X.$$

The η above gives the unit of the monad $\mathtt{T}$. The monad multiplication $\mu : \mathtt{T}\mathtt{T}X \to \mathtt{T}X$ is given as the unique morphism obtained by equipping $\mathtt{T}X$ with the $\mathtt{T}X + T(-)$-algebra structure $[\mathrm{id}, \kappa_X]$.

Recall from [31] that there exists a bijective correspondence between natural transformations

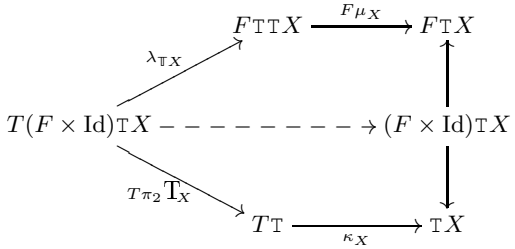
$$\lambda : T(F \times \mathrm{Id}) \Rightarrow F\mathtt{T}$$

and distributive laws

$$\lambda^\dagger \colon \mathtt{T}(F \times \mathrm{Id}) \Rightarrow (F \times \mathrm{Id})\mathtt{T}.$$

We briefly recall here how $\lambda^\dagger$ is obtained from λ. For X in $\mathcal{B}$, we equip $(F \times \mathrm{Id})\mathtt{T}X$ with a $FX \times X + T(-)$-algebra structure, given by the sum of :

$$FX \times X \xrightarrow{(F\times \mathrm{Id})\eta_X} (F \times \mathrm{Id})\mathtt{T}X$$



Hence $\lambda^\dagger_X$ is defined as the unique $(F \times \mathrm{Id})X + T(-)$-algebra morphism:

$$\begin{array}{ccc} T\mathtt{T}(F \times \mathrm{Id})X & \xrightarrow{T(\lambda^\dagger_X)} & T(F \times \mathrm{Id})\mathtt{T}X \\ {\scriptstyle \kappa_{(F\times\mathrm{Id})X}}\downarrow & & \downarrow{\scriptstyle \langle F\mu_X \lambda_{\mathtt{T}X} \kappa_X (T\pi_2 \mathbb{T})_X\rangle} \\ \mathtt{T}(F \times \mathrm{Id})X & \xrightarrow{\lambda^\dagger_X} & (F \times \mathrm{Id})\mathtt{T}X \\ {\scriptstyle \eta_{(F\times\mathrm{Id})X}}\uparrow & \nearrow_{(F\times\mathrm{Id})\eta_X} & \\ (F \times \mathrm{Id})X & & \end{array} \tag{8}$$

The following technical lemma is needed to establish that whenever the lifting of $\overline{T}$ of a functor T has free algebras, the free monad over $\overline{T}$ is the lifting of the free monad over T.

Lemma 7. *Consider a lifting $\overline{T}$ of a $\mathcal{B}$-endofunctor T and assume $\overline{T}$ has free algebras.*

1. *The functor $p : \mathcal{E} \to \mathcal{B}$ has a right adjoint $\mathbf{1} : \mathcal{B} \to \mathcal{E}$ inducing an adjunction*[2]

$$\mathsf{Alg}(\overline{T}) \underset{\mathsf{Alg}(\mathbf{1})}{\overset{\mathsf{Alg}(p)}{\rightleftarrows}} \mathsf{Alg}(T) \quad (\perp)$$

2. *The functor $\mathsf{Alg}(p)$ preserves the initial algebras.*
3. *When $P \in \mathcal{E}_X$ for some X in $\mathcal{B}$, the free $\overline{T}$-algebra over P sits above the free T-algebras over X.*
4. *The free monad $\overline{\mathtt{T}}$ over $\overline{T}$ exists and is a lifting of the free monad $\mathtt{T}$ over T.*

Proof. 1. Since the fibration considered here is bicartesian, one can define $\mathbf{1}(X)$ as the terminal object in $\mathcal{E}_X$. Then the statement of this item is an immediate consequence of [14, Theorem 2.14].
2. follows because $\mathsf{Alg}(p)$ is a left adjoint.
3. follows from item 1) applied for the lifting $P + \overline{T}$ of $X + T$.
4. is an immediate consequence of item 3). □

Lemma 1. *Consider a lifting $\overline{T}$ of a $\mathcal{B}$-endofunctor T and assume $\overline{T}$ has free algebras. Let $\overline{\lambda} : \overline{T}(\overline{F} \times \overline{\mathrm{Id}}) \Rightarrow \overline{F\mathtt{T}}$ be a natural transformation sitting above $\lambda : T(F \times \mathrm{Id}) \Rightarrow F\mathtt{T}$. Then $\overline{\lambda}^\dagger : \overline{\mathtt{T}}(\overline{F} \times \overline{\mathrm{Id}}) \Rightarrow (\overline{F} \times \overline{\mathrm{Id}})\overline{\mathtt{T}}$ sits above $\lambda^\dagger : \mathtt{T}(F \times \mathrm{Id}) \Rightarrow (F \times \mathrm{Id})\mathtt{T}$.*

Proof. We know that $\mathtt{T}X$ is the free T-algebra on X. Let

$$[\eta_X, \kappa_X] : X + T\mathtt{T}X \to \mathtt{T}X$$

denote the initial $X + T(-)$-algebra. Similarly, let

$$[\overline{\eta}_P, \overline{\kappa}_P] : P + \overline{T\mathtt{T}}P \to \overline{\mathtt{T}}P$$

denote the initial $P + \overline{T}(-)$-algebra. By Lemma 7 we know that when $P \in \mathcal{E}_X$ we have that $[\overline{\eta}_P, \overline{\kappa}_P]$ is a lifting of $[\eta_X, \kappa_X]$.

[2] The functor Alg stems from the 2-categorical notion of *inserter*, see [33] or [14, Theorem 2.14,Appendix A.5] for a concise exposition.

For $P \in \mathcal{E}_X$ the map $\overline{\lambda}^\dagger_P$ is defined similarly to (8), as the unique map such that:

$$\begin{array}{ccc} \overline{T\mathtt{T}}(\overline{F} \times \overline{\mathrm{Id}})P & \xrightarrow{\overline{T}(\overline{\lambda}^\dagger_P)} & \overline{T}(\overline{F} \times \overline{\mathrm{Id}})\overline{\mathtt{T}}P \\ {\scriptstyle \overline{\kappa}_{(\overline{F}\times\overline{\mathrm{Id}})P}}\downarrow & & \downarrow{\scriptstyle \langle \overline{F}\overline{\mu}_P \overline{\lambda}_{\overline{\mathtt{T}}P} \overline{\kappa}_P (\mathtt{T}\pi_2 \overline{\mathbb{T}})_P\rangle} \\ \overline{\mathtt{T}}(\overline{F} \times \overline{\mathrm{Id}})P & \xrightarrow{\overline{\lambda}^\dagger_P} & (\overline{F} \times \overline{\mathrm{Id}})\overline{\mathtt{T}}P \\ {\scriptstyle \overline{\eta}_{(\overline{F}\times\overline{\mathrm{Id}})P}}\uparrow & \nearrow_{(\overline{F}\times\overline{\mathrm{Id}})\overline{\eta}_P} & \\ (\overline{F} \times \overline{\mathrm{Id}})P & & \end{array} \tag{9}$$

By Lemma 7 we have that the $(\overline{F} \times \overline{\mathrm{Id}})P + \overline{T}(-)$-algebras $\overline{\mathtt{T}}(\overline{F} \times \overline{\mathrm{Id}})P$ and $(\overline{F} \times \overline{\mathrm{Id}})\overline{\mathtt{T}}P$ of diagram (9) sit above the $(F \times \mathrm{Id})X + T(-)$-algebras $\mathtt{T}(F \times \mathrm{Id})X$, respectively $(F \times \mathrm{Id})\mathtt{T}X$ of diagram (8). By uniqueness of $\lambda^\dagger_X$ it follows that $\overline{\lambda}^\dagger_P$ sits above $\lambda^\dagger_X$. □

C. Details on Divergence

In this appendix, we discuss some details for showing compatibility of Ctx^ℓ that were omitted in the main text for lack of space.

First of all, observe that the GSOS rules defining the parallel operator corresponds to a distributive law $\lambda \colon T(F \times \mathrm{Id}) \Rightarrow F\mathtt{T}$, which is defined for all sets X, $x, y \in X$ and $S, T \in \mathcal{P}_\omega(L \times X)$ as

$$\begin{aligned} (S, x), (T, y) \mapsto & \{(l, x'|y) \mid (l, x') \in S\} \\ & \cup \{(l, x|y') \mid (l, y') \in T\} \\ & \cup \{(\tau, x'|y') \mid \exists a, (a, x') \in S \wedge (\overline{a}, y') \in T\} \\ & \cup \{(\tau, x'|y') \mid \exists a, (\overline{a}, x') \in S \wedge (a, y') \in T\}. \end{aligned}$$

Intuitively, S and T are the sets of transitions of the states x and y. The first set $\{(l, x'|y) \mid (l, x') \in S\}$ corresponds to the first GSOS rule

$$\frac{x \xrightarrow{l} x'}{x|y \xrightarrow{l} x'|y}$$

and similarly for the others.

By virtue of Lemma 1, to prove compatibility of Ctx^ℓ, we only have to show that for all predicates $P \subseteq X$, the restriction of λ_X to $\overline{T}(\overline{F}^{\langle\tau\rangle} \times \mathrm{Id})P$ corestricts to $\overline{F}^{\langle\tau\rangle}\overline{\mathtt{T}}P$, that is whenever $(S, x), (T, y) \in \overline{T}(\overline{F}^{\langle\tau\rangle} \times \mathrm{Id})P$, then $\lambda_X((S, x), (T, y)) \in \overline{F}^{\langle\tau\rangle}\overline{\mathtt{T}}P$.

The latter means, by definition of $\overline{F}^{\langle\tau\rangle}$, that there exists a $(\tau, t) \in \lambda_X((S, x), (T, y))$ such that $t \in \overline{\mathtt{T}}P$. This can be proved as follows: since $S \in \overline{F}^{\langle\tau\rangle}P$, then there exists $(\tau, x') \in S$ such that $x' \in P$. By definition of λ_X, $(\tau, x'|y) \in \lambda_X((S, x), (T, y))$. Finally, since $x' \in P$, then $x'|y \in \overline{\mathtt{T}}P$.

D. Details on Nominal Automata

In this section we assume the reader has some familiarity with nominal sets, see [23].

D.1 The base category

We denote by $\mathtt{A}$ a countable set of names. The category Nom of nominal sets has as objects sets X equipped with an action $\cdot : Sym(\mathtt{A}) \times X \to X$ of the group of finitely supported permutations on $\mathtt{A}$ (that is, permutations generated by transpositions of the form $(a\ b)$) and such that each $x \in X$ has a finite support. Morphisms

in Nom are *equivariant* functions, i.e., functions that preserve the group action.

D.2 The fibration at issue

It is well known that Nom can equivalently be described as a Grothendieck topos. Since Nom is a regular category, by [16, Observation 4.4.1] we know that the subobject fibration on Nom is in fact a bifibration. Furthermore, by a change-of-base situation described below we obtain the bifibration $\mathsf{Rel}(\mathsf{Nom}) \to \mathsf{Nom}$, see also [16, Example 9.2.5(ii)]

$$\begin{array}{ccc} \mathsf{Rel}(\mathsf{Nom}) & \longrightarrow & \mathsf{Sub}(\mathsf{Nom}) \\ \downarrow & & \downarrow \\ \mathsf{Nom} & \xrightarrow{I \mapsto I \times I} & \mathsf{Nom} \end{array}$$

Objects of $\mathsf{Rel}(\mathsf{Nom})$ are equivariant relations. That is, if X is a nominal set, a nominal relation on X is just a subset $R \subseteq X^2$ such that xRy implies $(\pi \cdot x)R(\pi \cdot y)$ for all permutations π. This bifibration is also split and bicartesian.

D.3 The functors and the distributive law

We will use the following Nom-endofunctors:

1. $F : \mathsf{Nom} \to \mathsf{Nom}$ given by $FX = 2 \times X^{\mathbb{A}}$, where $2 = \{0, 1\}$ is equipped with the trivial action and $X^{\mathbb{A}}$ is given by the internal hom. Concretely, an element $f \in X^{\mathbb{A}}$ is a function $f : \mathbb{A} \to X$ such that there exists a finite subset $S \subseteq \mathbb{A}$ and $f(\pi(a)) = \pi \cdot f(a)$ for all names $a \in \mathbb{A}$ and permutations $\pi \in Sym(\mathbb{A})$ fixing the elements of S.
2. $\mathcal{P}_\omega : \mathsf{Nom} \to \mathsf{Nom}$ that maps a nominal set X to its orbit-finite finitely supported subsets. In particular one can check that $\mathcal{P}_\omega$ is a monad and let μ denote its multiplication, given by union.

The functors $\mathcal{P}_\omega$ and F are related by a distributive law

$$\lambda : \mathcal{P}_\omega F \Rightarrow F\mathcal{P}_\omega.$$

For a nominal set X, the map λ_X is given by the product of the morphisms acting on $S \in \mathcal{P}_\omega F(X)$ by

$$S \mapsto 1 \in 2 \text{ iff } 1 \in (\mathcal{P}_\omega \tau_1)(S)$$

and

$$S \mapsto \lambda a.\{x \in X | \exists f \in (\mathcal{P}_\omega \tau_2)(S).\ f(a) = x\} \in (\mathcal{P}_\omega X)^{\mathbb{A}}$$

where τ_1, τ_2 are the projections from FX to 2, respectively $X^{\mathbb{A}}$.

D.4 The liftings

The distributive law λ can be lifted to $\mathsf{Rel}(\mathsf{Nom})$, see [17, Exercise 4.4.6].

$$\mathsf{Rel}(\lambda) : \mathsf{Rel}(\mathcal{P}_\omega)\mathsf{Rel}(F) \Rightarrow \mathsf{Rel}(F)\mathsf{Rel}(\mathcal{P}_\omega).$$

Concretely, for $R \in \mathsf{Rel}(\mathsf{Nom})_X$, the nominal relation $\mathsf{Rel}(F)(R)$ is given by $(o, f)\ \mathsf{Rel}(F)(R)\ (o', f')$ iff $o = o'$ and for all $a \in \mathbb{A}$ we have $f(a)Rf'(a)$.

On the other hand $\mathsf{Rel}(\mathcal{P}_\omega)$ is given by $S\ \mathsf{Rel}(\mathcal{P}_\omega)(R)\ S'$ iff for all $x \in S$ exists $y \in S'$ with xRy and for all $y \in S'$ exists $x \in S$ with xRy. As for $\mathsf{Rel}(\lambda)_R$, this is obtained as the restriction of $\lambda_R \times \lambda_R$ to $\mathsf{Rel}(\mathcal{P}_\omega)\mathsf{Rel}(F)(R)$.

D.5 Soundness of bisimulation up to congruence

Nondeterministic nominal automata [4] can be modelled as $F\mathcal{P}_\omega$-coalgebras, while deterministic nominal automata are represented as F-coalgebras. The classical notion of finiteness is replaced by orbit-finiteness—from a categorical perspective this makes sense, since orbit-finite nominal sets are exactly the finitely presentable objects in the lfp category Nom.

The generalised powerset construction [28] can be applied in this situation as well, that is, a nondeterministic nominal automata modelled as a coalgebra

$$\langle o, t \rangle : X \to 2 \times \mathcal{P}_\omega(X)^A$$

yields an F-coalgebra structure

$$\langle o^\sharp, t^\sharp \rangle : \mathcal{P}_\omega X \to 2 \times (\mathcal{P}_\omega X)^A,$$

on $\mathcal{P}_\omega X$, given by the composite $F(\mu) \circ \lambda \circ \mathcal{P}_\omega(\langle o, t \rangle)$. The reason why determinisation fails in a nominal setting [4] is that the finitary power object functor $\mathcal{P}_\omega$ does not preserve orbit finiteness. This is the case in the example of Section 5.3.

Notice that $(\mathcal{P}_\omega X, \mu, \langle o^\sharp, t^\sharp \rangle)$ is a λ-bialgebra.

The fibrations $\mathsf{Rel}(\mathsf{Nom}) \to \mathsf{Nom}$ and $\mathsf{Sub}(\mathsf{Nom}) \to \mathsf{Nom}$ are well-founded in the sense of [13]. To prove this we can apply [13, Lemma 3.4], which gives as a sufficient condition for well-foundedness: that the fibre above each finitely presentable object be finite. Indeed, recall from [32] that finitely presentable nominal sets are the orbit-finite ones. Then, it is easy to check that a nominal set with n orbits has 2^n *equivariant* nominal subsets.

Hence, by [Theorem 3.7][13], the final $\mathsf{Rel}(F)_{\langle o,t \rangle}$-coalgebra exists and can be computed as the limit of an ω^{op}-chain in the fibre $\mathsf{Rel}(\mathsf{Nom})_X$. We will use this coinductive predicate to prove that two states of a nominal automata accept the same language.

We can apply Theorem 2 to prove that the contextual closure $Ctx = \coprod_\mu \circ \mathsf{Rel}(\mathcal{P}_\omega)$ is $\mathsf{Rel}(F)_{\langle o^\sharp, t^\sharp \rangle}$-compatible.

Thus bisimulation up to context is a valid proof technique for nominal automata.

Moreover, we can apply Proposition 2 to prove compatibility of the up to reflexive, symmetric and transitive closure techniques, respectively.

(n=0) Let $Rfl\colon \mathsf{Nom} \to \mathsf{Rel}(\mathsf{Nom})$ be the functor mapping each nominal set X to Δ_X, the identity relation on X. Then Rfl_X is $\mathsf{Rel}(F)_{\langle o,t \rangle}$-compatible since $\Delta_{FX} = \mathsf{Rel}(F)\Delta_X$.

(n=1) Let $Sym\colon \mathsf{Rel}(\mathsf{Nom}) \to \mathsf{Rel}(\mathsf{Nom})$ be the functor mapping each nominal relation $R \subseteq X^2$ to its converse $R^{-1} \subseteq X^2$. Sym_X is $\overline{F}_{\langle o,t \rangle}$-compatible since $\overline{F}(R)^{-1} \subseteq \overline{F}(R^{-1})$ for all relations $R \subseteq X^2$.

(n=2) Let $\otimes\colon \mathsf{Rel}(\mathsf{Nom}) \times_{\mathsf{Nom}} \mathsf{Rel}(\mathsf{Nom}) \to \mathsf{Rel}(\mathsf{Nom})$ be the nominal relational composition functor. Composition of nominal relations is computed just as in Set and one can show that $\mathsf{Rel}(F)$ preserves it. Thus $\otimes$ is $\mathsf{Rel}(F)_{\langle o,t \rangle}$-compatible.

Employing Proposition 1 and the fact that congruence closure is obtained as the composition of the equivalence, context and reflexive closure functors we derive that bisimulation up to congruence is a sound technique.

D.6 The concrete example

The nondeterministic nominal automaton of Section 5.3 (reported on the left below) is given formally by an $F\mathcal{P}_\omega$-coalgebra $\langle o, t \rangle$ on the nominal set $1 + 1 + \mathbb{A} + \mathbb{A} + 1$. For simplicity we denote the

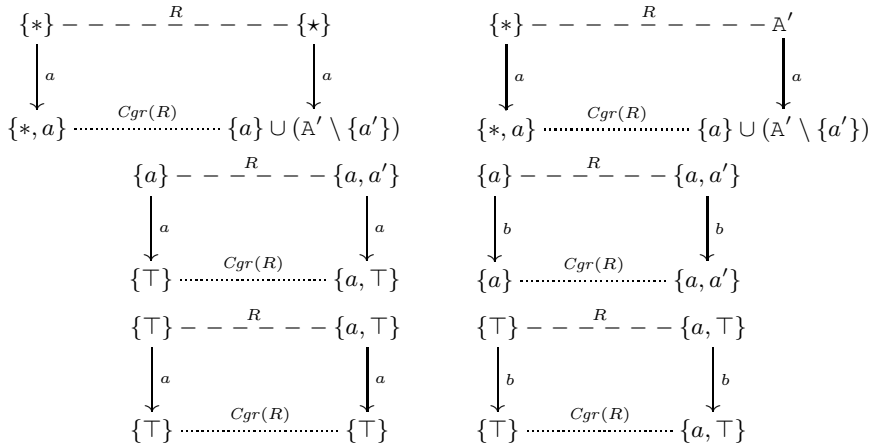
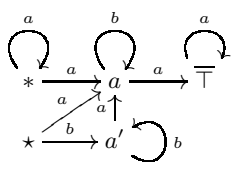


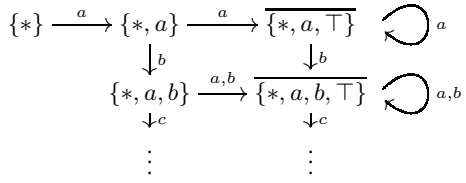
Figure 2. Proving R to be a bisimulation up to congruence

second copy of $\mathbb{A}$ by $\mathbb{A}'$. The map $\langle o, t\rangle$ is given below on the right.



$$\begin{aligned}
* &\mapsto (0, a \mapsto \{*, a\}) \\
a &\mapsto \left(0, \begin{cases} b \mapsto \{a\} & b \# a \\ a \mapsto \{\top\} \end{cases}\right) \\
\star &\mapsto \left(0, a \mapsto \{a\} \cup \mathbb{A}' \setminus \{a'\}\right) \\
a' &\mapsto \left(0, \begin{cases} b \mapsto \{a'\} & b \# a \\ a \mapsto \{a\} \end{cases}\right) \\
\top &\mapsto (1, a \mapsto \{\top\})
\end{aligned}$$

The determinisation of this automaton has infinitely many orbits. For example, the determinisation of the part reachable from $*$ is partially represented by



However, we can prove that $*$ and $\star$ accept the same language, showing that the nominal relation R spanned by

$$(\{*\}, \{\star\}),\ (\{a\}, \{a, a'\}),\ (\{\top\}, \{a, \top\}),\ (\{*\}, \mathbb{A}')$$

is a bisimulation up to congruence, that is, $R \subseteq \mathsf{Rel}(F)_{\langle o^\sharp, t^\sharp\rangle}\, Cgr(R)$.

This is shown in Figure 2: for each pair in R, we check that the successors are in $Cgr(R)$. Note that for the pairs $(\{a\}, \{a, a'\})$ and $(\{\top\}, \{a, \top\})$, in the second and third rows, one needs to check the successors for a and for a fresh name b. Instead for the pairs $(\{*\}, \{\star\})$ and $(\{*\}, \mathbb{A}')$ in the first row, only successors for a should be checked (since a does not belong to the support of these states).

The only non-trivial computation is to check whether $\{*, a\}\, Cgr(R)\, \{a\} \cup (\mathbb{A}' \setminus \{a'\})$. We proceed as follows:

$$\begin{array}{lll} \{*, a\} & Cgr(R) & \{a\} \cup \mathbb{A}' \\ & Cgr(R) & \{a, a'\} \cup (\mathbb{A}' \setminus \{a'\}) \\ & Cgr(R) & \{a\} \cup (\mathbb{A}' \setminus \{a'\}). \end{array}$$

E. Proofs for Section 6

Theorem 3. *Let $\overline{T}$ be a lifting of T having a $\gamma\colon \overline{T}\otimes \Rightarrow \otimes\overline{T}^2$ above $\mathrm{Id}\colon T \Rightarrow T$. Let both $\overline{F_1}$ and $\overline{F_2}$ be liftings of F. If $\lambda_1\colon \overline{T}\,\overline{F_1} \Rightarrow \overline{F_1}\,\overline{T}$ and $\lambda_2\colon \overline{T}\,\overline{F_2} \Rightarrow \overline{F_2}\,\overline{T}$ sit above the same $\lambda\colon TF \Rightarrow FT$, then there exists $\overline{\lambda}\colon \overline{T}(\overline{F_1} \otimes \overline{F_2}) \Rightarrow (\overline{F_1} \otimes \overline{F_2})\overline{T}$ above λ.*

Proof. Since $\overline{F_1}$ and $\overline{F_2}$ are liftings of $F : \mathcal{B} \to \mathcal{B}$ it follows that $\langle \overline{F_1}, \overline{F_2}\rangle : \mathcal{E} \to \mathcal{E} \times_{\mathcal{B}} \mathcal{E}$ is a lifting of F. Moreover $\langle \lambda_1, \lambda_2\rangle : \overline{T}^2\langle \overline{F_1}, \overline{F_2}\rangle \Rightarrow \langle \overline{F_1}, \overline{F_2}\rangle\overline{T}$ is a lifting of λ.

Using that $\otimes : \mathcal{E} \times_{\mathcal{B}} \mathcal{E} \to \mathcal{E}$ lifts the identity we get that $F_1 \otimes F_2 = \otimes \circ \langle F_1, F_2\rangle$ is also a lifting of F.

$$\overline{T} \otimes \langle F_1, F_2\rangle \xRightarrow{\gamma\langle F_1, F_2\rangle} \otimes\overline{T}^2\langle F_1, F_2\rangle \xRightarrow{\otimes\langle \lambda_1, \lambda_2\rangle} \otimes\langle F_1, F_2\rangle\overline{T} \quad (10)$$

$$TF \xRightarrow{\mathrm{id}} TF \xRightarrow{\lambda} FT$$

The required $\overline{\lambda}$ is obtained as the composite $\otimes\langle \lambda_1, \lambda_2\rangle \circ \gamma\langle F_1, F_2\rangle$ sitting above λ as in (10). □

E.1 Proofs for Similarity

Proposition 4. *Let λ be a monotone abstract GSOS specification and $(X, \alpha, \langle \xi, id\rangle)$ be a $\lambda^\dagger$-bialgebra. Then Ctx is $(\mathsf{Rel}(F)^{\sqsubseteq} \times \mathrm{Id})_{\langle \xi, id\rangle}$-compatible.*

Proof. Recall that Ctx is defined as $\coprod_\alpha \circ \mathsf{Rel}(\mathtt{T})$ and that, for the canonical lifting, it holds that $\mathsf{Rel}(\mathtt{T})\otimes \subseteq \otimes\mathsf{Rel}(\mathtt{T})^2$. We decompose the lifting $\mathsf{Rel}(F)^{\sqsubseteq} \times \mathrm{Id}$ as

$$(\overline{\sqsubseteq} \times \overline{\mathrm{Id}}) \otimes (\mathsf{Rel}(F) \times \mathrm{Id}) \otimes (\overline{\sqsubseteq} \times \overline{\mathrm{Id}})$$

where $\overline{\mathrm{Id}}$ is the constant functor mapping $R \subseteq X^2$ to Δ_X. By Theorem 3 we reduce the proof of the fact that $\mathsf{Rel}(T)$ distributes over $\mathsf{Rel}(F)^{\sqsubseteq} \times \mathrm{Id}$ to the fact that $\mathsf{Rel}(T)$ distributes over $\overline{\sqsubseteq} \times \overline{\mathrm{Id}}$ and $\mathsf{Rel}(F) \times \mathrm{Id}$ separately.

For the latter, observe that $\mathsf{Rel}(F) \times \mathrm{Id} = \mathsf{Rel}(F \times \mathrm{Id})$. Since $\mathsf{Rel}(-)$ is a 2-functor [17, Exercise 4.4.6], we take $\overline{\lambda_1^\dagger}\colon \mathsf{Rel}(\mathtt{T})\mathsf{Rel}(F \times \mathrm{Id}) \Rightarrow \mathsf{Rel}(F \times \mathrm{Id})\mathsf{Rel}(T)$ as $\mathsf{Rel}(\lambda)$.

For the former we need to use Lemma 1 and exhibit a $\overline{\lambda}\colon \mathsf{Rel}(T)(\overline{\sqsubseteq} \times \overline{\mathrm{Id}}) \Rightarrow \overline{\sqsubseteq}\mathsf{Rel}(\mathtt{T})$ sitting above λ. This amounts to show that, for all relations $R \subseteq X^2$, the restriction of $\lambda_X \times \lambda_X$ to $\mathsf{Rel}(T)(\overline{\sqsubseteq} \times \overline{\mathrm{Id}})R$ corestricts to $\overline{\sqsubseteq}\mathsf{Rel}(\mathtt{T})R$. Note that since $\overline{\sqsubseteq}$ and $\overline{\mathrm{Id}}$ are constant, this is exactly the condition for monotone abstract GSOS. This guarantees the existence of $\overline{\lambda_2^\dagger}\colon \mathsf{Rel}(\mathtt{T})(\overline{\sqsubseteq} \times \overline{\mathrm{Id}}) \Rightarrow (\overline{\sqsubseteq} \times \overline{\mathrm{Id}})\mathsf{Rel}(\mathtt{T})$ sitting above $\lambda^\dagger$.

The existence of $\overline{\lambda_1^\dagger}$ and $\overline{\lambda_2^\dagger}$ ensures, via Theorems 3 and 2, that Ctx is $(\mathsf{Rel}(F)^{\sqsubseteq} \times \mathrm{Id})_{\langle \xi, id\rangle}$-compatible. □

E.2 Proofs for Weak Bisimilarity

Lemma 8. *$(\overline{F \times F}, F)$ is a fibration map.*

Proof. Let $f : X \to Y$ be a function and $R \subseteq X^2$ be a relation. Then

$$\begin{aligned}
&\overline{F \times F}((f \times f^{-1}(R)) \\
&= \{(S,U,V,W) \mid \\
&\qquad \forall (a,x) \in S.\ \exists (a,y) \in W.\ f(x) R f(y), \\
&\qquad \forall (a,y) \in V.\ \exists (a,x) \in U.\ f(x) R f(y)\} \\
&= \{(S,U,V,W) \mid \\
&\qquad \forall (a,x') \in Ff[S].\ \exists (a,y') \in Ff[W].\ x' R y', \\
&\qquad \forall (a,y') \in Ff[V].\ \exists (a,x') \in Ff[U].\ x' R y'\} \\
&= (Ff \times Ff \times Ff \times Ff)^{-1}(\overline{F \times F}(R))
\end{aligned}$$

□

Proposition 5. *Let $\lambda \colon T(F \times \mathrm{Id}) \Rightarrow F\mathtt{T}$ be a positive GSOS specification and $(X, \alpha, \langle \xi_1, id\rangle)$ and $(X, \alpha, \langle \xi_2, id\rangle)$ be two $\lambda^\dagger$-bialgebras then Ctx is $(\overline{F \times F} \times \mathrm{Id})_{\langle \xi_1, \xi_2, id\rangle}$-compatible.*

Proof. From $\lambda \colon T(F \times \mathrm{Id}) \Rightarrow F\mathtt{T}$, we define $\tilde{\lambda} \colon T(F \times F \times \mathrm{Id}) \Rightarrow (F \times F)\mathtt{T}$ as $\langle \lambda \circ T(\tau_1 \times \tau_3), \lambda \circ T(\tau_2 \times \tau_3)\rangle$ where τ_i are the projections from $F \times F \times \mathrm{Id}$ to F and Id. Such $\tilde{\lambda}$ induces a distributive law

$$\tilde{\lambda}^\dagger \colon \mathtt{T}(F \times F \times \mathrm{Id}) \Rightarrow (F \times F \times \mathrm{Id})\mathtt{T}.$$

From the $\lambda^\dagger$-bialgebras $(X, \alpha, \langle \xi_1, id\rangle)$ and $(X, \alpha, \langle \xi_2, id\rangle)$, we construct $(X, \alpha, \langle \xi_1, \xi_2, id\rangle)$ which is a $\tilde{\lambda}^\dagger$-bialgebra.

Recall that Ctx is defined as $\coprod_\alpha \circ \mathsf{Rel}(\mathtt{T})$ and that, for the canonical lifting, it holds that $\mathsf{Rel}(\mathtt{T})\otimes \subseteq \otimes \mathsf{Rel}(\mathtt{T})^2$. We decompose the lifting $\overline{F \times F} \times \mathrm{Id}$ as

$$(\rho \times \overline{\mathrm{Id}}) \otimes (\mathsf{Rel}(F \times F)^{[\supseteq \subseteq]} \times \mathrm{Id})$$

where $\overline{\mathrm{Id}}$ is the constant functor mapping $R \subseteq X^2$ to Δ_X. By Theorem 3 we reduce the proof of the fact that $\mathsf{Rel}(T)$ distributes over $\overline{F \times F} \times \mathrm{Id}$ to the fact that $\mathsf{Rel}(T)$ distributes over $\rho \times \overline{\mathrm{Id}}$ and $\mathsf{Rel}(F \times F)^{[\supseteq \subseteq]} \times \mathrm{Id}$ separately.

For the former, by Lemma 1, we have to prove that for all relations $R \subseteq X^2$, the restriction of $\tilde{\lambda}_X \times \tilde{\lambda}_X$ to $\mathsf{Rel}(T)(\rho \times \overline{\mathrm{Id}})R$ corestricts to $\rho \mathsf{Rel}(\mathtt{T})R$. This can be easily checked by using the fact that both ρ and $\overline{\mathrm{Id}}$ are constant and exploiting the definition of $\tilde{\lambda}$. As a consequence there exists a $\overline{\tilde{\lambda}_1}^\dagger : \mathsf{Rel}(\mathtt{T})(\rho \times \overline{\mathrm{Id}}) \Rightarrow (\rho \times \overline{\mathrm{Id}})\mathsf{Rel}(\mathtt{T})$ sitting above $\tilde{\lambda}^\dagger$.

For $\mathsf{Rel}(F \times F)^{[\supseteq \subseteq]} \times \mathrm{Id}$ we can reuse Proposition 4, but first we have to prove that the GSOS specification $\tilde{\lambda}$ is monotone w.r.t. $[\supseteq \subseteq]$. Via simple computations, one can check that this is indeed the case when the original GSOS specification λ is positive. As a consequence there exists a $\overline{\tilde{\lambda}_2}^\dagger : \mathsf{Rel}(\mathtt{T})(\mathsf{Rel}(F \times F)^{[\supseteq \subseteq]} \times \mathrm{Id}) \Rightarrow (\mathsf{Rel}(F \times F)^{[\supseteq \subseteq]} \times \mathrm{Id})\mathsf{Rel}(\mathtt{T})$ sitting above $\tilde{\lambda}^\dagger$.

The existence of $\overline{\tilde{\lambda}_1}^\dagger$ and $\overline{\tilde{\lambda}_2}^\dagger$ entails, via Theorems 3 and 2 compatibility of Ctx for $(\overline{F \times F} \times \mathrm{Id})_{\langle \xi_1, \xi_2, id\rangle}$. □

Additional references for the appendix

[32] D. Petrişan. *Investigations into Algebra and Topology over Nominal Sets*. PhD thesis, University of Leicester, 2012.

[33] R. Street. Fibrations and Yoneda's lemma in a 2-category. In GregoryM. Kelly, editor, *Category Seminar*, volume 420 of *Lecture Notes in Mathematics*, pages 104–133. Springer Berlin Heidelberg, 1974.

Full abstraction for fair testing in CCS (expanded version)

Tom Hirschowitz

FULL ABSTRACTION FOR FAIR TESTING IN CCS

(EXPANDED VERSION)

TOM HIRSCHOWITZ

CNRS, Université de Savoie

Abstract. In previous work with Pous, we defined a semantics for CCS which may both be viewed as an innocent form of presheaf semantics and as a concurrent form of game semantics. We define in this setting an analogue of fair testing equivalence, which we prove fully abstract w.r.t. standard fair testing equivalence.

The proof relies on a new algebraic notion called *playground*, which represents the 'rule of the game'. From any playground, we derive two languages equipped with labelled transition systems, as well as a strong, functional bisimulation between them.

Key words and phrases: Programming languages; categorical semantics; presheaf semantics; game semantics; concurrency; process algebra.

An extended abstract of this paper has appeared in CALCO '13.

Partially funded by the French ANR projets blancs PiCoq ANR-10-BLAN-0305 and Récré ANR-11-BS02-0010.

LOGICAL METHODS
IN COMPUTER SCIENCE

DOI:10.2168/LMCS-???

Contents

1. Introduction

Games	Concurrency
position	configuration
player	agent
move	action
play	trace

This paper is about *game semantics* for CCS [43]. Game semantics is originally a very successful approach to *sequential* denotational semantics [45, 26, 2]. Its basic idea is to interpret programs as strategies for a player in a game, and the computational environment as an opponent. Composition of programs is handled by letting the corresponding strategies interact. We mostly use game semantical terminology in this paper, but the above dictionary may help the intuition of concurrency theorists.

Denotational models of CCS are extremely diverse, and treat various behavioural equivalences, as surveyed by Winskel and Nielsen [54]. The closest game semantical work seems to be Laird's model [33], which achieves full abstraction w.r.t. *trace* (a.k.a. *may testing*) equivalence for a fragment of π. The goal of the present paper is to design the first game semantics for a finer equivalence than trace equivalence, in the simpler setting of CCS (we plan to address the full π-calculus in future work). The reason Laird is limited to trace equivalence is that the standard notion of strategy is a set of plays (with well-formedness conditions). Hence, e.g., the famous coffee machines, $a.b + a.c$ and $a.(b + c)$, are identified. Following two recent, yet independent lines of work [49, 24], we generalise strategies by allowing them to accept plays *in several ways*, thus reconciling game semantics with presheaf models [30]. Winskel et al.'s approach is only starting to be applied to concrete languages, see for example the work in progress on an affine, concurrent variant of Idealised Algol [8]. The approach of [24, 25] (HP) was used to give a game semantics for CCS, and define a semantic analogue of fair testing equivalence, but no adequacy result was proved. We here prove full abstraction of semantic fair testing equivalence w.r.t. standard fair testing equivalence. Our model is compositional, since (1) all syntactic constructs of CCS have natural interpretations, and (2) global dynamics may be inferred from local dynamics, as in any game semantics (see the paragraph on innocence below and Sections 3.4.2 and 3.4.3).

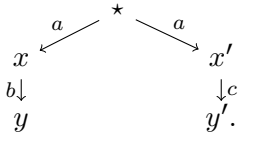
1.1. **Overview of the approach.**

Truly concurrent plays. First of all, as in [49], our notion of play is truly concurrent. Indeed, it does not keep track of the order in which (atomic) moves occur. Instead, it only retains causal dependencies between them (see Section 3.3). Furthermore, our plays form a proper category, which enables in particular a smooth treatment of bound variables. Briefly, plays that differ only up to a permutation of channels are isomorphic, and by construction strategies handle them correctly.

Branching behaviour. Second, we deal with branching behaviour. Standardly, and ignoring momentarily the previous paragraph, a strategy is essentially a prefix-closed set of 'accepted' plays. This is equivalent to functors $\mathbb{E}^{op} \to 2$, where $\mathbb{E}$ is the poset of plays ordered by prefix inclusion, and 2 is the poset $0 \leq 1$ ($\mathbb{E}$ stands for 'extension'). A play u is 'accepted' by such a functor F when $F(u) = 1$, and if $u' \leq u$, then functoriality imposes that $F(u) \leq F(u')$, hence $F(u') = 1$: this is prefix-closedness. In order to allow plays to be accepted in several ways, we follow *presheaf models* [30] and move to functors $\mathbb{E}^{op} \to \mathsf{set}$, where set is the category

of finite ordinals and all functions between them[1]. Thus, to each play $u \in \mathbb{E}$, a strategy associates a *set* of ways to accept it, empty if u is rejected. E.g., in the simplistic setting where $\mathbb{E}$ denotes the poset of words over actions, ordered by prefix inclusion, the coffee machine $a.b + a.c$ is encoded as the presheaf S defined on the left and pictured on the right:

- $S(\epsilon) = \{\star\}$,
- $S(a) = \{x, x'\}$,
- $S(ab) = \{y\}$,
- $S(ac) = \{y'\}$,
- S empty otherwise,
- $S(\epsilon \hookrightarrow a) = \{x \mapsto \star, x' \mapsto \star\}$,
- $S(a \hookrightarrow ab) = \{y \mapsto x\}$,
- $S(a \hookrightarrow ac) = \{y' \mapsto x'\}$,



This illustrates what is meant by 'accepting a play in several ways': the play a is here accepted in two ways, x and x'. The other coffee machine is of course obtained by identifying x and x'. In our setting, plays are considered relative to their initial position X, hence strategies are presheaves $\mathbb{E}_X^{op} \to \mathsf{set}$ on the category of plays over X.

Innocence. Finally, defining strategies as presheaves on plays is too naive, which leads us to reincorporate the game semantical idea of *innocence*. Example 3.14 below exhibits such a presheaf in which two players synchronise on a public channel a, without letting others interfere. In CCS, this would amount to a process like $\overline{a}.P \,|\, a.Q \,|\, a.R$ in which, say, the first two processes could arrange for ruling out the third. Considering such presheaves as valid strategies would break our main result.

In the Hyland-Ong approach, innocent strategies may be defined as prefix-closed sets of *views*, where views are special plays representing the information that a player may 'access' during a global play. The global strategy $\overline{S}$ associated to an innocent strategy S is then recovered by decreeing that $\overline{S}$ accepts all plays whose views are accepted by S. This leads us to consider a subcategory $\mathbb{E}^{\mathbb{V}}$ of the category $\mathbb{E}$ of plays, whose objects are called *views*. We thus have for each position X two categories of strategies: the naive one, the category $[\mathbb{E}_X^{op}, \mathsf{set}]$ of *behaviours* on X, consists of presheaves on plays; the more relevant one, the category $[(\mathbb{E}_X^{\mathbb{V}})^{op}, \mathsf{set}]$ of *strategies* on X, consists of presheaves on views.

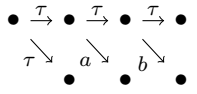
How, then, do we recover the global behaviour associated to a strategy, which is crucial for defining our semantic fair testing equivalence? The right answer is given by a standard categorical construction called right Kan extension (see Section 3.4.2). Roughly, for the behaviour B_S associated to a strategy S, a way to accept some play $u \in \mathbb{E}_X$ is a compatible family of ways for S to accept all views of u. In the boolean, setting (considering functors $\mathbb{E}_X^{op} \to 2$), this reduces to B_S accepting u iff all its views are accepted by S. Our definition thus generalises Hyland and Ong's.

Finally, game semantical *parallel composition* (different from CCS parallel composition, though inspired from it) intuitively lets strategies interact together. We account for it as follows. If we partition the players of a play X into two teams, we obtain two subpositions $X_1 \hookrightarrow X \hookleftarrow X_2$, each player of X belonging to X_1 or X_2 according to its team. We have that the category $\mathbb{E}_X^{\mathbb{V}}$ of views on X is isomorphic to the coproduct category $\mathbb{E}_{X_1}^{\mathbb{V}} + \mathbb{E}_{X_2}^{\mathbb{V}}$. The parallel composition of any two strategies S_1 and S_2 on X_1 resp. X_2 is simply obtained by universal property of coproduct, as above right.

$$\begin{array}{ccccc} (\mathbb{E}_{X_1}^{\mathbb{V}})^{op} & \hookrightarrow & (\mathbb{E}_X^{\mathbb{V}})^{op} & \hookleftarrow & (\mathbb{E}_{X_2}^{\mathbb{V}})^{op} \\ & {\scriptstyle S_1}\searrow & \downarrow{\scriptstyle [S_1,S_2]} & \swarrow{\scriptstyle S_2} & \\ & & \mathsf{set} & & \end{array}$$

[1]The author learnt this point of view from a talk by Sam Staton.

1.2. **Main result: which behavioural equivalence?** With our game in place, we easily define a translation of CCS processes into strategies. It then remains to demonstrate the adequacy of this translation. Our strategies are actually rather intensional, so we cannot hope for adequacy w.r.t. equality of strategies. Instead, we exploit the rich structure of our model to define both an LTS and an analogue of fair testing equivalence *on the semantic side*, i.e., for strategies. We then provide two results. The most important, in the author's view, is full abstraction w.r.t. standard *fair testing semantics* (Corollary 6.51). But the second result might be considered more convincing by many: it establishes that our semantics is fully abstract w.r.t. weak bisimilarity (Corollary 6.50). A reason why the latter result is here considered less important originates in the tension between LTS semantics and reduction semantics [47]. Briefly, reduction semantics is simple and intuitive, but it operates on equivalence classes of terms (under so-called *structural* congruence). On the other hand, designing LTSs is a subtle task, rewarded by easier, more structural reasoning over reductions. We perceive LTS semantics as less intrinsic than reduction semantics. E.g., for more sophisticated calculi than CCS, several LTSs exist, which yield significantly different notions of bisimilarity.



Beyond LTS-based equivalences, we see essentially two options: *barbed congruence* [52] or some *testing equivalence* [9]. Barbed congruence equates processes P and Q, roughly, when for all contexts C, $C[P]$ and $C[Q]$ are weakly bisimilar w.r.t. *reduction* (i.e., only τ-actions are allowed), and furthermore they have the same interaction capabilities at all stages. Barbed congruence is sometimes perceived as too discriminating w.r.t. guarded choice. Consider, e.g., the CCS process P_1 pictured above, and let P_2 be the same with a and b swapped. Both processes may disable both actions a and b, the only difference being that P_1 disables a *before* disabling b. Barbed congruence distinguishes P_1 from P_2 (take $C = \Box \,|\, \overline{a}$), which some view as a deficiency.

Another possibility would be *must testing* equivalence [9]. Recall that P *must pass* a test process R iff all maximal executions of $P \,|\, R$ perform, at some point, a fixed 'tick' action [19], here denoted by $\heartsuit$. Then, P and Q are must testing equivalent iff they must pass the same tests. Must testing equivalence is sometimes perceived as too discriminating w.r.t. divergence. E.g., consider $Q_1 = !\tau \,|\, a$ and $Q_2 = a$. Perhaps surprisingly, Q_1 and Q_2 are *not* must testing equivalent. Indeed, Q_2 must pass the test $\overline{a}.\heartsuit$, but Q_1 does not, due to an infinite, silent reduction sequence.

We eventually go for fair testing equivalence, which was originally introduced (for CCS-like calculi) to rectify both the deficiency of barbed congruence w.r.t. choice and that of must testing equivalence w.r.t. divergence. The idea is that two processes are equivalent when they *should* pass the same tests. A process P should pass the test T iff their parallel composition $P \,|\, T$ never loses the ability of performing the special 'tick' action, after any tick-free reduction sequence. Fair testing equivalence thus equates P_1 and P_2 above, as well as Q_1 and Q_2. Cacciagrano et al. [7] provide an excellent survey.

1.3. **Plan and overview.** We now give a bit more detail on the contents. In Section 2, we introduce our notations and some preliminaries. Section 3 summarises from HP the game for CCS, the notions of strategy and behaviour, the translation $(\!|-|\!)$ of CCS processes into strategies, and semantic fair testing equivalence. The rest is devoted to proving that $(\!|-|\!)$, here decomposed as $[\![-]\!] \circ \theta$ (see below), is such that $P \sim_{f,s} Q$ iff $(\!|P|\!) \sim_f (\!|Q|\!)$, where $\sim_{f,s}$ is standard fair testing equivalence (Corollary 6.51).

1.3.1. *Playgrounds.* Our proof of this result takes a long detour to introduce a new algebraic gadget called *playground*, which we now motivate. Our first attempts at proving the full abstraction result were obscured by a tight interleaving of

- results stating common properties of moves in the game, or of plays, and
- results and constructions on strategies derived from those (e.g., the LTS for strategies).

On the other hand, the reasons why our constructions work are intuitively simple. Namely, innocent strategies essentially amount to describing syntax trees by selecting their branches amongst a set of all possible branches. This enlarges the universe of terms slightly, but in game semantics, one studies properties of terms which also make sense for such generalised terms. Compositionality and the definition of our semantic fair testing equivalence are examples where using strategies instead of terms tends to simplify the constructions. E.g., associated behaviours are recovered from innocent strategies through Kan extension, thanks to an expressive notion of morphism between plays. Our results essentially follow from this correspondence between terms and strategies.

Example 1.1. To illustrate what we mean by generalised terms, consider standard, unlabelled binary trees as a stripped down example of a term language. Such trees admit a description as prefix-closed sets of words over $\{0, 1\}$ (their sets of *occurrences*). In order to get exactly trees, such sets should be constrained a bit. E.g., the empty set of words, or the set $\{(), (0)\}$ do not describe any tree.

Playgrounds are a first attempt at a general framework describing this correspondence between terms and strategies. We develop their theory in Sections 4 and 5, whose main result is a strong bisimulation between both presentations (i.e., terms vs. strategies). This is then expoited in the next sections to derive the main results.

The basis for playgrounds are *pseudo double categories* [20, 21, 35, 17], a weakening of Ehresmann's double categories [11, 12]. Playgrounds are thus pseudo double categories with additional structure. The objects of a playground represent positions in the game. There are two kinds of morphisms: *vertical* morphisms represent plays, while *horizontal* ones represent embeddings of positions. E.g., there are special objects representing 'typical' players; and a player of a position X is a horizontal morphism $d \to X$ from such a typical player, in a Yoneda-like way. There are then axioms to model atomicity (plays may be decomposed into atomic moves) and locality (plays over a large position may be restricted to any subposition; each player only sees part of the play). There are finally a few more technical axioms.

In Section 4, we give the definition and derive a few basic results and constructions. In particular, we define a naive notion of strategy, *behaviours*, and a less naive notion, *strategies*. Finally, we relate the two by exhibiting a functor from strategies to behaviours. In Section 5, we prove that strategies are in bijective correspondence with infinite terms in a certain language. We then derive from this an LTS $\mathcal{S}_{\mathbb{D}}$ for strategies. Furthermore, we define a second language, which is closer to usual process calculi. And indeed, instantiating this general language to our game for CCS yields essentially CCS, the only difference being that channel creation is treated on an equal footing with input and output. We further equip this language of *process terms* with an LTS $\mathcal{T}_{\mathbb{D}}$. Finally, we define a translation from process terms to strategies $[\![-]\!]\colon \mathcal{T}_{\mathbb{D}} \to \mathcal{S}_{\mathbb{D}}$, which is proved to be a strong bisimulation (Theorem 5.35).

At this point, it remains

(1) to show that the pseudo double category $\mathbb{D}^{CCS}$ formed by our game does satisfy the axioms for playgrounds, and
(2) to use the strong bisimulation $[\![-]\!]$ to derive our main results.

1.3.2. *Graphs with complementarity.* We start with (2), because we feel doing otherwise would disrupt the flow of the paper. Indeed, it should not be surprising at all that $\mathbb{D}^{CCS}$ forms a playground; and furthermore the methods employed to show this are in sharp contrast with the rest of the paper. The plan for (2), carried out in Section 6, is as follows.

First, we reduce semantic fair testing equivalence to fair testing equivalence in the LTS $\mathcal{S}_{\mathbb{D}^{CCS}}$, thus bridging the gap between the game semantical world and LTSs. But this is not as simple as it looks. Indeed, Hennessy and De Nicola's original setting for testing equivalences [9] is not quite expressive enough for our purposes, which leads us to define a slightly more general one, called *modular graph with complementarity*. First, our setting is 'typed', in the sense that not all tests may be applied to a process P, only tests of a type 'compatible' with P. Furthermore, in modular graphs with complementarity, fair testing equivalence relies on a notion of *complementarity* saying when two transitions may be glued together to form a *closed-world* transition. Thus, fair testing equivalence is 'intrinsic', i.e., does not depend on any alphabet. So we have a mere LTS $\mathcal{S}_{\mathbb{D}^{CCS}}$ over an *ad hoc* alphabet $\mathbb{Q}$ derived from $\mathbb{D}^{CCS}$, and we need promote it into a modular graph with complementarity. This goes by refining the original alphabet $\mathbb{Q}$ with 'interfaces', yielding a new alphabet $\mathbb{IQ}$. We then define a morphism $\chi\colon \mathbb{IQ} \to \mathbb{Q}$, and pull $\mathcal{S}_{\mathbb{D}^{CCS}}$ back along χ, thus obtaining our modular graph with complementarity $\mathcal{S}^{\mathbb{IQ}}_{\mathbb{D}^{CCS}}$ (which is thus also an LTS over $\mathbb{IQ}$). In passing, we do the same for $\mathcal{T}_{\mathbb{D}^{CCS}}$, which yields $\mathcal{T}^{\mathbb{IQ}}_{\mathbb{D}^{CCS}}$: this will be useful later. We finally prove that fair testing equivalence in $\mathcal{S}^{\mathbb{IQ}}_{\mathbb{D}^{CCS}}$ coincides with semantic fair testing equivalence (Lemma 6.26). Similarly, we construct a modular graph with complementarity CCS for CCS, and show that fair testing equivalence therein coincides with standard fair testing equivalence (Proposition 6.21). We are thus reduced to proving that some composite $CCS \xrightarrow{\theta} \mathcal{T}^{\mathbb{IQ}}_{\mathbb{D}^{CCS}} \xrightarrow{[\![-]\!]} \mathcal{S}^{\mathbb{IQ}}_{\mathbb{D}^{CCS}}$ is *fair*, i.e., preserves and reflects fair testing equivalence.

Our second step is to establish a sufficient condition for a relation $R\colon G \nrightarrow H$ to be fair and to apply this to the graph of our translation $CCS \to \mathcal{S}^{\mathbb{IQ}}_{\mathbb{D}^{CCS}}$. The idea is to define what an adequate alphabet A should be in our setting, and to prove that, essentially, if we can find an adequate alphabet A for G and H, such that R is a relation over A, then R is fair as soon as

- R is included in weak bisimilarity over A, and
- both graphs have enough A-*trees*, in a sense inspired by the notion of *failure* [48].

In order to apply this, we transform $\mathcal{S}^{\mathbb{IQ}}_{\mathbb{D}^{CCS}}$ and $\mathcal{T}^{\mathbb{IQ}}_{\mathbb{D}^{CCS}}$ into modular graphs with complementarity over the same alphabet $\mathbb{A}$ (i.e., set of labels) as CCS. We proceed by 'relabeling' along some morphism of graphs $\mathbb{IQ} \xrightarrow{\xi} \mathbb{A}$. We still have our translation $\mathcal{T}^{\mathbb{IQ}}_{\mathbb{D}^{CCS}} \xrightarrow{[\![-]\!]} \mathcal{S}^{\mathbb{IQ}}_{\mathbb{D}^{CCS}}$, which is a strong, functional bisimulation over $\mathbb{A}$. It thus remains to check that (a) the map $CCS \xrightarrow{\theta} \mathcal{T}^{\mathbb{IQ}}_{\mathbb{D}^{CCS}}$ is included in weak bisimilarity, and (b) both CCS and $\mathcal{S}^{\mathbb{IQ}}_{\mathbb{D}^{CCS}}$ have enough $\mathbb{A}$-trees. Roughly, G has enough A-trees when, for any t in a certain class of tree-like LTSs over A called A-trees, there exists $x_t \in G$ weakly bisimilar to t. For (b), all three LTSs under consideration clearly have enough $\mathbb{A}$-trees. For (a), our proof is brute force.

1.3.3. *CCS as a playground.* We finally deal in Section 7 with the last missing bit of our proof: we show that $\mathbb{D}^{CCS}$ forms a playground. This rests upon the following two main ingredients.

First, we design a correctess criterion for plays, in a sense close to correctness criteria in linear logic. Namely, plays from some position X to position Y are represented as particular cospans $Y \xrightarrow{s} U \xleftarrow{t} X$ in some category. Specifically, they are obtained by closing a given set of cospans named *moves* under identities and composition. We design a combinatorial criterion for deciding when an arbitrary cospan is indeed a play.

The second main ingredient is a construction of the *restriction* of a play U from some position X to a subposition $X' \hookrightarrow X$. Briefly, this means computing the part of U which is relevant to players in X'. This construction is almost easy: most of U may be 'projected' back onto the initial position X, and then a mere pullback

$$\begin{array}{ccc} U_{|X'} & \hookrightarrow & U \\ \downarrow & \lrcorner & \downarrow \\ X' & \hookrightarrow & X \end{array}$$

of sets gives the needed restriction. The glitch is that in general some parts of U may not canonically be projected back onto X. The principle for this projection is as simple as: project, e.g., input moves to the inputting player. The problem arises for synchronisations. Projecting them to the channel over which the synchronisation occurs does not yield the desired result, and similarly projecting to either of the involved players fails. Our solution is to ignore synchronisations at first, and later reintroduce them automatically using a technique from algebraic topology: factorisation systems [28].

With both of these ingredients in place, the proof is relatively straightforward.

Section 8 concludes and provides some perspectives for future work.

1.4. **Related work.** Our bisimulation result relating terms to strategies for any playground draws inspiration from *Kleene coalgebra* [6, 5]. There, the main idea is that both the syntax and the semantics of various kinds of automata should be *derived* from more basic data describing, roughly, the 'rule of the game'. Formally, starting from a well-behaved (*polynomial*) endofunctor on sets, one constructs both (1) an equational theory and (2) a sound and complete coalgebraic semantics. This framework has been applied in standard automata theory, as well as in quantitative settings. Nevertheless, its applicability to programming language theory is yet to be established. E.g., the derived languages do not feature parallel composition. Our playgrounds may be seen as a first attempt to convey such ideas to the area of programming language theory. Technically, our framework is rather different though, in that we replace the equational theory by a transition system, and the coalgebraic semantics by a game semantics. To summarise, our approach is close in spirit to Kleene coalgebra, albeit without quantitative aspects. Conversely, Kleene coalgebra resembles our approach without innocence.

Building upon previous work [1, 40, 42] on *asynchronous* games, a series of papers by Winskel and collaborators (see, e.g., Rideau and Winskel [49], Winskel [53]) attempt to define a notion of concurrent strategy encompassing both innocent game semantics and presheaf models. Ongoing work evoked above [8] shows that the model does contain innocent game semantics, but presheaf models are yet to be investigated. (Their notion of innocence, borrowed from Faggian and Piccolo [13], is not intended to be related to that of Hyland

and Ong.) In their framework, a game is an event structure, whose events are thought of as moves, equipped with a notion of polarity. In one of the most recent papers in the series [53], Winskel establishes a strong relationship between his concurrent strategies and presheaves. For a given event structure with polarity A, he considers the so-called *Scott order* on the set $\mathcal{C}(A)$ configurations of A. For two configurations c and d, we have $c \sqsubseteq_A d$ iff d may be obtained from c by removing some negative moves and then adding some positive ones, in a valid way. Strategies are then shown to coincide with presheaves on $(\mathcal{C}(A), \sqsubseteq_A)$. This is close in spirit to our use of presheaves, but let us mention a few differences. First, our games do not directly deal with polarity. Furthermore, in our setting, for any morphism $p \to q$ of plays, q is intuitively bigger than p in some way, unlike what happens with the Scott ordering. Finally, an important point in our use of (pre)sheaves is that, unlike configuration posets, our plays form proper *categories*, i.e., homsets may contain more than one element (intuitively, the same view may have several occurrences in a given play). Thus, potential links between both approaches remain to be further investigated.

To conclude this paragraph, let us mention a few, more remotely related lines of work. Melliès [41], although in a deterministic and linear setting, incorporates some 'concurrency' into plays by presenting them as string diagrams. Our notion of innocent strategy shares with Harmer et al.'s [23] presentation of innocence based on a distributive law the goal of better understanding the original notion of innocence. Finally, others have studied game semantics in non-deterministic [22] or concurrent [18, 33] settings, using coarser, trace-based behavioural equivalences.

2. Prerequisites and preliminaries

In this section, we recall some needed material and introduce our notations. We attempt to provide intuitive, yet concise explanations, but these may not suffice to get the non-specialist reader up to speed, so we also provide references when possible.

For the reader's convenience, we finally provide in Figure 6 (end of paper) a summary of notations, beyond those introduced here.

2.1. Sets, categories, presheaves.

We make intensive use of category theory, of which we assume prior knowledge of categories, functors, natural transformations, limits and colimits, adjoint functors, presheaves, bicategories, Kan extensions, and pseudo double categories. All of this except pseudo double categories is entirely covered in Mac Lane's standard textbook [38] and the beginning of Mac Lane and Moerdijk [39]. For a more leisurely introduction, one may consult Lawvere and Schanuel [34], or Leinster [36]. The needed material on Kan extensions roughly amounts to their expression as ends, which is recalled when used (Section 3.4.2). The last bit, namely the notion of *pseudo double category* is briefly recalled below, after fixing some notation. Finally, there are very local uses of locally presentable categories [3] in the present section, and of adhesive category theory [32] in the proof of Lemma 7.35.

Throughout the paper, any finite ordinal n is seen as $\{1, \dots, n\}$ (rather than $\{0, \dots, n-1\}$). In any category, for any object C and set X, let $X \cdot C$ denote the $|X|$-fold coproduct of C with itself, i.e., $C + \dots + C$, $|X|$ times.

Set is the category of sets; set is a skeleton of the category of finite sets, e.g., the category of finite ordinals and arbitrary maps between them; ford is the category of finite ordinals and monotone maps between them. For any category $\mathbb{C}$, $\widehat{\mathbb{C}} = [\mathbb{C}^{op}, \mathsf{Set}]$ denotes the

category of presheaves on $\mathbb{C}$, while $\overline{\mathbb{C}} = [\mathbb{C}^{op}, \mathsf{set}]$ and $\widehat{\mathbb{C}} = [\mathbb{C}^{op}, \mathsf{ford}]$ respectively denote the categories of presheaves of finite sets and of finite ordinals. One should distinguish, e.g., 'presheaf of finite sets' $\mathbb{C}^{op} \to \mathsf{set}$ from 'finite presheaf of sets' $F\colon \mathbb{C}^{op} \to \mathsf{Set}$. The category $\widehat{\mathbb{C}}^f$ of *finite* presheaves is the full subcategory of $\widehat{\mathbb{C}}$ spanning presheaves F which are finitely presentable [3]. In presheaf categories, finitely presentable objects are the same as finite colimits of representables. In the only case we will use ($\mathbb{C}$ below), because representables have finite categories of elements, the latter in turn coincide with presheaves F such that the disjoint union $\sum_{c \in \mathrm{ob}(\mathbb{C})} F(c)$ is finite. For all presheaves F of any such kind, $x \in F(d)$, and $f\colon c \to d$, let $x \cdot f$ denote $F(f)(x)$.

Remark 2.1. This conflicts with the notation $X \cdot C$ above, but context should disambiguate, as in $X \cdot C$ a set X acts on an object C, whereas in $x \cdot f$, a morphism f acts on an object x.

We denote the Yoneda embedding by $\mathsf{y}\colon \mathbb{C} \to \widehat{\mathbb{C}}$, and often abbreviate $\mathsf{y}(c)$ to just c.

For any functor $F\colon \mathbb{C} \to \mathbb{D}$ and object $D \in \mathbb{D}$, let F_D denote the comma category on the left below, and $F(D)$ denote the pullback category on the right:

$$\begin{array}{ccc} F_D & \longrightarrow & 1 \\ \downarrow & \swarrow & \downarrow{\scriptstyle \ulcorner D \urcorner} \\ \mathbb{C} & \xrightarrow[F]{} & \mathbb{D} \end{array} \qquad\qquad \begin{array}{ccc} F(D) & \longrightarrow & 1 \\ \downarrow & \lrcorner & \downarrow{\scriptstyle \ulcorner D \urcorner} \\ \mathbb{C} & \xrightarrow[F]{} & \mathbb{D}. \end{array} \tag{2.1}$$

When F is clear from context, we simply write $\mathbb{C}_D$, resp. $\mathbb{C}(D)$. Also, as usual, when F is the identity, we use the standard slice notation $\mathbb{D}/D$.

Finally, we briefly recall pseudo double categories. They are a weakening of Ehresmann's double categories [11, 12], notably studied by Grandis and Paré [20, 21], Leinster [35], and Garner [17]. The weakening lies in the fact that one dimension is strict and the other weak (i.e., bicategory-like). We need to consider proper pseudo double categories, notably we use cospans in examples, but we often handle pseudoness a bit sloppily. Indeed, the proofs of Section 4 quickly become unreadable when accounting for pseudoness.

A pseudo double category $\mathbb{D}$ consists of a set $\mathrm{ob}(\mathbb{D})$ of *objects*, shared by a 'horizontal' category $\mathbb{D}_h$ and a 'vertical' bicategory $\mathbb{D}_v$. Following Paré [46], $\mathbb{D}_h$, being a mere category, has standard notation (normal arrows, $\circ$ for composition, id for identities), while the bicategory $\mathbb{D}_v$ earns fancier notation ($\multimap$ arrows, $\bullet$ for composition, $id^\bullet$ for identities). $\mathbb{D}$ is furthermore equipped with a set of *double cells* α, which have vertical, resp. horizontal, domain and codomain, denoted by $\mathrm{dom}_v(\alpha)$, $\mathrm{cod}_v(\alpha)$, $\mathrm{dom}_h(\alpha)$, and $\mathrm{cod}_h(\alpha)$.

We picture this as, e.g., α on the right, where $u = \mathrm{dom}_h(\alpha)$, $u' = \mathrm{cod}_h(\alpha)$, $h = \mathrm{dom}_v(\alpha)$, and $h' = \mathrm{cod}_v(\alpha)$. Finally, there are operations for composing double cells: *horizontal* composition $\circ$ composes them along a common vertical morphism, *vertical* composition $\bullet$ composes along horizontal morphisms. Both vertical compositions (of morphisms and of double cells) may be associative only up to coherent isomorphism. The full axiomatisation is given by Garner [17], and we here only mention the *interchange law*, which says that the two ways of parsing the above diagram coincide: $(\beta' \circ \beta) \bullet (\alpha' \circ \alpha) = (\beta' \bullet \alpha') \circ (\beta \bullet \alpha)$.

$$\begin{array}{ccccc} X & \xrightarrow{h} & X' & \xrightarrow{k} & X'' \\ {\scriptstyle u}\downarrow & \overset{\alpha}{\Rightarrow} & {\scriptstyle u'}\downarrow & \overset{\alpha'}{\Rightarrow} & \downarrow{\scriptstyle u''} \\ Y & \xrightarrow{h'} & Y' & \xrightarrow{k'} & Y'' \\ {\scriptstyle v}\downarrow & \overset{\beta}{\Rightarrow} & {\scriptstyle v'}\downarrow & \overset{\beta'}{\Rightarrow} & \downarrow{\scriptstyle v''} \\ Z & \xrightarrow{h''} & Z' & \xrightarrow{k''} & Z'' \end{array}$$

For any (pseudo) double category $\mathbb{D}$, we denote by $\mathbb{D}_H$ the category with vertical morphisms as objects and double cells as morphisms, and by $\mathbb{D}_V$ the bicategory with horizontal morphisms as objects and double cells as morphisms. Domain and codomain maps arrange into functors $\mathrm{dom}_v, \mathrm{cod}_v\colon \mathbb{D}_H \to \mathbb{D}_h$ and $\mathrm{dom}_h, \mathrm{cod}_h\colon \mathbb{D}_V \to \mathbb{D}_v$. We will refer to dom_v and cod_v simply as dom and cod, reserving subscripts for dom_h and cod_h.

We introduce a bit more notation.

Definition 2.2. A double cell is *special* when its vertical domain and codomain are (horizontal) identities.

For any object $X \in \mathrm{ob}(\mathbb{D})$, $\mathbb{D}_H(X)$ denotes the category with

- objects all vertical morphisms to X, and
- morphisms $u \to v$ all double cells $\begin{array}{ccc} Y & \xrightarrow{h} & Y' \\ {\scriptstyle u}\downarrow & \overset{\alpha}{\Longrightarrow} & \downarrow{\scriptstyle v} \\ X & \xrightarrow[k]{} & X \end{array}$ with $\mathrm{cod}_v(\alpha) = k = id_X$.

This complies with noting $\mathbb{C}(D)$ for the pullback category (2.1), taking cod_v for F and X for D.

2.2. **Transition systems.** Beyond category theory, this paper also makes heavy use of the theory of LTSs and associated techniques, especially bisimulation and other behavioural equivalences. The notion of LTS that we'll use here is a little more general than usual. Indeed, usually, the transitions of an LTS are labelled with letters in a given set called the *alphabet*, or the set of *actions*. Here, we consider the case where the vertices of an LTS may be typed, and actions may change the type. Extending the usual theory to this setting is straightforward, so we only provide a brief overview. For more on the usual theory, modern references are Sangiorgi [50] and Sangiorgi and Rutten [51]. Our setting is essentially a baby version of Fiore's [14] (see the references therein for precursors).

Let Gph be the category of reflexive graphs, which has as objects diagrams $s, t : E \rightrightarrows V$ in Set, equipped with a further arrow $e : V \to E$ such that $s \circ e = t \circ e = id_V$. We will as usual denote $e(v)$ by id_v. Morphisms are those morphisms between underlying graphs which preserve identity arrows. Gph is thus the category of presheaves over the category

$\star \overset{s}{\underset{t}{\overleftarrow{\overset{e}{\longleftarrow}}}} [1]$ with $e \circ s = id_\star$ and $e \circ t = id_\star$.

Definition 2.3. For any $A \in \mathsf{Gph}$, let the *category of* LTS*s over* A be just the slice category Gph/A.

2.2.1. *Basic notation.* A is called the *alphabet*, which goes slightly beyond the usual notion of an alphabet. The latter would here come in the form of the graph with one vertex, an identity edge, plus an edge for each letter. By convention, and mainly to ease graphical intuitions in Sections 4 and 5, for any LTS $p : G \to A$, we understand an edge $e : x' \to x$ in G as a transition from x to x'. Of course, to recover a more standard notation, one may replace all graphs with their opposites. When e does not matter, but $p(e)$ does, we denote such a transition by $x \ {}_A\!\xleftarrow{p(e)}\ x'$, omitting the subscript A when clear from context.

For any reflexive graph A, we denote by $A^\star$ the graph with the same vertices and arbitrary paths as edges. $A^\star$ is reflexive, with identity edges given by empty paths. Similarly, $f^\star : A^\star \to B^\star$ is the morphism induced by $f : A \to B$. This defines a functor $\mathsf{Gph} \to \mathsf{Cat}$, which is *not* left adjoint to the forgetful functor $U : \mathsf{Cat} \to \mathsf{Gph}$. There is a left adjoint, though, which we denote by fc. It is given by a quotient of $A^\star$, essentially equating (id) and $()$, i.e., the singleton, identity path and the empty one.

Definition 2.4. Let $\mathsf{fc}(A)$ denote the graph with the same vertices as A, whose edges $x \to x'$ are paths $x \to^\star x'$ in A, considered equivalent modulo removal of identity edges.

Any path ρ has a normal form, obtained by removing all identity edges and denoted by $\widetilde{\rho}$. We will deem such normal forms *identity-free*. We denote by $x \overset{a}{{}_A\Longleftarrow} x'$ any path $\rho : x' \to^\star x$ in G, such that $\widetilde{p^\star(\rho)} = \widetilde{(a)}$. Concretely, if a is an identity, then $p^\star(\rho)$ only consists of identity edges; otherwise, $p^\star(\rho)$ consists of a, possibly surrounded by identity edges. In the former case, we further abbreviate the notation to $x \Longleftarrow x'$ (observe that ρ may well be empty). Similarly, for any path r in $A^\star$, $x \overset{r}{{}_A\Longleftarrow} x'$ denotes any path $\rho : x' \to^\star x$ in G such that $\widetilde{p^\star(\rho)} = \widetilde{r}$.

2.2.2. *Bisimulation and change of base.* In this section, we revisit the usual notion of (strong and weak) bisimulation in our graph-based setting, and provide a few stability results under base change and cobase change. Let us start with strong bisimulations.

Definition 2.5. For any $G, G' \in \mathsf{Gph}$, a morphism $f : G \to G'$ is a *graph fibration* iff for all $x \in G$, $y \in G'$, and $e' \in G'(y, f(x))$, there exist $x' \in G$ and $e \in G(x', x)$ such that $f(e) = e'$.

Consider morphisms $p : G \to A$ and $p' : G' \to A$. A *relation over* A is a subgraph of the pullback

$$\begin{array}{ccc} G \times_A G' & \longrightarrow & G' \\ \downarrow & \lrcorner & \downarrow{\scriptstyle p'} \\ G & \xrightarrow[p]{} & A. \end{array}$$

In particular, if two edges (e, e') are related by some $R \subseteq G \times_A G'$, then so are their sources, resp. targets. We denote such relations by $R : G \nrightarrow G'$.

We will most often deal with *full* relations, i.e., such that $R(e, e')$ iff both sources and targets are related. Of course, such relations need only to be defined on vertices.

Definition 2.6. A *simulation* $G \nrightarrow G'$ is a relation R over A such that for all $e \in G(x', x)$, if $R(x, y)$ then there exist y' and $e' \in G'(y', y)$ such that $R(e, e')$. A *bisimulation* is a simulation whose converse also is a simulation.

When R is full, R is a simulation iff for all $e \in G(x', x)$, if $R(x, y)$ then there exists y' and $e' \in G'(y', y)$ such that $R(x', y')$ and e and e' are mapped to the same edge in A.

Proposition 2.7. *R is a simulation iff its first projection $R \hookrightarrow G \times_A G' \to G$ is a graph fibration. Accordingly, R is a bisimulation iff both projections are graph fibrations.*

Proof. Straightforward. □

Remark 2.8. The characterisation of simulations in terms of graph fibrations may be attributed to Joyal et al. [30], who first observed that a morphism $f : G \to G'$ in Gph/A is a functional bisimulation iff for any commuting square as the exterior of

$$\begin{array}{ccc} \mathsf{y}(\star) & \longrightarrow & G \\ {\scriptstyle \mathsf{y}(t)}\downarrow & \nearrow & \downarrow{\scriptstyle f} \\ \mathsf{y}[1] & \longrightarrow & G', \end{array}$$

there exists a dashed arrow making both triangles commute. Here, $\mathsf{y}(t)\colon \mathsf{y}(\star) \to \mathsf{y}[1]$ maps the reflexive graph with a single vertex (and its identity edge) to the one with two vertices and just one non-identity edge e between them, by picking out the target of e. This precisely says that f is a graph fibration.

A peculiar aspect of this characterisation is that it may seem independent from A. Actually, R is a relation over $G \times_A G'$, and f is a morphism over A.

As usual, fixing G and G' over A, we have:

Proposition 2.9. *Bisimulations are closed under union, and the union of all bisimulations, called* bisimilarity*, is again a bisimulation, the maximum one.*

Considering *endo*relations $G \nrightarrow G$, we talk about bisimilarity *in* G.

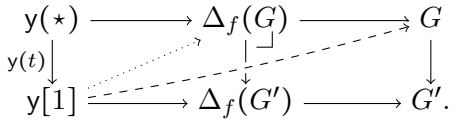
Notation 2.10. Bisimilarity in G over A is denoted by $\sim_A$. It may, upon a slight abuse of notation, be understood as an equivalence relation over all vertices of any two graphs over A. Namely, if G and G' are graphs over A, we may write $x \sim_A y$ when $x \in G$ and $y \in G'$ to mean bisimilarity in $G + G'$.

Before treating weak bisimulations, we consider a first stability result, which is all we need about strong bisimulations.

Any morphism $f\colon A \to B$ induces by pullback a change-of-base functor $\Delta_f\colon \mathsf{Gph}/B \to \mathsf{Gph}/A$, which has a left adjoint Σ_f given by composition with f.

Proposition 2.11. *For any morphism of graphs $f\colon A \to B$, both functors $\Delta_f\colon \mathsf{Gph}/B \to \mathsf{Gph}/A$ and $\Sigma_f\colon \mathsf{Gph}/A \to \mathsf{Gph}/B$, i.e., pullback along and post-composition with f, preserve functional bisimulations.*

Proof. The case of Σ_f is actually trivial. For Δ_f, we use Remark 2.8. By the pullback lemma, the square on the right below is a pullback. We check that $\Delta_f(G) \to \Delta_f(G')$ is again a bisimulation. Indeed, consider any square as on the left below:



Because $G \to G'$ is a bisimulation, we obtain the dashed arrow making both triangles commute. But then by universal property of pullback, we obtain the dotted arrow, making the corresponding bottom triangle commute. Finally, the top triangle commutes upon postcomposition with $\Delta_f(G) \to G$, and after composition with $\Delta_f(G) \to \Delta_f(G')$, hence commutes by uniqueness in the universal property of pullback. □

Remark 2.12. This is an instance of the fact that *right maps* are stable under pullback in any weak factorisation system [28], here with the factorisation system cofibrantly generated by the sole map $\mathsf{y}(t)$.

Let us now treat weak bisimulations. We start with the functional case.

Definition 2.13. A morphism $f\colon G \to G'$ in Gph/A is a *functional, weak bisimulation* iff $\mathsf{fc}(f)\colon \mathsf{fc}(G) \to \mathsf{fc}(G')$ is a graph fibration.

Proposition 2.14. *This equivalent to the fact that, for any edge $e\colon y' \to f(x)$ in G', there exists x' in G and a path $r\colon x' \to^\star x$ such that $\widetilde{f^\star(r)} = (\widetilde{e})$.*

Proof. If e is an identity, then taking the empty path for r will do, so the condition really says something about non-identity edges e. □

Remark 2.15. Remark 2.8 adapts to weak, functional bisimulations, using $\mathsf{fc}(f)$ instead of f.

Let us now handle the relational case. In the strong case, a relation between graphs G and G' over A was defined to be a subobject of the pullback $G \times_A G'$, and simulation properties were related to the projections being graph fibrations. In order to follow this pattern here, we need to consider $\mathsf{fc}(A)$ instead of A. However, in general, $\mathsf{fc}(G) \times_{\mathsf{fc}(A)} \mathsf{fc}(G')$ differs from $\mathsf{fc}(G \times_A G')$. We consider the former:

Definition 2.16. A *weak simulation* $G \nrightarrow G'$ is a relation $R \subseteq \mathsf{fc}(G) \times_{\mathsf{fc}(A)} \mathsf{fc}(G')$ whose first projection $R \hookrightarrow \mathsf{fc}(G) \times_{\mathsf{fc}(A)} \mathsf{fc}(G') \to \mathsf{fc}(G)$ is a graph fibration.

R is a *weak bisimulation* iff both projections are graph fibrations.

Explicitly, consider $p : G \to A$ and $p' : G' \to A$, and R as above a weak simulation. For any edge $r : x \leftarrow x'$ in $\mathsf{fc}(G)$, i.e., identity-free path $r : x \leftarrow^\star x'$, and $y \in G'$ such that $R(x, y)$, there should be an identity-free path $r' : y \leftarrow^\star y'$ in G' such that $(r, r') \in R$. If R is full, this is equivalent to the existence, for each edge $e : x \leftarrow x'$ in G and $y \in G'$ such that $R(x, y)$, of an identity-free path $r' : y \leftarrow^\star y'$ such that $R(x', y')$ and $\widetilde{(p(e))} = \widetilde{(p')^\star(r')}$. We will only consider full relations in this paper, hence only the last characterisation will matter to us.

As in the strong case, we have for any fixed G and G' over A:

Proposition 2.17. *Weak bisimulations are closed under union, and the union of all weak bisimulations, called* weak bisimilarity*, is again a weak bisimulation, the maximum one.*

Notation 2.18. Weak bisimilarity over A is denoted by $\approx_A$. As for strong bisimilarity, we will abuse notation and consider $\approx_A$ as a relation between the vertices of any two graphs over A.

2.3. **CCS.** The main subject of this paper is CCS [44], and fair testing equivalence over it. We work with a standard version, except in two respects. First, we work with infinite terms, which spares us the need for replication, recursion, or other possible mechanisms for describing infinite processes in a finite way. Second, we work with a de Bruijn-like presentation: terms carry their (finite) sets of known channels, in the form of a finite number. I.e., the number n indicates that the considered process knows channels $1, \dots, n$ (which complies with our notation for finite ordinals, introduced in Section 2.1).

Remark 2.19. While the de Bruijn-like presentation clearly is a matter of convenience, working with infinite terms does have an impact on our results. Restricting ourselves to recursive processes (e.g., by introducing some recursion construct), we would still have that $(\!|P|\!) \sim_f (\!|Q|\!)$ implies $P \sim_{f,s} Q$. The converse is less obvious and may be stated in very simple terms: suppose you have two recursive CCS processes P and Q and a test process T, possibly non-recursive, distinguishing P from Q; is there any recursive T' also distinguishing P from Q? We leave this question open.

Our (infinite) CCS terms are coinductively generated by the typed grammar

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P|Q} \qquad \frac{\Gamma, a \vdash P}{\Gamma \vdash \nu a.P} \qquad \frac{\dots \qquad \Gamma \vdash P_i \qquad \dots}{\Gamma \vdash \sum_{i \in n} \alpha_i.P_i}\ (n \in \mathbb{N})\,.$$

$$\frac{}{(\Gamma \vdash P) \overset{id}{\leftarrow} (\Gamma \vdash P)} \qquad \frac{}{(\Gamma \vdash \sum_{i \in n} \alpha_i.P_i) \overset{\alpha_i}{\leftarrow} (\Gamma \vdash P_i)}$$

$$\frac{(\Gamma \vdash P_1) \overset{\alpha}{\leftarrow} (\Gamma \vdash P_1')}{(\Gamma \vdash P_1 \mid P_2) \overset{\alpha}{\leftarrow} (\Gamma \vdash P_1' \mid P_2)} \qquad \frac{(\Gamma \vdash P_2) \overset{\alpha}{\leftarrow} (\Gamma \vdash P_2')}{(\Gamma \vdash P_1 \mid P_2) \overset{\alpha}{\leftarrow} (\Gamma \vdash P_1 \mid P_2')}$$

$$\frac{(\Gamma, a \vdash P) \overset{\alpha}{\leftarrow} (\Gamma, a \vdash P')}{(\Gamma \vdash \nu a.P) \overset{\alpha}{\leftarrow} (\Gamma \vdash \nu a.P')} \ (\alpha \notin \{a, \overline{a}\}) \qquad \frac{(\Gamma \vdash P_1) \overset{\alpha}{\leftarrow} (\Gamma \vdash P_1') \qquad (\Gamma \vdash P_2) \overset{\overline{\alpha}}{\leftarrow} (\Gamma \vdash P_2')}{(\Gamma \vdash P_1 \mid P_2) \overset{id}{\leftarrow} (\Gamma \vdash P_1 \mid P_2')}$$

Figure 1: CCS transitions

Here, as announced, Γ ranges over $\mathbb{N}$, i.e., the free names of a process always are $1 \dots n$ for some n. Accordingly, Γ, a denotes just $n+1$ (and then $a = n+1$). Furthermore, α_i is either a, $\overline{a}$, or $\heartsuit$ (for $a \in \Gamma$). The latter is a 'tick' move used in the definition of fair testing equivalence.

Definition 2.20. Let $\mathbb{A}$ be the reflexive graph with vertices given by finite ordinals, edges $\Gamma \to \Gamma'$ given by $\varnothing$ if $\Gamma \neq \Gamma'$, and by $\Gamma + \Gamma + \{id, \heartsuit\}$ otherwise, $id{:}\Gamma \to \Gamma$ being the identity edge on Γ. Elements of the first summand are denoted by $a \in \Gamma$, while elements of the second summand are denoted by $\overline{a}$.

We view terms as a graph *CCS* over $\mathbb{A}$ with the usual transition rules, as recalled in Figure 1 (which is an inductive definition). There, we let $\overline{\alpha}$ denote $\overline{a}$ when $\alpha = a$, or a when $\alpha = \overline{a}$.

Remark 2.21. The graph $\mathbb{A}$ only has 'endo'-edges, hence only relates terms with the same set of free channels. Some LTSs below do use more general graphs.

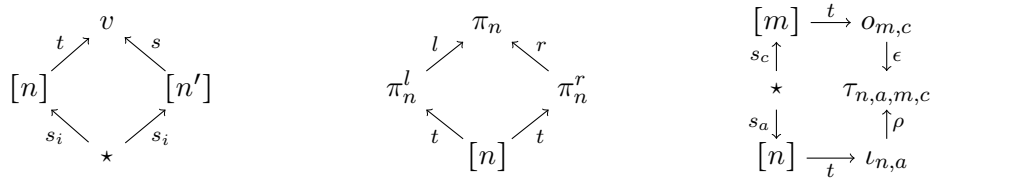
Let us finally recall the definition of fair testing equivalence. Let $\perp$ denote the set of processes P such that for all paths $P \ {}_{\mathbb{A}}\!\Leftarrow P'$, there exists a path $P' \ {}_{\mathbb{A}}\!\overset{\heartsuit}{\Leftarrow} P''$.

Definition 2.22. A *test* for $\Gamma \vdash P$ is any process $\Gamma \vdash Q$. A test Q is *passed* by P when $(\Gamma \vdash P \mid Q) \in \perp$. Two processes $\Gamma \vdash P$ and $\Gamma' \vdash P'$ are *fair testing equivalent*, notation $(\Gamma \vdash P) \sim_{f,s} (\Gamma' \vdash P')$, iff $\Gamma = \Gamma'$ and P and P' pass exactly the same tests.

3. Summary of previous work

In this section, we recall some material from HP. Apart from the admittedly numerous prerequisites mentioned in the previous section, the paper should be self-contained, although the material in this section would usefully be complemented by reading HP.

As sketched in the introduction, we construct a multi-player game, consisting of positions and plays between them. Positions are certain graph-like objects, where vertices represent players and channels. But what might be surprising is that moves are not just a binary relation between positions, because we not only want to say *when* there is a move from one position to another, but also *how* one moves from one to the other. This will be implemented by viewing moves from X to Y as *cospans* $Y \overset{s}{\to} M \overset{t}{\leftarrow} X$ in a certain category $\widehat{\mathbb{C}}^f$ of higher-dimensional graph-like objects, or 'string diagrams', where X and



$(\forall n \in \mathbb{N}, i \in n, v \in \cup_{a \in n}\{\pi^l_n, \pi^r_n, \heartsuit_n, \iota_{n,a}, o_{n,a}, \nu_n\})$ $(\forall\ n)$ $(\forall\ n \in \mathbb{N}, a \in n, \text{ and } c \in m)$

Figure 2: Equations for $\mathbb{C}$

Y respectively are the initial and final positions, and M describes how one goes from X to Y. By composing such moves (by pushout), we get a bicategory $\mathbb{D}^{CCS}_v$ of positions and plays. This is described in Sections 3.1–3.3. In Section 4, we will equip this bicategory with more structure, namely that of a pseudo double category, where one direction models dynamics, and the other models space, e.g., the inclusion of a position into another. Section 3.4 further recalls our two notions of strategies derived from the game (behaviours and innocent strategies, respectively), and Section 3.5 recalls our semantic variant of fair testing equivalence.

3.1. **Diagrams.** In preparation for the definition of our base category $\mathbb{C}$, recall that (directed, multi) graphs may be seen as presheaves over the category freely generated by the graph with two objects $\star$ and $[1]$, and two edges $s, t : \star \to [1]$. Any presheaf G represents the graph with vertices in $G(\star)$ and edges in $G[1]$, the source and target of any $e \in G[1]$ being respectively $e \cdot s$ and $e \cdot t$. A way to visualise how such presheaves represent graphs is to compute their *categories of elements* [39]. Recall that the category of elements $\int G$ for a presheaf G over $\mathbb{C}$ has as objects pairs (c, x) with $c \in \mathbb{C}$ and $x \in G(c)$, and as morphisms $(c, x) \to (d, y)$ all morphisms $f : c \to d$ in $\mathbb{C}$ such that $y \cdot f = x$. This category admits a canonical projection functor π_G to $\mathbb{C}$, and G is the colimit of the composite $\int G \xrightarrow{\pi_G} \mathbb{C} \xrightarrow{\mathsf{y}} \widehat{\mathbb{C}}$ with the Yoneda embedding. E.g., the category of elements for $\mathsf{y}[1]$ is the poset $(\star, s) \xrightarrow{s} ([1], id_{[1]}) \xleftarrow{t} (\star, t)$, which could be pictured as $\bullet \longrightarrow \bullet$, where dots represent vertices, the triangle represents the edge, and links materialise the graph of $G(s)$ and $G(t)$, the convention being that t goes from the apex of the triangle. We thus recover some graphical intuition.

Our string diagrams will also be defined as (finite) presheaves over some base category $\mathbb{C}$. Let us give the formal definition of $\mathbb{C}$ for reference. We advise to skip it on first reading, as we then attempt to provide some graphical intuition.

Definition 3.1. Let $G_{\mathbb{C}}$ be the graph with, for all $n, m \in \mathbb{N}$, $a \in n$, and $c \in m$:

- vertices $\star$, $[n]$, π^l_n, π^r_n, π_n, ν_n, $\heartsuit_n$, $\iota_{n,a}$, $o_{n,a}$, and $\tau_{n,a,m,c}$;
- edges $s_1, ..., s_n : \star \to [n]$;
- for all $v \in \{\pi^l_n, \pi^r_n, \heartsuit_n, \iota_{n,a}, o_{n,a}\}$, edges $s, t : [n] \to v$;
- edges $[n] \xrightarrow{t} \nu_n \xleftarrow{s} [n+1]$;
- edges $\pi^l_n \xrightarrow{l} \pi_n \xleftarrow{r} \pi^r_n$;
- edges $\iota_{n,a} \xrightarrow{\rho} \tau_{n,a,m,c} \xleftarrow{\epsilon} o_{m,c}$.

Let $\mathbb{C}$ be the free category on $G_{\mathbb{C}}$, modulo the equations in Figure 2, where, in the left-hand one, n' is $n+1$ when $v = \nu_n$, and n otherwise.

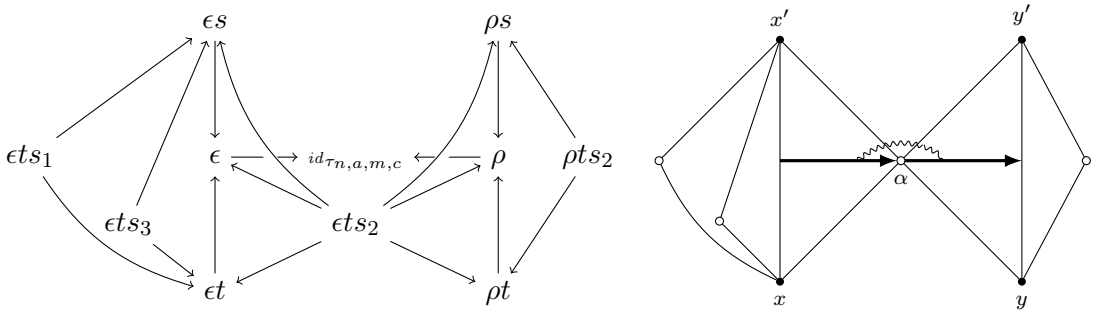
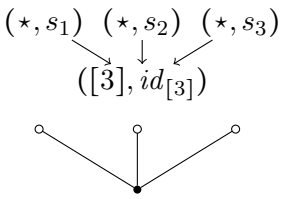


Figure 3: Category of elements for $\tau_{2,1,3,2}$ and graphical representation

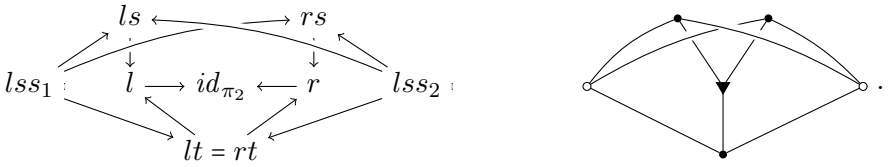
Our category of string diagrams will be the category $\widehat{\mathbb{C}}^f$ of finite presheaves on $\mathbb{C}$.

To explain this seemingly arbitrary definition, let us compute a few categories of elements. Let us start with an easy one, that of $[3] \in \mathbb{C}$ (we implicitly identify any $c \in \mathbb{C}$ with $\mathsf{y}c$). An easy computation shows that it is the poset pictured in the top part on the right. We will think of it as a position with one player $([3], id_{[3]})$ connected to three channels, and draw it as in the bottom part on the right, where the bullet represents the player, and circles represent channels. The *positions* of our game are finite presheaves empty except perhaps on $\star$ and $[n]$'s. Other objects will represent moves. The graphical representation is slightly ambiguous, because the ordering of channels known to players is implicit. We will disambiguate in the text when necessary. A *morphism of positions* is an injective morphism of presheaves. The intuition for a morphism $X \to Y$ between positions is thus that X embeds into Y.



Definition 3.2. Positions and morphisms between them form a category $\mathbb{D}_h^{CCS}$.

A more difficult category of elements is that of π_2. It is the poset generated by the graph on the left (omitting base objects for conciseness):



We think of it as a binary player (lt) forking into two players (ls and rs), and draw it as on the right. The graphical convention is that a black triangle stands for the presence of id_{π_2}, l, and r. Below, we represent just l as a white triangle with only a left-hand branch, and symmetrically for r. Furthermore, in all our pictures, time flows 'upwards'.

Another category of elements, characteristic of CCS, is the one for synchronisation $\tau_{n,a,m,c}$. The case $(n, a, m, c) = (2, 1, 3, 2)$ is the poset generated by the graph on the left of Figure 3, which we will draw as on the right. The left-hand ternary player x outputs on its 2nd channel, here α. The right-hand unary player y receives on its 1st channel, again α. Both players have two occurrences, one before and one after the move, respectively marked as x/x' and y/y'. Both x and x' have arity 3 here, and both y and y' have arity 1. There

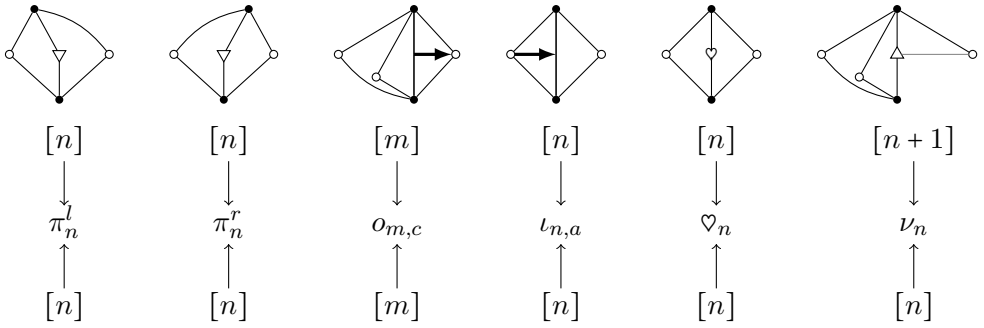


Figure 4: String diagrams and corresponding cospans for π_n^l, π_n^r, $o_{m,c}$, $\iota_{n,a}$, $\heartsuit_n$, and ν_n

are actually three moves, in the sense that there are three higher-dimensional objects in the corresponding category of elements. The first is the output move from x to x', graphically represented as the left-hand $\longrightarrow$ (intended to evoke the 'ping' sent by x entering channel α). The second move is the input move from y to y', graphically represented as the right-hand $\longrightarrow$ (intended to evoke a 'ping' exiting channel α). The third and final move is the synchronisation itself, which 'glues' the other two together, as represented by the squiggly line.

We leave the computation of other categories of elements as an exercise to the reader. The remaining diagrams are depicted in the top row of Figure 4, for $(n, a, m, c) = (2, 1, 3, 2)$. The first two are *views*, in the game semantical sense, of the fork move π_2 explained above. The next two, $o_{m,c}$ (for 'output') and $\iota_{n,a}$ (for 'input'), respectively represent what the sender and receiver can see of the above synchronisation move. The last two diagrams are a 'tick' move, used for defining fair testing equivalence, and a channel creation move.

3.2. **From diagrams to moves.** In the previous section, we have defined our category of diagrams as $\widehat{\mathbb{C}^f}$, and provided some graphical intuition on its objects. The next goal is to construct a bicategory whose objects are positions (recall: presheaves empty except perhaps on $\star$ and $[n]$'s), and whose morphisms represent plays in our game. We start in this section by defining moves as cospans in $\widehat{\mathbb{C}^f}$, and continue in the next one by explaining how to compose moves to form plays. Moves are defined in two stages: *seeds*, first, give the local form for moves, which are then defined by embedding seeds into bigger positions.

To start with, until now, our diagrams contain no information about the 'flow of time' (although it was mentioned informally for pedagogical purposes). To add this information, for each diagram M representing a move, we define its initial and final positions, say X and Y, and view the whole move as a cospan $Y \xrightarrow{s} M \xleftarrow{t} X$. We have taken care, in drawing our diagrams before, of placing initial positions at the bottom, and final positions at the top. We leave it to the reader to define, based on the above pictures, the cospans

for forking and synchronisation, plus the ones specified in the bottom row of Figure 4. In these cospans, initial positions are on the bottom row, and we denote by $[m]_{c_1,\dots,c_p}|_{a_1,\dots,a_p}[n]$ the position consisting of an m-ary player x and an n-ary player y, quotiented by the equations $x \cdot s_{c_k} = y \cdot s_{a_k}$ for all $k \in p$. When both lists are empty, by convention, $m = n$ and the players share all channels in order.

Definition 3.3. These cospans are called *seeds*.

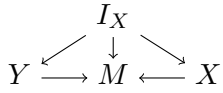
Remark 3.4. Such cospans will be used below as the morphisms of a bicategory $\mathbb{D}_v^{CCS}$, using their lower object as their *target*. Thus, we often denote the corresponding leg by t and the other by s. The reason for this convention is that it emphasises below that the fibration axiom (P1) is very close to a universal property of pullback [27].

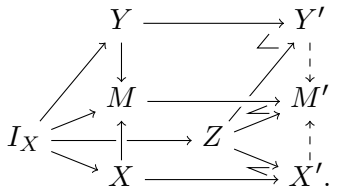
Remark 3.5. Both legs of each seed are monic, as will be below both legs of each move, and then of each play (because monics are stable under pushout in presheaf categories).

As announced, the moves of our game are obtained by embedding seeds into bigger positions. This means, e.g., allowing a fork move to occur in a position with more than one player. We proceed as follows.

Definition 3.6. Let the *interface* of a seed $Y \xrightarrow{s} M \xleftarrow{t} X$ be $I_X = X(\star) \cdot \star$, i.e., the position consisting only of the channels of the initial position of the seed. More generally, an *interface* is a position consisting only of channels.

Since channels present in the initial position remain in the final one, we have for each seed a commuting diagram as on the right. By gluing any position Z to the seed along its interface, we obtain a new cospan, say $Y' \to M' \leftarrow X'$. I.e., for any injective morphism $I_X \to Z$, we push $I_X \to X$, $I_X \to M$, and $I_X \to Y$ along $I_X \to Z$ and use the universal property of pushout, as in:



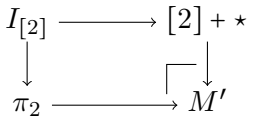


(3.1)

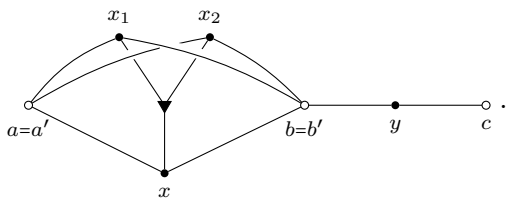
Definition 3.7. Let *moves* be all cospans obtained in this way.

Recall that colimits in presheaf categories are pointwise. So, e.g., taking pushouts along injective maps graphically corresponds to gluing diagrams together.

Example 3.8. The cospan $[2] \,|\, [2] \xrightarrow{[ls,rs]} \pi_2 \xleftarrow{lt} [2]$ has as canonical interface the presheaf $I_{[2]} = 2 \cdot \star$, consisting of two channels, say a and b. Consider the position $[2] + \star$ consisting of a player y with two channels b' and c, plus an additional channel a'. Further consider the map $h\colon I_{[2]} \to [2] + \star$ defined by $a \mapsto a'$ and $b \mapsto b'$. The pushout



is

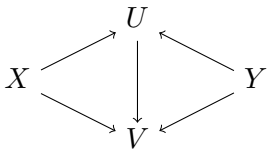


We conclude with a useful classification of moves.

Definition 3.9. A move is *full* iff it is neither a left nor a right fork. A seed is *basic* iff it is neither a full fork nor a synchronisation. We call $\mathbb{F}$ the identity-on-objects subgraph of $\mathsf{Cospan}(\widehat{\mathbb{C}}^f)$ spanning full moves.

Intuitively, a move is full when its final position contains all possible avatars of involved players.

3.3. **From moves to plays.** Having defined moves, we now define their composition to construct our bicategory $\mathbb{D}_v^{CCS}$ of positions and plays. $\mathbb{D}_v^{CCS}$ will be a sub-bicategory of $\mathsf{Cospan}(\widehat{\mathbb{C}}^f)$, the bicategory which has as objects all finite presheaves on $\mathbb{C}$, as morphisms $X \to Y$ all cospans $X \to U \leftarrow Y$, and as 2-cells $U \to V$ all commuting diagrams as on the right. Composition is given by pushout, and hence not strictly associative.

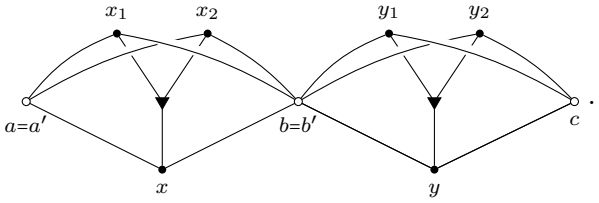


Definition 3.10. Let $\mathbb{D}_v^{CCS}$ denote the locally full subbicategory of $\mathsf{Cospan}(\widehat{\mathbb{C}}^f)$ with positions as objects, whose morphisms, *plays*, are either equivalences or isomorphic to some composite of moves.

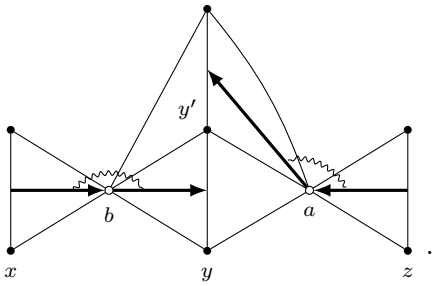
We denote morphisms in $\mathsf{Cospan}(\widehat{\mathbb{C}}^f)$ with special arrows $X \multimap Y$; composition and identities are denoted with $\bullet$ and $id^\bullet$ (recalling the notation for vertical morphisms in a pseudo double category in Section 2.1).

Again, composition by pushout glues diagrams on top of each other.

Example 3.11. Composition features some concurrency. Composing the move of Example 3.8 with a forking move by y yields



Example 3.12. Composition retains causal dependencies between moves. To see this, consider the following diagram. In the initial position, there are channels a and b, plus three players $x(b), y(a,b)$, and $z(a)$ (we indicate the channels known to each player in parentheses). In a first move, x outputs on b, while y inputs. In a second move, z outputs on a, while (the avatar y' of) y inputs. The fact that y first inputs on b then on a is encoded in the corresponding diagram, which looks like the following:



3.4. **Behaviours and strategies.**

3.4.1. *Behaviours.* Recall from HP the category $\mathbb{E}$

- whose objects are maps $U \leftarrow X$ in $\widehat{\mathbb{C}^f}$, such that there exists a play $Y \to U \leftarrow X$, i.e., objects are plays, where we forget the final position;
- and whose morphisms $(U \leftarrow X) \to (U' \leftarrow X')$ are commuting diagrams as on the right with all arrows monic.

$$\begin{array}{ccc} U & \longrightarrow & U' \\ \uparrow & & \uparrow \\ X & \longrightarrow & X' \end{array}$$

Morphisms $(U \leftarrow X) \to (U' \leftarrow X')$ in $\mathbb{E}$ represent extensions of U, both spatially (i.e., embedding into a larger position) and dynamically (i.e., adding more moves).

We may relativise this category $\mathbb{E}$ to a particular position X, yielding a category $\mathbb{E}(X)$ of plays on X as follows. Consider the functor $\mathrm{cod}{:}\mathbb{E} \to \mathbb{D}_h^{CCS}$ mapping any play $U \leftarrow X$ to its initial position X, and consider the pullback category $\mathbb{E}(X)$ as defined in Section 2.1. The objects of $\mathbb{E}(X)$ are just plays $(U \leftarrow X)$ on X, and morphisms are morphisms of plays whose lower border is id_X. This yields the definition of a category of 'naive' strategies, called behaviours.

Definition 3.13. The category B_X of *behaviours* on X is the category $\widehat{\mathbb{E}(X)}$ of presheaves of finite sets on $\mathbb{E}(X)$.

Behaviours suffer from the deficiency of allowing unwanted cooperation between players.

Example 3.14. Consider a position X with three players x, y, z sharing a channel a, and the following plays on it: in $u_{x,y}$, x outputs on a, and y inputs; in $u_{x,z}$, x outputs on a, and z inputs; in i_z, z inputs on a. One may define a behaviour S mapping $u_{x,y}$ and i_z to a singleton, and $u_{x,z}$ to $\varnothing$. Because $u_{x,y}$ is accepted, x accepts to output on a; and because i_z is accepted, z accepts to input on a. The problem is that S rejecting $u_{x,z}$ roughly amounts to x refusing to synchronise with z, or conversely.

3.4.2. *Strategies.* To rectify this, we consider the following notion of view:

Definition 3.15. Let $\mathbb{E}^{\mathbb{V}}$ denote the full subcategory of $\mathbb{E}$ consisting of *views*, i.e., composites of basic seeds.

We relativise views to a position X by considering the comma category $\mathbb{E}^{\mathbb{V}}_X$ as defined in Section 2.1. Its objects are pairs of a view $V \leftarrow [n]$ on a single n-ary player, and an embedding $[n] \hookrightarrow X$, i.e., a player of X.

Definition 3.16. The category S_X of *strategies* on X is the category $\widehat{\mathbb{E}^{\mathbb{V}}_X}$ of presheaves of finite ordinals on $\mathbb{E}^{\mathbb{V}}_X$.

Remark 3.17. We could here replace finite ordinals with a wider category and still get a valid semantics. But then to show the correspondence with the syntax below we would work with the subcategory of presheaves of finite ordinals.

This definition of strategies rules out undesired behaviours. We now sketch how to map strategies to behaviours (this is done in more detail for arbitrary playgrounds below): let first $\mathbb{E}_X$ be the category obtained by taking a comma category instead of a pullback in the definition of $\mathbb{E}(X)$. Then, embedding $\widehat{\mathbb{E}^{\mathbb{V}}_X}$ into $\widehat{\mathbb{E}^{\mathbb{V}}_X}$ via $\mathsf{ford} \hookrightarrow \mathsf{set}$, followed by right Kan

extension to $\mathbb{E}_X^{op}$ followed by restriction to $\mathbb{E}(X)^{op}$ yields a functor $\overline{(-)}\colon \mathsf{S}_X \to \mathsf{B}_X$. The image of a strategy S may be computed as in

$$\begin{array}{ccccc} (\mathbb{E}_X^{\mathbb{V}})^{op} & \hookrightarrow & \mathbb{E}_X^{op} & \hookleftarrow & \mathbb{E}(X)^{op} \\ {\scriptstyle S}\downarrow & & {\scriptstyle S'}\downarrow & \swarrow_{\overline{S}} & \\ \mathsf{ford} & \hookrightarrow & \mathsf{set}, & & \end{array}$$

where S' is here obtained by right Kan extension (the embedding $(\mathbb{E}_X^{\mathbb{V}})^{op} \hookrightarrow \mathbb{E}_X^{op}$ being full and faithful, we may choose the diagram to strictly commute). By the standard formula for right Kan extensions as ends [38] we have, for any $S\colon (\mathbb{E}_X^{\mathbb{V}})^{op} \to \mathsf{ford}$:

$$\overline{S}(U) = \int_{v \in \mathbb{E}_X^{\mathbb{V}}} S(v)^{\mathbb{E}_X(v,U)}.$$

If S is boolean, i.e., takes values in $\{\varnothing, 1\}$, then the involved end may be viewed as a conjunction, saying that U is accepted by $\overline{S}$ whenever all its views are accepted by S. Equivalently, $\overline{S}(U)$ is a limit of $(\mathbb{E}_X^{\mathbb{V}}/U)^{op} \xrightarrow{\mathrm{dom}} (\mathbb{E}_X^{\mathbb{V}})^{op} \xrightarrow{S} \mathsf{ford} \hookrightarrow \mathsf{set}$.

3.4.3. *Decomposition: a syntax for strategies.* Our definition of strategies is rather semantic in flavour. Indeed, presheaves are akin to domain theory. However, they also lend themselves well to a syntactic description (unlike behaviours). Again, this is treated at length in the abstract setting below, so we here only sketch the construction.

First, it is shown in HP that strategies on an arbitrary position X are in 1-1 correspondence with families of strategies indexed by the players of X. Recall that $[n]$ is the position consisting of one n-ary player. A player of X is the same as a morphism $[n] \to X$ (for some n) in $\mathbb{D}_h^{CCS}$. Thus, we define the set $\mathrm{Pl}(X) = \sum_{n \in \mathbb{N}} \mathbb{D}_h^{CCS}([n], X)$ of players of X.

Proposition 3.18. *We have $\mathsf{S}_X \cong \prod_{(n,x) \in \mathrm{Pl}(X)} \mathsf{S}_{[n]}$. For any $S \in \mathsf{S}_X$, we denote by $S \cdot x$ the component corresponding to $x \in \mathrm{Pl}(X)$ under this isomorphism.*

So, strategies on arbitrary positions may be entirely described by strategies on 'typical' players $[n]$. As an important particular case, we may let two strategies interact along an interface (recall from Definition 3.6 that this means a position consisting only of channels), which will be the basis of our semantic definition of fair testing equivalence. We proceed as follows. Consider any pushout Z of $X \leftarrow I \to Y$ where I is an interface. We have

Corollary 3.19. $\mathsf{S}_Z \cong \mathsf{S}_X \times \mathsf{S}_Y$.

Proof. We have $\mathbb{E}_Z^{\mathbb{V}} \cong \mathbb{E}_X^{\mathbb{V}} + \mathbb{E}_Y^{\mathbb{V}}$, and conclude by universal property of coproduct. □

We denote by $[S, T]$ the image of $(S, T) \in \mathsf{S}_X \times \mathsf{S}_Y$ under this isomorphism.

Having shown how strategies may be decomposed into strategies on 'typical' players $[n]$, we now explain that strategies on such players may be further decomposed. First, we observe that $\mathbb{E}_{[n]}^{\mathbb{V}}$ is isomorphic to the full subcategory $\mathbb{E}^{\mathbb{V}}([n])$ of $\mathbb{E}([n])$ spanning views. For any strategy S on $[n]$ and seed $b\colon [n'] \rightarrowtail\!\!\!\!\bullet\, [n]$, let the *residual* $S \cdot b$ of S after b be the strategy playing like S after b, i.e., for all $v \in \mathbb{E}_{[n']}^{\mathbb{V}}$, $(S \cdot b)(v) = S(b \bullet v)$. S is almost determined by its residuals. The only information missing from the $S \cdot b$'s to reconstruct S is the set of initial states and how they relate to the initial states of each $(S \cdot b)$. This may be taken into account as follows.

Definition 3.20. For any $S \in \mathsf{S}_{[n]}$ and initial state $\sigma \in S(id^\bullet)$, let $S_{|\sigma}$, the *restriction* of S to σ, be determined by

$$S_{|\sigma}(v) = \{\sigma' \in S(v) \mid S(!_v)(\sigma') = \sigma\},$$

where $!_v$ denotes the unique morphism $!\colon id^\bullet \to v$.

S is determined by its set $S(id^\bullet)$ of initial states, plus the map $(\sigma, b) \mapsto (S_{|\sigma} \cdot b)$ sending any $\sigma \in S(id^\bullet)$ and isomorphism class b of seeds to $S_{|\sigma} \cdot b$. In other words, we have for all n:

Theorem 3.21. $\mathsf{S}_{[n]} \cong (\prod_{n' \in \mathbb{N}, b:[n'] \to\!\!\!\bullet\!\!\!\to [n]} \mathsf{S}_{[n']})^\star$.

Given an element $(D_1, \ldots, D_m)$ of the right-hand side, the corresponding strategy maps the identity view $id^\bullet$ to m, and any non-identity view $b \bullet v$ on $[n]$ to the sum $\sum_{i \in m} D_i(b)(v)$.

A closely related result is that strategies on a player $[n]$ are in bijection with infinite terms in the following typed grammar, with judgements $n \vdash_{\mathsf{D}} D$ and $n \vdash S$, where D is called a *definite strategy* and S is a *strategy*:

$$\frac{\ldots\ n_b \vdash S_b\ \ldots\ (\forall b{:}[n_b] \to\!\!\!\bullet\!\!\!\to [n] \in [\mathbb{B}]_n)}{n \vdash_{\mathsf{D}} \langle (S_b)_{b \in [\mathbb{B}]_n} \rangle} \qquad \frac{\ldots\ n \vdash_{\mathsf{D}} D_i\ \ldots\ (\forall i \in m)}{n \vdash \oplus_{i \in m} D_i}\ (m \in \mathbb{N}).$$

Here, $[\mathbb{B}]_n$ denotes the set of all isomorphism classes of seeds from $[n]$. This achieves the promised syntactic description of strategies. We may readily define the translation of CCS processes, coinductively, as follows. For processes with channels in Γ, we define

$$\begin{array}{rclcrcl} (\!|\textstyle\sum_{i \in n} \alpha_i.P_i|\!) & = & \langle b \mapsto \oplus_{\{i \in n \mid b = (\!|\alpha_i|\!)\}} (\!|P_i|\!) \rangle & & (\!|a|\!) & = & \iota_{\Gamma,a} \\ (\!|\nu a.P|\!) & = & \langle \nu_\Gamma \mapsto (\!|P|\!), \_ \mapsto \varnothing \rangle & & (\!|\overline{a}|\!) & = & o_{\Gamma,a} \\ (\!|P \mid Q|\!) & = & \langle \pi^l_\Gamma \mapsto (\!|P|\!), \pi^r_\Gamma \mapsto (\!|Q|\!), \_ \mapsto \varnothing \rangle & & (\!|\heartsuit|\!) & = & \heartsuit_\Gamma. \end{array} \tag{3.2}$$

For example, $a.P + a.Q + \bar{b}.R$ is mapped to

$$\langle \iota_{\Gamma,a} \mapsto ((\!|P|\!) \oplus (\!|Q|\!)), o_{\Gamma,b} \mapsto (\!|R|\!), \_ \mapsto \varnothing \rangle.$$

3.5. **Semantic fair testing.** The tools developed in the previous section yield the following semantic analogue of fair testing equivalence.

Definition 3.22. *Closed-world* moves are those generated by some seed among ν_n,$\heartsuit_n$,π_n, and $\tau_{n,i,m,j}$. A play is *closed-world* when it is a composite of closed-world moves. Let a closed-world play be *successful* when it contains a $\heartsuit$ move, and *unsuccessful* otherwise. A state $\sigma \in B(U)$ of a behaviour $B \in \mathsf{B}_Z$ over a closed-world play $U \leftarrow Z$ is successful when the play U is, and unsuccessful otherwise.

Let then $\perp\!\!\!\perp_Z$ denote the set of behaviours $B \in \mathsf{B}_Z$ such that any unsuccessful, closed-world state admits a successful extension. Formally:

Definition 3.23. Let $B \in \perp\!\!\!\perp_Z$ iff, for any unsuccessful, closed-world play $U \leftarrow Z$ and $\sigma \in B(U)$, there exists a successful, closed-world U', a morphism $f\colon U \to U'$ in $\mathbb{E}(Z)$, and a state $\sigma' \in B(U')$ such that $\sigma' \cdot f = \sigma$.

Finally, let us say that a triple (I, h, S), for any $h\colon I \to X$ (where I is an interface) and $S \in \mathsf{S}_X$, *passes* the test consisting of a morphism $k\colon I \to Y$ of positions and a strategy $T \in \mathsf{S}_Y$ iff $\overline{[S,T]} \in \perp\!\!\!\perp_Z$, where Z is the pushout of h and k. Let $(I, h, S)^{\perp\!\!\!\perp}$ denote the set of all such (k, T).

Definition 3.24. For any $h\colon I \to X$, $h'\colon I' \to X'$, $S \in \mathsf{S}_X$, and $S' \in \mathsf{S}_{X'}$, $(I,h,S) \sim_f (I',h',S')$ iff $I = I'$ and $(I,h,S)^{\perp\!\!\!\perp} = (I,h',S')^{\perp\!\!\!\perp}$.

Obviously, $\sim_f$ is an equivalence relation, analogous to standard fair testing equivalence, which we hence also call (semantic) fair testing equivalence.

This raises the question of whether the translation $(\!|-|\!)$ preserves or reflects fair testing equivalence. The rest of the paper is devoted to proving that it does both. As announced in the introduction, this is done by organising the game into a *playground*, as defined in the next section.

4. Playgrounds: from behaviours to strategies

4.1. **Motivation: a pseudo double category.** We start by organising the game described above into a (pseudo) double category. We have seen that positions are the objects of the category $\mathbb{D}_h^{CCS}$, whose morphisms are embeddings of positions. We have also seen that positions are the objects of the bicategory $\mathbb{D}_v^{CCS}$, whose morphisms are plays. It should seem natural to define a pseudo double category structure with

$$\begin{array}{ccc} Y' & \xrightarrow{h} & Y \\ {\scriptstyle s'}\downarrow & & \downarrow{\scriptstyle s} \\ U' & \xrightarrow[k]{} & U \\ {\scriptstyle t'}\uparrow & & \uparrow{\scriptstyle t} \\ X' & \xrightarrow[l]{} & X \end{array}$$

- $\mathbb{D}_h^{CCS}$ as horizontal category,
- $\mathbb{D}_v^{CCS}$ as vertical bicategory,
- commuting diagrams as on the right as double cells.

Here, X is the initial position and Y is the final one; all arrows are mono. This forms a pseudo double category $\mathbb{D}^{CCS}$, and we have:

Proposition 4.1. *The functor* $\mathrm{cod}_v\colon \mathbb{D}_H^{CCS} \to \mathbb{D}_h^{CCS}$ *is a Grothendieck fibration [27].*

Intuitively, cod_v being a fibration demands some canonical way of *restricting* a given play on some position X to some 'subposition' $X' \to X$. More technically, it amounts to the existence, for all plays $Y \overset{u}{\dashrightarrow} X$ and horizontal morphisms $X' \overset{l}{\to} X$, of a universal ($\approx$ maximal) way of restricting u to X', as on the left below:

$$\begin{array}{ccc} Y' & \overset{h}{\dashrightarrow} & Y \\ {\scriptstyle u'}\vdots & \overset{\alpha}{\Rightarrow} & \vdots{\scriptstyle u} \\ X' & \xrightarrow{l} & X \end{array} \qquad \begin{array}{ccccc} E'' & & \xrightarrow{\quad t \quad} & & \\ & \searrow^{s} & & & \\ \downarrow & & E' & \xrightarrow[r]{} & E \\ p(E'') & & \xrightarrow{\quad p(t) \quad} & & \downarrow \\ & \searrow_{k} & & \searrow & \\ & & p(E') & \xrightarrow[p(r)]{} & p(E). \end{array}$$

Formally, consider any functor $p\colon \mathbb{E} \to \mathbb{B}$. A morphism $r\colon E' \to E$ in $\mathbb{E}$ is *cartesian* when, as on the right above, for all $t\colon E'' \to E$ and $k\colon p(E'') \to p(E')$, if $p(r) \circ k = p(t)$ then there exists a unique $s\colon E'' \to E'$ such that $p(s) = k$ and $r \circ s = t$.

Definition 4.2. A functor $p\colon \mathbb{E} \to \mathbb{B}$ is a *fibration* iff for all $E \in \mathbb{E}$, any $h\colon B' \to p(E)$ has a cartesian lifting, i.e., a cartesian antecedent by p.

Proposition 4.1 is proved among other facts in Section 7. This was the starting point of the notion of playground: which axioms should we demand of a pseudo double category in order to enable the constructions of HP? We follow the constructions in this section, considering an arbitrary pseudo double category $\mathbb{D}$, on which we impose axioms along the way. Objects and vertical morphisms will respectively be called *positions* and *plays*. The pseudo double category $\mathbb{D}^{CCS}$ does satisfy the axioms, albeit in a non-trivial way. This is stated and proved in Section 7, but we use the result in advance in examples to illustrate our constructions.

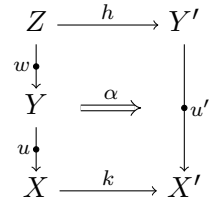
Let us record the axioms imposed on $\mathbb{D}$ in the next sections to obtain our bisimulation result (Theorem 5.35):

- (P1), page 25,
- (P2)—(P5), page 27,
- (P6), page 27,
- (P7), page 28,
- (P8), page 28,
- (P9), page 35,
- (P10), page 42.

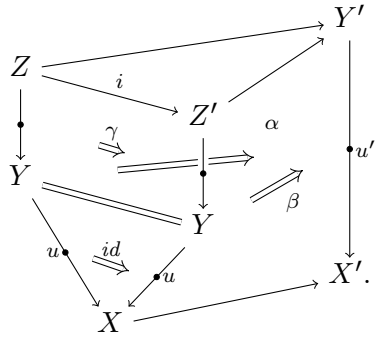
4.2. **Behaviours.** The easiest construction of HP to carry over to the abstract setting of playgrounds is that of behaviours. First, let us stress that, in the case of $\mathbb{D}^{CCS}$, $\mathbb{D}_H^{CCS}$ is very different from the category of plays called $\mathbb{E}$ recalled in Section 3.4.1. Indeed, any morphism $\alpha : u \to u'$ in $\mathbb{D}_H^{CCS}$ in particular induces an embedding of the final position $\mathrm{dom}(u)$ of u into that of u'. In $\mathbb{E}$, instead, a morphism $U \to U'$ may involve extending U with more moves.

Example 4.3. The move of Example 3.8 embeds into the play of Example 3.11 in the sense of $\mathbb{E}$, but not in the sense of $\mathbb{D}_H^{CCS}$. Indeed, the passive player y of Example 3.8 does belong to the final position, but its image in Example 3.11 does not.

So our first step is to construct an analogue of $\mathbb{E}$ from any playground $\mathbb{D}$. Intuitively, it should have as objects all plays, and as morphisms $u \to u'$ all pairs (w, α) as on the right. However, this definition is slightly wrong on morphisms, in that α carries some information about how w embeds into u', while we are only interested in how u does. Thus, we instead define morphisms $u \to u'$ to be pairs (w, α) as in (4.1), quotiented by the equivalence relation generated by pairs $((w, \alpha), (w', \beta))$ such that there exists morphisms i and γ satisfying $\alpha = \beta \circ (u \bullet \gamma)$, as in



(4.1)

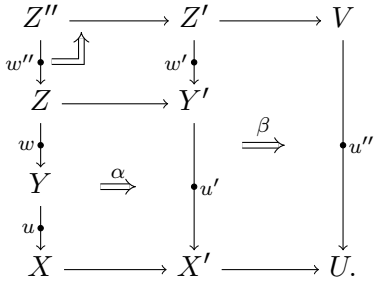


(4.2)

In order to define composition in this category, we state the following axiom (cf. Proposition 4.1).

Axiom. (P1) (Fibration) The vertical codomain functor $\mathrm{cod} : \mathbb{D}_H \to \mathbb{D}_h$ is a fibration.

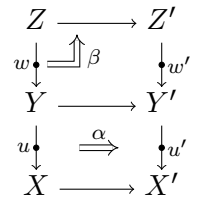
Composition may now be defined by pullback (i.e., cartesian lifting in the fibration $\mathrm{cod}\colon \mathbb{D}_H \to \mathbb{D}_h$) and pasting:



(We use 'double pullback' marks to denote cartesian double cells.) Quotienting makes composition functional and associative, and furthermore it is compatible with the above equivalence. Identities are obvious.

Proposition 4.4. *This forms a category* $\mathbb{E}$.

Example 4.5. Consider the move M' from Example 3.8, and let us name its initial and final positions as in $M'\colon Y' \nrightarrow X'$. Let us further call $U\colon Y'' \nrightarrow X'$ the play from Example 3.11, obtained by composing M' with a forking move by $y \in Y'[2]$. In order to obtain a double cell $M' \to U$, we need to provide an extension of M' with some move by y, and there are actually three ways of doing this. One is with a left forking move, another is with a right forking move, and the last is with a full forking move. In this example, the last possibility actually yields an identity double cell $U \to U$, and may be obtained using (P1) in the following general way. Consider any double cell $\alpha\colon u \to u'$ in $\mathbb{D}_H$, and play w' such that $u' \bullet w'$ is well-defined. Then, letting $\beta\colon w \to w'$ be the cartesian lifting of w' along $\mathrm{dom}(\alpha)$, we obtain a morphism $u \to u' \bullet w'$ in $\mathbb{E}$, as in on the right. The universal property of β here amounts to the fact that left and right forking moves both embed uniquely into full forking, which makes our three candidate morphisms $u \to u' \bullet w'$ equal in $\mathbb{E}$.



Recalling notation from Section 2.1, consider now the pullback category $\mathbb{E}(X)$, where X is any position. Following Definition 3.13, we state:

Definition 4.6. The category B_X of *behaviours* on X is $\overline{\mathbb{E}(X)}$, i.e., the category of presheaves of finite sets on $\mathbb{E}(X)$.

This construction has a bit of structure. Indeed, the map $X \mapsto \mathbb{E}(X)$ extends to a pseudo functor $\mathbb{E}(-)\colon \mathbb{D}_v \to \mathsf{Cat}$ by vertical post-composition. Post-composing the opposite of this pseudo functor by $\overline{(-)}\colon \mathsf{Cat}^{op} \to \mathsf{Cat}$, we obtain a pseudo functor $\mathsf{B}_-\colon \mathbb{D}_v^{op} \to \mathsf{Cat}$, satisfying $\mathsf{B}_u(B)(u') = B(u \bullet u')$.

4.3. **More axioms.** We now turn to generalising further constructions of HP to the general setting of playgrounds. We mentioned in Section 3 that strategies on a position X should be defined as presheaves on the category of views on X. We will further want to generalise the decomposition theorems for strategies of HP, which crucially rely on a property of views stated (in Section 4.4 below) as Proposition 4.27.

In order for this to work, we need to state more axioms for $\mathbb{D}$. In particular, the axioms equip $\mathbb{D}$ with a notion of *player* for a position X. Each position has a set of players, each

player having a certain 'type'. Furthermore, in Section 4.4, $\mathbb{D}$ is equipped with a notion of view; and views have a type, too. Proposition 4.27, e.g., states that views on a position X form a coproduct, over all players x in X, of views over the type of x.

We first state a series of simple axioms, and then, building on these, two more complicated axioms.

Axiom. $\mathbb{D}$ is equipped with

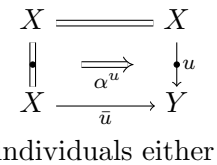
- a full subcategory $\mathbb{I} \hookrightarrow \mathbb{D}_h$ of objects called *individuals*,
- a replete class $\mathbb{M}$ of vertical morphisms called *moves*, with replete subclasses $\mathbb{B}$ and $\mathbb{F}$, respectively called *basic* and *full* moves,
- a map $|-| : \mathrm{ob}(\mathbb{D}_H) \to \mathbb{N}$ called the *length*,

satisfying the following conditions:

(P2) $\mathbb{I}$ is discrete. Basic moves have no non-trivial automorphisms in $\mathbb{D}_H$. Vertical identities on individuals have no non-trivial endomorphisms.

(P3) (Individuality) Basic moves have individuals as both domain and codomain.

(P4) (Atomicity) For any cell $\alpha : v \to u$, if $|u| = 0$ then also $|v| = 0$. Up to a special isomorphism in $\mathbb{D}_H$, all plays u of length $n > 0$ admit decompositions into n moves. For any $u : X \bullet\!\!\to Y$ of length 0, there is an isomorphism $id^{\bullet}_X \to u$ as on the right in $\mathbb{D}_H$.



(P5) (Fibration, continued) Restrictions of moves (resp. full moves) to individuals either are moves (resp. full moves), or have length 0.

Replete means stable under isomorphism (here in $\mathbb{D}_H$). In (P5), *restriction* is w.r.t. the fibration $\mathrm{cod} : \mathbb{D}_H \to \mathbb{D}_h$, as explained below Proposition 4.1.

Definition 4.7. A *player* in a position (i.e., object) X, is a pair (d, x), where $d \in \mathbb{I}$ and $x : d \to X$. Let $\mathrm{Pl}(X) = \sum_{d \in \mathbb{I}} \mathbb{D}_h(d, X)$ be the set of players of X.

Example 4.8. In $\mathbb{D}^{CCS}$, individuals are representable positions $[n]$, which consist for some n of a single n-ary player, connected to n distinct channels. Importantly, for each isomorphism class of such positions we pick one representative: this makes $\mathbb{I}$ discrete by Yoneda. Furthermore, basic moves are basic seeds.

Here is a further, crucial axiom.

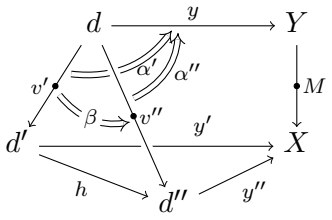
Definition 4.9. Let $\mathbb{B}_0$ be the full subcategory of $\mathbb{D}_H$ having as objects basic moves and morphisms of length 0 between individuals.

Axiom. (P6) (Views) For any move $M : Y \bullet\!\!\to X$ in $\mathbb{D}_v$, the domain functor $\mathrm{dom} : \mathbb{B}_0/M \to \mathbb{I}/Y$ is an equivalence of categories.

In elementary terms, for any $y : d \to Y$ in $\mathbb{D}_h$ with $d \in \mathbb{I}$, there exists a cell

$$\begin{array}{ccc} d & \xrightarrow{\;y\;} & Y \\ {\scriptstyle v^{y,M}}\downarrow & \overset{\alpha^{y,M}}{=\!=\!=\!\Rightarrow} & \downarrow{\scriptstyle M} \\ d^{y,M} & \xrightarrow[y^M]{\quad\quad} & X, \end{array}$$

with $v^{y,M} \in \mathbb{B}_0$, which is unique up to canonical isomorphism of such. An isomorphism between two such tuples, say (d', v', y', α') and $(d'', v'', y'', \alpha'')$ is a diagram

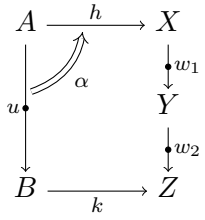


such that $\alpha'' \circ \beta = \alpha'$ (where necessarily $d' = d''$, $h = id$, and $y' = y''$).

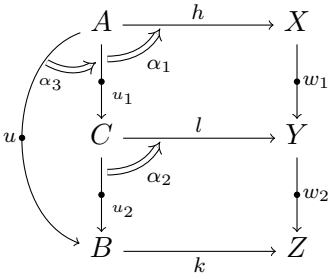
Example 4.10. This axiom is obviously satisfied by $\mathbb{D}^{CCS}$.

We then have two decomposition axioms.

Axiom. (P7) (Left decomposition) Any double cell



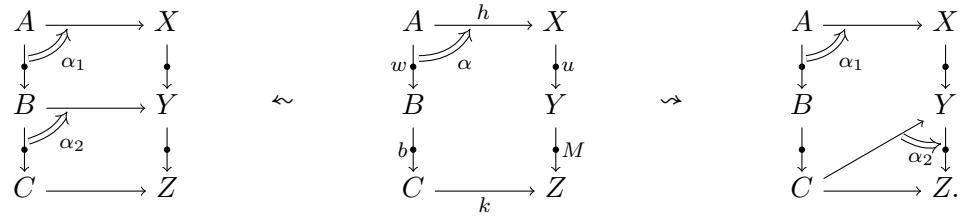
decomposes as



with α_3 an isomorphism, in an essentially unique way.

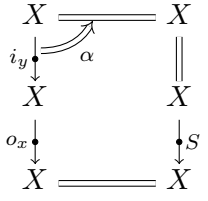
Here is our second decomposition axiom.

Axiom. (P8) (Right decomposition) Any double cell as in the center below, where b is a basic move and M is a move, decomposes in exactly one of the forms on the left and right:



Remark 4.11. This axiom takes pseudoness rather sloppily. Indeed, the domain of the right-hand composite is not really $b \bullet w$, but rather $id^{\bullet}_{C} \bullet (b \bullet w)$. So we actually mean $\alpha = (\alpha_2 \bullet \alpha_1) \circ \lambda^{-1}_{b \bullet w}$, where λ cancels identities on the left.

Example 4.12.
That this axiom is satisfied by $\mathbb{D}^{CCS}$ is not obvious and is proved in Section 7. However, let us disprove the more general version where b is not required to be basic. Let X consist of two players x and y sharing a channel a. Let $i_y : X \nrightarrow X$ be the play where y inputs on a, $o_x : X \nrightarrow X$ be the play where x outputs on a, and let $S : X \nrightarrow X$ be the play where both players synchronise on a. We obtain a double cell as on the right, which does not decompose as in (P8). The problem here is that, on the left-hand side, the upper input by y has to be mapped to S, which prevents any suitable decomposition.



We now define and study views.

4.4. **Views.**

Definition 4.13. A *view* in $\mathbb{D}$ is a play which is specially isomorphic in $\mathbb{D}_H$ to a possibly empty (vertical) composite of basic moves. I.e., if $d_n \xrightarrow{b_n} d_{n-1} \dots d_1 \xrightarrow{b_1} d_0$ are all basic moves, then the composite is a view. Let $\mathbb{V}$ be the full subcategory of $\mathbb{D}_H$ consisting of views.

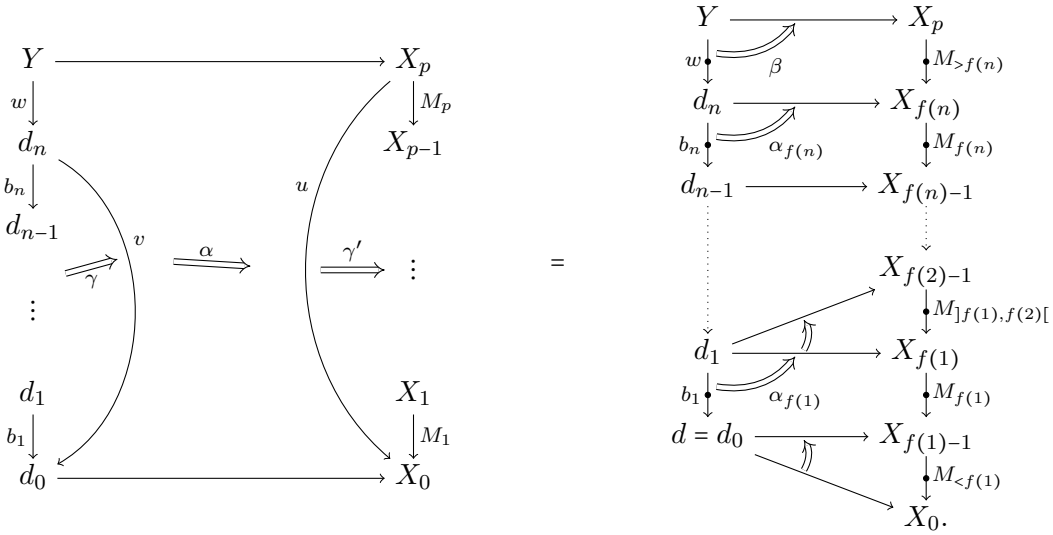
The definition includes the 'identity' view $id^{\bullet}_{d}$. In $\mathbb{D}^{CCS}$, this of course coincides with views as defined in HP.

Here is an important consequence of our axioms. It is a bit complicated to state, but very useful in the (more intelligible) developments on views below.

Lemma 4.14. *For all plays $w : Y \rightarrowtail\!\!\bullet\, d_n$ and $u : X_p \rightarrowtail\!\!\bullet\, X_0$, views $v : d_n \rightarrowtail\!\!\bullet\, d_0$, and double cells $\alpha : v \bullet w \to u$, for all special isomorphisms $\gamma : (b_1 \bullet (\dots (b_{n-1} \bullet b_n) \dots)) \to v$ and $\gamma' : u \to (M_1 \bullet (\dots (M_{p-1} \bullet M_p) \dots))$ decomposing v and u into moves, there exists a unique, strictly monotone map $f : n \cup \{0\} \to p \cup \{0\}$ with $f(0) = 0$ and double cells $\beta : w \to (M_{f(n)+1} \bullet (\dots \bullet M_p))$ and $\alpha_k : \bar{b}_k \to M_k$ for $1 \le k \le f(n)$, where*

$$\bar{b}_k = \begin{cases} b_i & \text{(if } k \in \mathrm{Im}(f)\text{, with } f(i) = k\text{)} \\ id^{\bullet}_{\mathrm{cod}(b_{\min\{i \in n \mid f(i) > k\}})} & \text{(otherwise),} \end{cases}$$

such that $\alpha_1 \bullet (\dots \bullet (\alpha_{f(n)} \bullet \beta)) = \gamma' \circ \alpha \circ (\gamma \bullet w)$, as in



In the case $p = 0$, also $n = |w| = 0$, and the decomposition of u should be understood as $M_1 \bullet \dots \bullet M_{f(n)}$ being an *identity*, with $M_{f(n)+1} \bullet \dots \bullet M_p$ being u.

Remark 4.15. Only f is claimed to be unique here. Furthermore, as in (P8), we are a bit sloppy regarding pseudoness. Also, in the following, we consider only the underlying map $f : n \to p$, implicitly extended with $f(0) = 0$. Finally, for all $\alpha : v \bullet w \to u$, there exist γ and γ' as in the lemma. This is obvious when n and $p \neq 0$; we just explained it for the case $p = 0$; and when $n = 0$ it follows from Lemma 4.17 below.

Proof. We proceed by lexicographic induction on the pair (n, p).

If $n = 0$ then our map $f : n \to p$ is the unique map $0 \to p$, $f(n) = 0$, and we take $\beta = \gamma' \circ \alpha \circ \gamma$. Otherwise, we apply (P8) with $b = b_1$, $w = (b_2 \bullet \ldots \bullet b_{n-1} \bullet w)$, $M = M_1$ and $u = (M_2 \bullet \ldots \bullet M_{p-1})$.

- If we are in the left-hand case, α decomposes as $\alpha_1 \bullet \alpha_2$, with $\alpha_1 : b_1 \to M_1$ and $\alpha_2 : (b_2 \bullet \ldots \bullet b_n \bullet w) \to (M_2 \bullet \ldots \bullet M_p)$. By induction hypothesis, we obtain a map $f' : n-1 \to p-1$ and a corresponding decomposition of α_2. We then let $f : n \to p$ map 1 to 1, and $k+1$ to $f'(k)+1$ for any $k \in (n-1)$.
- If we are in the right-hand case, we obtain a map $f' : n \to p-1$, and return the map $k \mapsto f'(k) + 1$.

This shows existence of the desired decomposition. For uniqueness, consider any map $g : n \to p$ and corresponding decomposition. Axiom (P7) entails that at each stage, $f^{-1}\{1, \ldots, k\}$ and $g^{-1}\{1, \ldots, k\}$ have the same cardinality. Indeed, otherwise, we would find isomorphic decompositions of $b_1 \bullet \ldots \bullet (b_n \bullet w)$ with incompatible lengths. Thus, $f = g$. □

We continue with a few easy results. Recall the family of isomorphisms α^u from Axiom (P4), indexed by vertical morphisms of length 0. Furthermore, let us denote by $\rho_u : u \bullet id^\bullet_X \to u$ and $\lambda_u : id^\bullet_Y \bullet u \to u$ the coherence isomorphisms from $\mathbb{D}_v$ for cancelling vertical identities.

Lemma 4.16. *For any $u : X \nrightarrow Y$ of length 0, there is an isomorphism*

$$\begin{array}{ccc} X & \xrightarrow{\bar{u}} & Y \\ {\scriptstyle u}\downarrow & \overset{\alpha_u}{\Longrightarrow} & \| \\ Y & = & Y \end{array}$$

in $\mathbb{D}_H$, such that $\alpha_u \bullet \alpha^u = \lambda_u^{-1} \circ \rho_u$ and $\alpha_u \circ \alpha^u = id^\bullet_{\bar{u}}$.

Proof. Pose $\alpha_u = id^\bullet_{\bar{u}} \circ (\alpha^u)^{-1}$. □

Lemma 4.17. *If $b : d \nrightarrow d'$ has length 0, then $d = d'$, $\bar{b} = id_d$, and α_b and α^b are horizontal inverses.*

Proof. By (P2). □

Lemma 4.18. $\mathbb{B}_0$ *(Definition 4.9) is a groupoid.*

Proof. This means that any $\alpha : b \to b'$ in $\mathbb{B}_0$ is an isomorphism. Let $b : d_1 \nrightarrow d_2$ and $b' : d'_1 \nrightarrow d'_2$. Existence of α entails $d_1 = d'_1$ and $d_2 = d'_2$, by (P2).

If $b' \in \mathbb{B}$, then α and $id_{b'}$ are both mapped by $\mathrm{dom} : \mathbb{B}_0/b' \to \mathbb{I}/d'_1$ to $\mathrm{dom}(\alpha) = id_{d'_1}$. By (P6), there is thus a unique isomorphism $\gamma : b \to b'$ in $\mathbb{D}_H$ such that $id_{b'} \circ \gamma = \alpha$, i.e., $\gamma = \alpha$. This shows that α is an iso.

If b' has length 0, then by (P4) we furthermore have $|b| = 0$ and $d_1 = d_2 = d'_1 = d'_2$. Moreover, the composite $\alpha_{b'} \circ \alpha \circ \alpha^b$ (with $\alpha_{b'}$ and α^b as in Lemma 4.16 and (P4)) is an endomorphism of $id^\bullet_{d_1}$, hence $id_{id^\bullet_{d_1}}$ by (P2). It is thus an isomorphism, hence so is $\alpha^{b'} \circ \alpha_{b'} \circ \alpha \circ \alpha^b \circ \alpha_b$, which is equal to α by two applications of Lemma 4.17. □

Lemma 4.19. *In any category $\mathbb{C}$, for any object c isomorphic to an object d such that d has no non-trivial endomorphisms, c does not have any non-trivial endomorphisms either.*

Proof. By the Yoneda lemma, we have $\mathbb{C}(c, c) \cong \mathbb{C}(c, d) \cong \mathbb{C}(d, d) \cong 1$. □

Lemma 4.20. *Any groupoid $\mathbb{C}$ whose objects have no non-trivial endomorphisms is an equivalence relation.*

Proof. For any objects c and d, we have that if $\mathbb{C}(c,d)$ is non-empty then c and d are isomorphic, so by Yoneda $\mathbb{C}(c,d) \cong \mathbb{C}(c,c) \cong 1$. □

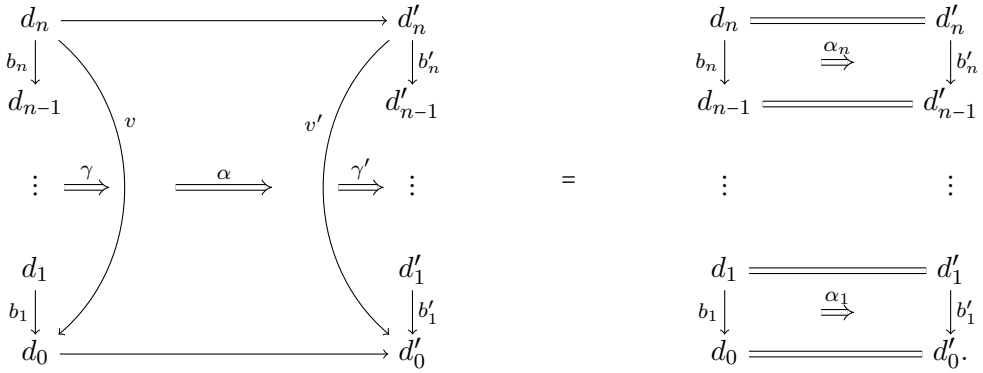
Corollary 4.21. *$\mathbb{B}_0$ is an equivalence relation.*

This adds to Lemma 4.18 that there is at most one morphism between any two objects.

Proof. By Lemma 4.19 and (P4), its objects have no non-trivial automorphisms, which in a groupoid is the same as having no non-trivial endomorphisms. By the last result, $\mathbb{B}_0$ is an equivalence relation. □

This leads to a better understanding of $\mathbb{V}$.

Lemma 4.22. *Consider any morphism of views $\alpha : v \to v'$, with isomorphisms $\gamma : (b_1 \bullet \ldots \bullet b_n) \to v$ and $\gamma' : v' \to (b'_1 \bullet \ldots \bullet b'_{n'})$, for basic moves $b_i : d_i \to\!\!\!\!\bullet\, d_{i-1}$ and $b'_j : d'_j \to\!\!\!\!\bullet\, d'_{j-1}$ for all $i \in n$ and $j \in n'$. We have $n = n'$, $d_{i-1} = d'_{i-1}$ for all $i \in n+1$, and there exist unique isomorphisms $\alpha_i : b_i \to b'_i$ such that $\gamma' \circ \alpha \circ \gamma = (\alpha_n \bullet \ldots \bullet \alpha_1)$, as in*



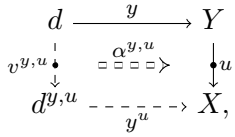
Proof. Applying Lemma 4.14 with $w = id^{\bullet}_{d_n}$ yields $f : n \to n'$ which by Corollary 4.21 and (P4) has to be a bijection. This yields the desired α_i's, which are unique by Corollary 4.21 again. □

This entails:

Corollary 4.23. *$\mathbb{V}$ is an equivalence relation, compatible with length.*

Here is an analogue of (P6) for general plays and views instead of just moves and basic moves.

Proposition 4.24. *For any $y : d \to Y$ in $\mathbb{D}_h$ with $d \in \mathbb{I}$, and any $u : Y \to\!\!\!\!\bullet\, X$ in $\mathbb{D}_v$, there exists a cell*

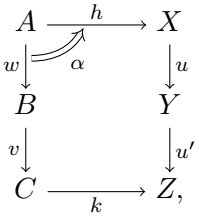


with $v^{y,u}$ a view, which is unique up to canonical isomorphism of such.

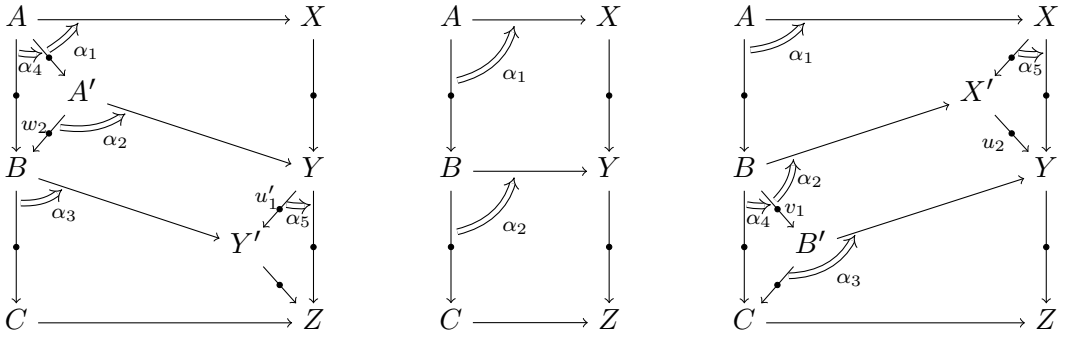
Proof. We find $v^{y,u}$ by repeated application of (P6). For essential uniqueness, by repeated application of (P6), we find an isomorphism between any two such views, which by Corollary 4.23 is unique. □

We continue with an analogue of (P8):

Proposition 4.25. *Any double cell*



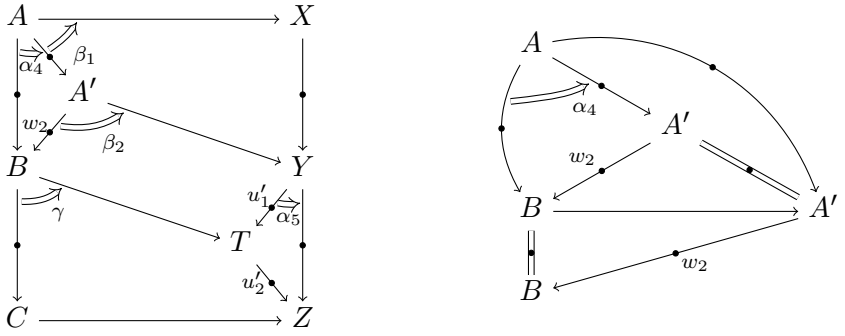
where v is a view, decomposes in exactly one of the following forms:



with $|w_2| > 0$, $|v_1| > 0$, and α_4 and α_5 iso in $\mathbb{D}_H$.

A possible reading of this is that in the left and middle cases, the whole of v embeds into u'. In the left case, a non-trivial part of w embeds into u'. In the right case, a non-trivial part of v embeds into u.

Proof. Choose decompositions of u' and u as $M_1 \bullet \ldots \bullet M_p$ and $M_{p+1} \bullet \ldots \bullet M_{p+q}$, respectively, and of v as $b_1 \bullet \ldots \bullet b_n$. Apply Lemma 4.14 to obtain $f : n \to p+q$. If $f(n) > p$, we are in the right-hand case. If $f(n) = p$, we are in the middle case. If $f(n) = r < p$, let $u'_2 = M_1 \bullet \ldots M_r$ and $u'_1 = M_{r+1} \bullet \ldots \bullet M_p$. Lemma 4.14 provides $\beta : w \to u'_1 \bullet u$ and $\gamma : v \to u'_2$ such that $\gamma \bullet \beta = \alpha$. Applying (P7) to β gives a decomposition of α as on the left below



with α_4 and α_5 isos. If $|w_2| \neq 0$, then we are in the left-hand case of the proposition, and the middle case is impossible by essential uniqueness in (P7). Otherwise, we may decompose α_4 as on the right by atomicity (empty cells are given by coherence or (P4)), so we are in the middle case of the proposition. □

Lastly, we need a few more definitions before Proposition 4.27.

Definition 4.26. Let $\mathbb{E}^{\mathbb{V}}$ be the full subcategory of $\mathbb{E}$ consisting of views.

Consider, for any X, the comma category $\mathbb{E}_X$ induced by the vertical codomain functor $\mathrm{cod}\colon \mathbb{E} \to \mathbb{D}_h$ mapping (4.1) to k (following notation from Section 2.1). Similarly, consider $\mathbb{E}^{\mathbb{V}}_X$. Concretely, an object of $\mathbb{E}^{\mathbb{V}}_X$ is a pair of a view $v\colon d' \rightarrowtail\!\!\!\!\bullet\, d$, and a player $x\colon d \to X$ of X. A morphism $(v_1, x_1) \to (v_2, x_2)$ is a morphism $(w, \alpha)\colon v_1 \to v_2$ in $\mathbb{E}^{\mathbb{V}}$, such that $x_2 \circ \mathrm{cod}(\alpha) = x_1$.

Recall now from above Definition 4.6 the pullback category $\mathbb{E}(X)$. It is isomorphic to the full subcategory of $\mathbb{E}_X$ consisting of pairs (u, x) where $x = id_X$. Similarly, we have $\mathbb{E}^{\mathbb{V}}(X)$, which is empty unless X is an individual.

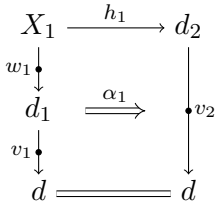
Proposition 4.27. *We have*

(i) The inclusion $\mathbb{E}^{\mathbb{V}}(d) \hookrightarrow \mathbb{E}^{\mathbb{V}}_d$ mapping v to (v, id_d) is an isomorphism of categories.

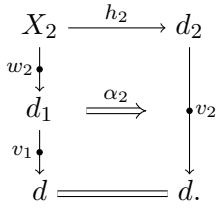
(ii) The inclusion $\sum_{(d,x)\in \mathrm{Pl}(X)} \mathbb{E}^{\mathbb{V}}(d) \hookrightarrow \mathbb{E}^{\mathbb{V}}_X$ mapping $((d, x), v)$ to (v, x) is an isomorphism of categories.

(iii) $\mathbb{E}^{\mathbb{V}}(d)$ is a preorder.

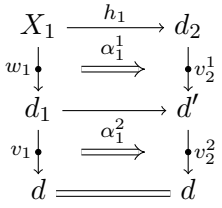
Proof. First, because $\mathbb{I}$ is discrete, $\mathbb{D}_h(d, d) = \{id_d\}$, hence *(i)*. For *(ii)*, the functor $\mathbb{E}^{\mathbb{V}}_X \to \sum_{(d,x)\in \mathrm{Pl}(X)} \mathbb{E}^{\mathbb{V}}(d)$ mapping any (v, x) to $((d, x), v)$, with $v\colon d' \rightarrowtail\!\!\!\!\bullet\, d$ a view and $x\colon d \to X$ a player, is inverse to the given functor. Finally, consider any two morphisms $v_1 \to v_2$ in $\mathbb{E}^{\mathbb{V}}(d)$, say



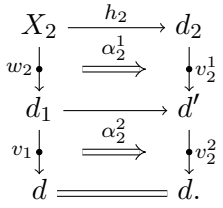
and



Fixing decompositions of v_1 and v_2 into basic moves, we obtain by Lemmas 4.14 and 4.22 that α_1 and α_2 respectively decompose as



and



By Corollary 4.23, $\alpha_1^2 = \alpha_2^2$. Furthermore, we conclude by (P1) and the quotienting (4.2) in the definition of $\mathbb{E}$ that both morphisms are equal in $\mathbb{E}^{\mathbb{V}}(d)$ to $\alpha_1^2 \bullet id_{v_2^1}$. □

4.5. **From behaviours to strategies.**

Definition 4.28. The category S_X of *strategies* on X is the category $\widehat{\mathbb{E}^{\mathbb{V}}_X}$ of presheaves of finite ordinals on $\mathbb{E}^{\mathbb{V}}_X$.

Example 4.29. On $\mathbb{D}^{CCS}$, $\mathbb{E}^{\mathbb{V}}_X$ as defined here yields a category equivalent to the definition in HP, so the categories of strategies are also equivalent (even isomorphic because ford contains no non-trivial automorphism).

The rest of this section develops some structure on strategies, which is needed for constructing the LTS in Section 5.2. We start by extending the assignment $X \mapsto \mathsf{S}_X$ to a pseudo double functor $\mathbb{D}^{op} \to \mathbb{Q}\mathsf{Cat}$, where $\mathbb{Q}\mathsf{Cat}$ is Ehresmann's double category of *quintets* on the 2-category Cat:

Definition 4.30. $\mathbb{Q}\mathsf{Cat}$ has small categories as objects, functors as both horizontal and vertical morphisms, and natural transformations as double cells.

Actually, our first step is to extend the assignment $X \mapsto \mathbb{E}^{\mathbb{V}}_X$ to pseudo double functor $\mathbb{D} \to \mathbb{Q}\mathsf{Cat}$. Define the action of a horizontal map $h\colon X \to X'$ to map any object (v, x) of $\mathbb{E}^{\mathbb{V}}_X$ to $(v, h \circ x)$, and any morphism to itself viewed as a morphism in $\mathbb{E}^{\mathbb{V}}_{X'}$. (This functor is induced by universal property of $\mathbb{E}^{\mathbb{V}}_X$ as a comma category.) This defines a functor $\mathbb{E}^{\mathbb{V}}_-\colon \mathbb{D}_h \to \mathsf{Cat}$. The pseudo functor $\mathbb{D}_v \to \mathsf{Cat}$ is a bit harder to construct. For any $u\colon Y \nrightarrow X$ in $\mathbb{D}_v$ and $y\colon d \to Y$, the cell $\alpha^{y,u}$ from Proposition 4.24 induces a functor $\Sigma_{v^{y,u}}\colon \mathbb{E}^{\mathbb{V}}(d) \to \mathbb{E}^{\mathbb{V}}(d^{y,u})$ mapping any $v\colon d' \nrightarrow d$ to $v^{y,u} \bullet v$. Composing with the coproduct injection $inj_{d^{y,u},y^u}\colon \mathbb{E}^{\mathbb{V}}(d^{y,u}) \hookrightarrow \sum_{(d'',x)\in \mathrm{Pl}(X)} \mathbb{E}^{\mathbb{V}}(d'')$, because $\mathbb{E}^{\mathbb{V}}_X \cong \sum_{(d'',x)\in \mathrm{Pl}(X)} \mathbb{E}^{\mathbb{V}}(d'')$, we obtain functors

$$\mathbb{E}^{\mathbb{V}}(d) \xrightarrow{\Sigma_{v^{y,u}}} \mathbb{E}^{\mathbb{V}}(d^{y,u}) \xhookrightarrow{inj_{d^{y,u},y^u}} \mathbb{E}^{\mathbb{V}}_X,$$

whose copairing defines a functor $\mathbb{E}^{\mathbb{V}}_u\colon \mathbb{E}^{\mathbb{V}}_Y \to \mathbb{E}^{\mathbb{V}}_X$.

Now, for any cell as on the left below, we obtain by Proposition 4.24 a canonical natural isomorphism as on the right

$$\begin{array}{ccc} Y & \xrightarrow{k} & Y' \\ {\scriptstyle u}\downarrow\bullet & \overset{\alpha}{\Longrightarrow} & \bullet\downarrow{\scriptstyle u'} \\ X & \xrightarrow[h]{} & X' \end{array} \qquad\qquad \begin{array}{ccc} \mathbb{E}^{\mathbb{V}}_Y & \xrightarrow{\mathbb{E}^{\mathbb{V}}_k} & \mathbb{E}^{\mathbb{V}}_{Y'} \\ {\scriptstyle \mathbb{E}^{\mathbb{V}}_u}\downarrow & \overset{\cong}{\Longrightarrow} & \downarrow{\scriptstyle \mathbb{E}^{\mathbb{V}}_{u'}} \\ \mathbb{E}^{\mathbb{V}}_X & \xrightarrow[\mathbb{E}^{\mathbb{V}}_h]{} & \mathbb{E}^{\mathbb{V}}_{X'}. \end{array}$$

By canonicity of the above double cell, we have

Proposition 4.31. *This assignment defines a pseudo double functor* $\mathbb{E}^{\mathbb{V}}_-\colon \mathbb{D} \to \mathbb{Q}\mathsf{Cat}$.

Definition 4.32. Let the *opposite* $\mathbb{D}^{op}$ of a pseudo double category $\mathbb{D}$ be obtained by reversing both vertical and horizontal arrows, and hence double cells.

We obtain:

Definition 4.33. Let $\mathsf{S}\colon \mathbb{D}^{op} \to \mathbb{Q}\mathsf{Cat}$ be the composite $\mathbb{D}^{op} \xrightarrow{(\mathbb{E}^{\mathbb{V}}_-)^{op}} \mathbb{Q}\mathsf{Cat}^{op} \xrightarrow{\sim} \mathbb{Q}\mathsf{Cat}$.

As a shorthand, we denote $\mathsf{S}(f)(S)$ by $S \cdot f$ for f horizontal or vertical. Concretely, for any horizontal $h\colon Z \to X$, $S \cdot h$ satisfies

$$(S \cdot h)(v, z) = S(v, h \circ z),$$

whereas for any vertical $u\colon Y \nrightarrow X$, $S \cdot u$ satisfies

$$(S \cdot u)(v, y) = S(v^{y,u} \bullet v, y^u).$$

We conclude this section by constructing the *extension* functor from strategies to behaviours, in arbitrary playgrounds.

Recall that strategies on a position X are presheaves of finite ordinals on $\mathbb{E}^{\mathbb{V}}_X$, and that behaviours are presheaves of finite sets on $\mathbb{E}(X)$. To go from the former to the latter, we use $\mathbb{E}_X$ as a bridge. Recall from Section 2.1 that objects of $\mathbb{E}^{\mathbb{V}}_X$ are diagrams of the shape $d' \overset{v}{\bullet\!\!\to} d \overset{x}{\to} X$, with v a view, and that objects of $\mathbb{E}(X)$ are just plays $Y \overset{u}{\bullet\!\!\to} X$. The idea here is that on the one hand $\mathbb{E}^{\mathbb{V}}_X$ is richer than $\mathbb{E}(X)$, in that its objects may be plays on *subpositions* of X, whereas objects of $\mathbb{E}(X)$ are plays on the whole of X. But on the other hand, $\mathbb{E}(X)$ is richer than $\mathbb{E}^{\mathbb{V}}_X$ because its objects may be arbitrary plays, whereas objects of $\mathbb{E}^{\mathbb{V}}_X$ have to be views. $\mathbb{E}_X$ contains both $\mathbb{E}^{\mathbb{V}}_X$ and $\mathbb{E}(X)$, its objects being diagrams $Y \overset{u}{\bullet\!\!\to} Z \overset{h}{\to} X$, for arbitrary plays u.

First, let $k_X : \widehat{\mathbb{E}^{\mathbb{V}}_X} \to \overline{\mathbb{E}^{\mathbb{V}}_X}$ denote postcomposition with $\mathsf{ford} \hookrightarrow \mathsf{set}$. Because views form a full subcategory of $\mathbb{D}_H$, all embeddings $i_X : \mathbb{E}^{\mathbb{V}}_X \hookrightarrow \mathbb{E}_X$ are also full. This entails:

Lemma 4.34. *For all X, right Kan extension $(i^{op}_X)_\star : \overline{\mathbb{E}^{\mathbb{V}}_X} \hookrightarrow \overline{\mathbb{E}_X}$ along i^{op}_X is well-defined, full, and faithful.*

Proof. One easily shows that, when defined, right extension along a full and faithful functor is full and faithful.

It remains to show that the considered right extensions exist. It is well-known [38] that the right Kan extension of any $S \in \overline{\mathbb{E}^{\mathbb{V}}_X}$ maps any (u,h) to the limit of the functor $(\mathbb{E}^{\mathbb{V}}_X/(u,h))^{op} \to (\mathbb{E}^{\mathbb{V}}_X)^{op} \overset{S}{\to} \mathsf{set}$, if the latter exists. Since finite limits exist in set (though not in ford, which explains why we use set instead of ford for extending strategies), it is enough to prove that each $\mathbb{E}^{\mathbb{V}}_X/(u,h)$ is essentially finite, i.e., equivalent to a finite category. This is proved in the next lemma. □

Lemma 4.35. *For any play $u : Z \bullet\!\!\to Y$ and horizontal $h : Y \to X$, the category $\mathbb{E}^{\mathbb{V}}_X/(u,h)$ is essentially finite.*

For this lemma to hold, we need more axioms.

Axiom. (P9) (Finiteness) For any position X, there are only finitely many players, i.e., the category $\mathbb{I}/X$ is finite.

Proof of Lemma 4.35. Let us fix a pair (u,h). By Proposition 4.27, $\mathbb{E}^{\mathbb{V}}_X/(u,h)$ is a preorder, so we just need to prove that its object set is essentially finite. Now, letting $n = |u|$, we fix a decomposition of u into moves, say $Z = Y_n \overset{M_n}{\bullet\!\!\to} Y_{n-1} \dots Y_1 \overset{M_1}{\bullet\!\!\to} Y_0$. For any morphism $\alpha : (v,x) \to (u,h)$ in $\mathbb{E}_X$, by Lemma 4.14, $m = |v|$ may not exceed n. Furthermore, by Lemma 4.14, Proposition 4.24, and our quotienting (4.2), any such α is determined up to isomorphism by m, a strictly monotone map $f : m \to n$, and a player y of $Y_{f(m)}$. Because such triples (m,f,y) are in finite number, $\mathbb{E}^{\mathbb{V}}_X/(u,h)$ is essentially finite. □

This concludes the proof of Lemma 4.34: right Kan extension along $i^{op}_X : (\mathbb{E}^{\mathbb{V}}_X)^{op} \hookrightarrow \mathbb{E}^{op}_X$ yields a full and faithful functor. We now design the second half of our bridge from $\mathbb{E}^{\mathbb{V}}_X$ to $\mathbb{E}(X)$ via $\mathbb{E}_X$. Consider the embedding $j_X : \mathbb{E}(X) \hookrightarrow \mathbb{E}_X$ mapping any u to (u, id_X). Restriction along $(j_X)^{op}$ defines a functor $\Delta_{(j_X)^{op}} : \overline{\mathbb{E}_X} \to \overline{\mathbb{E}(X)}$.

Recall from Definition 4.6 the notion of behaviour.

Definition 4.36. For any X, let the *extension* functor $\mathsf{ext}_X : \mathsf{S}_X \to \mathsf{B}_X$ be the composite

$$\widehat{\mathbb{E}^{\mathbb{V}}_X} \xrightarrow{k_X} \overline{\mathbb{E}^{\mathbb{V}}_X} \xrightarrow{(i^{op}_X)_\star} \overline{\mathbb{E}_X} \xrightarrow{\Delta_{(j_X)^{op}}} \overline{\mathbb{E}(X)}.$$

We call a behaviour on X *innocent* when it is in the essential image of ext_X.

Notation: when X is clear from context, we abbreviate $\mathsf{ext}_X(S)$ as $\overline{S}$.

Remark 4.37. The calculations of Section 3.4.2 carry over unchanged to the new setting.

Finally, the definitions of Section 3.5 apply more or less verbatim to the playground $\mathbb{D}^{CCS}$, yielding a semantic fair testing equivalence which coincides with that of HP.

5. Playgrounds: transition systems

In the previous section, we have defined behaviours and strategies, and constructed the extension functor from the former to the latter. In this section, we first build on this to state decomposition theorems, which lead to a syntax and an LTS for strategies. Then, we define our second LTS, and relate the two by a strong, functional bisimulation.

5.1. **A syntax for strategies.** Let us begin by proving in the abstract setting of playgrounds analogues of the decomposition results of HP, in particular that strategies form a terminal coalgebra for a certain polynomial functor. This is equivalent to saying that they are essentially infinite terms in a typed grammar. We use this in the next section to define our LTS $\mathcal{S}_{\mathbb{D}}$, and study transitions therein.

First, we have *spatial* decomposition:

Proposition 5.1. *The functor* $\mathsf{S}_X \to \prod_{(d,x)\in \mathrm{Pl}(X)} \mathsf{S}_d$ *given at* (d,x) *by* $\mathsf{S}(x)\colon \mathsf{S}_X \to \mathsf{S}_d$ *is an isomorphism of categories.*

Proof. We have:

$$\begin{array}{rcll} \mathsf{S}_X & = & \mathsf{Cat}((\mathbb{E}^{\mathbb{V}}_X)^{op}, \mathsf{ford}) & \\ & \cong & \mathsf{Cat}(\sum_{(d,x)\in \mathrm{Pl}(X)} \mathbb{E}^{\mathbb{V}}(d)^{op}, \mathsf{ford}) & \text{(by Proposition 4.27)} \\ & \cong & \prod_{(d,x)\in \mathrm{Pl}(X)} \mathsf{Cat}(\mathbb{E}^{\mathbb{V}}(d)^{op}, \mathsf{ford}) & \\ & = & \prod_{(d,x)\in \mathrm{Pl}(X)} \mathsf{S}_d. & \end{array}$$

□

For any $S \in \mathsf{S}_X$, let $S \cdot x$ denote the strategy on d corresponding to (d,x) accross the isomorphism.

The second decomposition result is less straightforward, but goes through essentially as in the concrete case. Let us be a bit more formal here than in Section 3.4.3, by showing that strategies form a terminal coalgebra for some endofunctor on $\mathsf{Set}^{\mathbb{I}}$. We start by defining the relevant endofunctor.

Definition 5.2. Let $[\mathbb{B}]_d$ denotes the set of all isomorphism classes of basic moves from d (i.e., with vertical codomain d).

Definition 5.3. Let $G\colon \mathsf{Set}^{\mathbb{I}} \to \mathsf{Set}^{\mathbb{I}}$ be the functor mapping any family U to

$$(G(U))_d = \left(\prod_{b\in[\mathbb{B}]_d} U_{\mathrm{dom}(b)} \right)^{\star},$$

where $(-)^{\star}$ denotes finite sequences.

Remark 5.4. This functor is polynomial in the sense of Kock [31], as

$$(G(U))_d = \sum_{n\in\mathbb{N}} \left(\prod_{i\in n, b\in[\mathbb{B}]_d} U_{\mathrm{dom}(b)} \right).$$

We now show that strategies, viewed as the $\mathbb{I}$-indexed family $(\mathrm{ob}(\mathsf{S}_d))_{d\in\mathbb{I}}$, form a terminal G-coalgebra. We drop the $\mathrm{ob}(-)$ for readability.

Definition 5.5. For any $S \in \mathsf{S}_d$ and $\sigma \in S(id_d^\bullet)$, let the *restriction* $S_{|\sigma} \in \mathsf{S}_d$ of S to σ be defined by the fact that $S_{|\sigma}(v) = \{\sigma' \in S(v) \mid S(!_v)(\sigma') = \sigma\}$.

(Here, we freely use the isomorphism $\mathbb{E}_d^{\mathbb{V}} \cong \mathbb{E}^{\mathbb{V}}(d)$ from Proposition 4.27, and let $!_v$ denote the unique morphism $id_d^\bullet \to v$ in $\mathbb{E}^{\mathbb{V}}(d)$.)

In view of Remark 5.4, $(G(\mathsf{S}))_d = \sum_{n\in\mathbb{N}} \left(\prod_{b\in[\mathbb{B}]_d} \mathsf{S}_{\mathrm{dom}(b)}\right)^n$. We thus may define the G-coalgebra structure $\partial\colon \mathsf{S} \to G(\mathsf{S})$ in $\mathsf{Set}^{\mathbb{I}}$ of strategies as follows.

Definition 5.6. Let, for all $d \in \mathbb{I}$, $\partial_d\colon \mathsf{S}_d \to \sum_n (\prod_{b\in[\mathbb{B}]_d} \mathsf{S}_{\mathrm{dom}(b)})^n$ send any $S \in \mathsf{S}_d$ to $n = S(id_d^\bullet)$ and the map

$$\begin{array}{rcl} S(id_d^\bullet) & \to & \prod_{b\in[\mathbb{B}]_d} \mathsf{S}_{\mathrm{dom}(b)} \\ \sigma & \mapsto & b \mapsto (S_{|\sigma}) \cdot b. \end{array}$$

Here, we view the ordinal $S(id_d^\bullet)$ as a natural number, and the given map $S(id_d^\bullet) \to \prod_{b\in[\mathbb{B}]_d} \mathsf{S}_{\mathrm{dom}(b)}$ as a list of elements of $\prod_{b\in[\mathbb{B}]_d} \mathsf{S}_{\mathrm{dom}(b)}$. We further use the action of b on S, as below Definition 4.33. We have:

Theorem 5.7. *The map $\partial\colon \mathsf{S} \to G(\mathsf{S})$ makes S into a terminal G-coalgebra.*

This intuitively means that strategies, on individuals, are infinite terms for the following typed grammar with judgements $d \vdash_{\mathsf{D}} D$ and $d \vdash S$, where D is a *definite strategy* and S is a *strategy*

$$\frac{\ldots\ d' \vdash S_b\ \ldots\ (\forall b\colon d' \bullet\!\!\to d \in [\mathbb{B}]_d)}{d \vdash_{\mathsf{D}} \langle (S_b)_{b\in[\mathbb{B}]_d} \rangle} \qquad \frac{\ldots\ d \vdash_{\mathsf{D}} D_i\ \ldots\ (\forall i \in n)}{d \vdash \bigoplus_{i\in n} D_i}\ (n \in \mathbb{N}).$$

Semantically, definite strategies correspond to strategies S such that $S(id_d^\bullet) = 1$, which will play a crucial role in the LTS below.

The rest of this section is a proof of Theorem 5.7.

First of all, we construct an inverse to ∂.

Definition 5.8. Consider $B = (B_1, \ldots, B_n) \in (G(\mathsf{S}))_d$. For any view $v\colon d' \bullet\!\!\to d$, define $\partial'(B) \in \mathsf{S}_d$ by

$$\partial'(B)(v) = \begin{cases} n & \text{if } v = id_d^\bullet \\ \sum_{i\in n} B_i(b)(v') & \text{if } v = b \bullet v', \end{cases}$$

and on morphisms

$$\partial'(B)(v \xrightarrow{(w,\alpha)} v')(\sigma) = \begin{cases} i & \text{if } v' = id_d^\bullet \text{ and } \sigma = i \in n \\ & \text{or if } v = id_d^\bullet \text{ and } \sigma = (i, x) \\ (i, B_i(b)(w, \alpha_1)(x)) & \text{if } v = b \bullet v_1 \text{ and } \sigma = (i, x), \end{cases}$$

where in the last clause necessarily $v' \cong b' \bullet v_1'$ and Lemma 4.14 yields $\alpha_b\colon b \xrightarrow{\cong} b'$ and $\alpha_1\colon v_1 \bullet w \to v_1'$ such that $\alpha_b \bullet \alpha_1 = \alpha$.

Lemma 5.9. *We have $\partial' = \partial^{-1}$.*

Proof. Starting from a strategy $S \in \mathsf{S}_d$, let $n = S(id_d^\bullet)$, and $B_i(b) = (S_{|i}) \cdot b$, for any $d' \xrightarrow{b} d$. We have $\partial S = (B_1, \dots, B_n)$, and thus $\partial'(\partial S)(v) = n$ if $v = id_d^\bullet$, and $\partial'(\partial S)(v) = \sum_{i \in n}(S_{|i})(b \bullet v') = S(v)$ if $v = b \bullet v'$, as desired.

Conversely, starting from $B = (B_1, \dots, B_n) \in (G(\mathsf{S}))_d$, let $S = \partial' B$. We have that ∂S has length n, and its ith component maps any $b : d' \to d$ to the strategy mapping any $v' : d'' \to d'$ to the strategy $(S_{|i}) \cdot b$. Thus, $(\partial S)_i(b)(v') = ((S_{|i}) \cdot b)(v') = (S_{|i})(b \bullet v')$. But by definition, this is equal to $B_i(b)(v')$, as desired. □

Consider any G-coalgebra $a : U \to GU$.

We define by induction on N a sequence of maps $f_N : U \to \mathsf{S}$, such that for any d and $u \in U_d$ the $f_{n+N}(u)$'s agree on views of length $\leq n$. I.e., for any $d \in \mathbb{I}$, $u \in U_d$, view v of length less than n, and any N, $f_{n+N}(u)(v) = f_n(u)(v)$, and similarly the action of $f_{n+N}(u)$ on morphisms between such views is the same as that of $f_n(u)$.

To start the induction, take $f_0(u)$ to be the strategy mapping $id_d^\bullet$ to $|a(u)|$, i.e., the length of $a(u) \in (\prod_b U_{\mathrm{dom}(b)})^\star$, and all other views to 0.

Furthermore, given f_N, define f_{N+1} to be

$$U \xrightarrow{a} GU \xrightarrow{G(f_N)} G(\mathsf{S}) \xrightarrow{\partial^{-1}} \mathsf{S}.$$

In other words, f_N is

$$U \xrightarrow{a} GU \xrightarrow{G(a)} \dots G^{N-1}U \xrightarrow{G^{N-1}a} G^N U \xrightarrow{G^N f_0} G^N \mathsf{S} \xrightarrow{G^{N-1}(\partial^{-1})} G^{N-1}(\mathsf{S}) \dots G(\mathsf{S}) \xrightarrow{\partial^{-1}} \mathsf{S}.$$

Unfolding the definitions yields:

Lemma 5.10. *Consider any $u \in U_d$, and let $a(u) = (z_1, \dots, z_k)$. For any $f : U \to \mathsf{S}$, we have*

- $\partial^{-1}(G(f)(a(u)))(id_d^\bullet) = k$, *and*
- $\partial^{-1}(G(f)(a(u)))(b \bullet v) = \sum_{i \in k} f(z_i(b))(v)$ *for any composable basic move b and view v.*

Corollary 5.11. *We have, for any $N \in \mathbb{N}$, $f_N(u)(id_d^\bullet) = k$. Furthermore, for any basic move $b : d' \to d$, and view $v : d'' \to d'$, we have for any $N \in \mathbb{N}$:*

$$f_{N+1}(u)(b \bullet v) = \sum_{i \in k} f_N(z_i(b))(v).$$

As announced, we have:

Lemma 5.12. *For any view $v : d' \to d$ and $n \in \mathbb{N}$, $f_{|v|+n}(u)(v) = f_{|v|}(u)(v)$.*

Proof. We proceed by well-founded induction on $(|v|, n)$, for the lexical ordering. Let again $a(u) = (z_1, \dots, z_k)$. First, we have $f_{|id^\bullet|}(u)(id^\bullet) = k$, and for any n, $f_{|id^\bullet|+n+1}(u)(id^\bullet) = k$ by Corollary 5.11. Now, if $v = b \bullet v'$, then by Corollary 5.11 again:

$$\begin{array}{rcll} f_{|v|+n+1}(u)(b \bullet v') & = & \sum_{i \in k} f_{|v|+n}(z_i(b))(v') & \\ & = & \sum_{i \in k} f_{|v'|+n+1}(z_i(b))(v') & (\text{by } |v| = |v'| + 1) \\ & = & \sum_{i \in k} f_{|v'|}(z_i(b))(v') & (\text{by induction hypothesis}) \\ & = & f_{|v|}(u)(b \bullet v') & (\text{by Corollary 5.11 again}). \end{array}$$

□

The sequence $(f_n(u))$ thus has a colimit in $\mathsf{S}_d = \widehat{\mathbb{E}^{\mathbb{V}}_d}$: the presheaf mapping any view v to $f_{|v|}(u)(v)$. This allows us to define:

Definition 5.13. Let $f : U \to \mathsf{S}$ map any $u \in U_d$ to the colimit of the $f_N(u)$'s.

Lemma 5.14. *The following diagram commutes:*

$$\begin{array}{ccc} U & \xrightarrow{\quad a \quad} & GU \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle G(f)} \\ \mathsf{S} & \xleftarrow{\partial^{-1}} & G(\mathsf{S}). \end{array}$$

Proof. Consider any $u \in U_d$ and view v, and let $a(u) = (z_1, \dots, z_k)$. Let also $n = f(u)(v) = f_{|v|}(u)(v)$ and $n' = \partial^{-1}(G(f)(a(u)))(v)$.

- If $|v| = 0$, then by Lemma 5.10 $n = n' = k$.
- If $v = b \bullet v'$, then by Lemma 5.10 again we have $n' = \sum_{i \in k} f(z_i(b))(v')$. But by definition of f, we obtain $n' = \sum_{i \in k} f_{|v'|}(z_i(b))(v')$, which is in turn equal to $f_{|v|}(u)(v) = n$ by Corollary 5.11. □

Corollary 5.15. *The map f is a map $U \to \mathsf{S}$ of G-coalgebras.*

Lemma 5.16. *The map f is the unique map $U \to \mathsf{S}$ of G-coalgebras.*

Proof. Consider any such map g of coalgebras, and let $a(u) = (z_1, \dots, z_k)$. The map g must be such that

$$g(u)(id^{\bullet}_d) = \partial^{-1}(G(g)(a(u)))(id^{\bullet}_d) = k,$$

by Lemma 5.10. Furthermore, by the same lemma, it must satisfy:

$$g(u)(b \bullet v) = \partial^{-1}(G(g)(a(u)))(b \bullet v) = \sum_{i \in k} g(z_i(b))(v),$$

which imposes by induction that $f = g$. □

The last two results directly entail Theorem 5.7.

5.2. **The labelled transition system for strategies.** In this section, we go beyond HP, and define an LTS for strategies, for an arbitrary playground $\mathbb{D}$.

First, the alphabet for our LTS will constist of quasi-moves, in the following sense.

Notation 5.17. We use the following notation for cartesian lifting (by (P1)) of a play u along a horizontal morphism k (fixing a global choice of liftings):

$$\begin{array}{ccc} D_{k,u} & \xrightarrow{h_{k,u}} & X' \\ {\scriptstyle u_{|k}}\downarrow\bullet & \overset{\alpha_{k,u}}{\Longrightarrow} & \bullet\downarrow{\scriptstyle u} \\ Y & \xrightarrow[k]{} & X. \end{array}$$

Definition 5.18. A *quasi-move* is a vertical morphism which locally either is a move or has length 0. More precisely, a play $u : Y \nrightarrow X$ is a quasi-move iff for all players $x : d \to X$, $u_{|x}$ either is a move or has length 0.

A quasi-move is *full* when it locally either is a full move or has length 0. Let $\mathbb{Q}$ denote the subgraph of $\mathbb{D}_v$ consisting of full quasi-moves.

Observe that a quasi-move on an individual either is a move or has length 0.

States in our LTS will be the following special kind of strategies:

Definition 5.19. A strategy $S \in \mathsf{S}_X$ is *definite* when $\overline{S}(id^\bullet_X) = 1$, or equivalently when for all players $(d,x) \in \mathrm{Pl}(X)$, we have $S(id^\bullet_d, x) = 1$.

Intuitively, for any quasi-move $X' \xrightarrow{M}\!\!\!\!\bullet\; X$, we would like transitions $(X',S') \xrightarrow{M} (X,S)$ in our LTS to occur when S' is a definite restriction of $S \cdot M$ to some state of $\overline{S}(M)$. I.e., a transition roughly corresponds to a way for $\overline{S}$ to accept M. However, $S \cdot M$ is not quite $\overline{S}(M)$ so the right notion of restriction may not be obvious. But we have defined a notion of restriction in Definition 5.5, for strategies on individuals. We now define restriction for general strategies, and use this to define our LTS. Finally, we elucidate the connection with $\overline{S}(M)$.

Consider, for any $S \in \mathsf{S}_X$ and $\sigma \in \prod_{(d,x)\in\mathrm{Pl}(X)} S(id^\bullet_d, x)$, and recall from below Definition 4.33 that $S \cdot h$ is shorthand for the image of $S \in \mathsf{S}_X$ under the action of a horizontal morphism $h : Y \to X$ for the horizontal part of our pseudo double functor S.

Definition 5.20. Let the *restriction* $S_{|\sigma} \in \mathsf{S}_X$ of S to σ be defined by the fact that for any player $x : d \to X$,

$$(S_{|\sigma}) \cdot x = (S \cdot x)_{|\sigma(d,x)}.$$

Concretely, we have, for any v, $S_{|\sigma}(v,x) = \{\sigma' \in S(v,x) \mid S(!_v)(\sigma') = \sigma(d,x)\}$, where $!_v$ is the unique morphism $(id^\bullet_d, x) \to (v,x)$ in $\mathbb{E}^{\mathbb{V}}_X$.

We now define our LTS for strategies over $\mathbb{Q}$.

Definition 5.21. The underlying graph $\mathcal{S}_{\mathbb{D}}$ for our LTS is the graph with as vertices all pairs (X,S) where X is a position and $S \in \mathsf{S}_X$ is a definite strategy, and whose edges $(X',S') \to (X,S)$ are all full quasi-moves $M : X' \xrightarrow{}\!\!\!\!\bullet\; X$ such that there exists a state $\sigma \in \prod_{(d',x')\in\mathrm{Pl}(X')}(S \cdot M)(id^\bullet_{d'}, x')$ with $S' = (S \cdot M)_{|\sigma}$.

The assignment $(X,S) \mapsto X$ defines a morphism $p_{\mathsf{S}} : \mathcal{S}_{\mathbb{D}} \to \mathbb{Q}$ of reflexive graphs, which is our LTS.

An alternative characterisation of transitions $(X,S) \xleftarrow{M} (X',S')$ is the existence of σ such that

$$S' \cdot x' = (S \cdot M \cdot x')_{|\sigma(d',x')} = (S \cdot (x')^M \cdot v^{x',M})_{|\sigma(d',x')}$$

for all $(d',x') \in \mathrm{Pl}(X')$.

Let us now return to the connection between $\prod_{(d',x')\in\mathrm{Pl}(X')}(S \cdot M)(id^\bullet_{d'}, x')$ and $\overline{S}(M)$. First, we have by definition $(S \cdot M)(id^\bullet_{d'}, x') = S(v^{x',M}, (x')^M)$, for any player $x' : d' \to X'$. Now, as recalled above, $\overline{S}(M)$ may be characterised as a limit of

$$(\mathbb{E}^{\mathbb{V}}_X / M)^{op} \xrightarrow{\mathrm{dom}} (\mathbb{E}^{\mathbb{V}}_X)^{op} \xrightarrow{S} \mathsf{ford} \hookrightarrow \mathsf{set}.$$

Since $\alpha^{x',M} : v^{x',M} \to M$ is an object in $(\mathbb{E}^{\mathbb{V}}_X / M)^{op}$, we obtain by projection a map $\overline{S}(M) \to S(v^{x',M}, (x')^M)$.

Definition 5.22. For any $S \in \mathsf{S}_X$, let $\psi_M : \overline{S}(M) \to \prod_{(d',x')\in\mathrm{Pl}(X')} S(v^{x',M}, (x')^M)$ denote the corresponding tupling map.

Proposition 5.23. *For any definite $S \in \mathsf{S}_X$, the map ψ_M is a bijection.*

We prove this through the following lemma. For any full quasi-move $M : X' \nrightarrow X$, observe that for any player $x' : d' \to X'$, $v^{x',M}$ has length at most 1 (consider $M_{|(x')^M}$), and let

$$\mathrm{Pl}_M(X') = \{(d', x') \in \mathrm{Pl}(X') \mid |v^{x',M}| \neq 0\}.$$

Lemma 5.24. *For any definite $S \in \mathsf{S}_X$, and full quasi-move $M : X' \nrightarrow X$, the map*

$$\overline{S}(M) \xrightarrow{\psi_M} \prod_{(d',x') \in \mathrm{Pl}(X')} S(v^{x',M}, (x')^M) \to \prod_{(d',x') \in \mathrm{Pl}_M(X')} S(v^{x',M}, (x')^M),$$

where the second map is by projection, is bijective.

Proof. Recall that $\overline{S}(M)$ is a limit of

$$(\mathbb{E}^{\mathbb{V}}_X/M)^{op} \xrightarrow{\mathrm{dom}} (\mathbb{E}^{\mathbb{V}}_X)^{op} \xrightarrow{S} \mathsf{ford} \hookrightarrow \mathsf{set},$$

and consider the poset P with underlying set $\mathrm{Pl}(X) + \mathrm{Pl}_M(X')$ and ordering given by $(d, x) < (d', x')$ iff $x = (x')^M$. Consider the functor $p : P \to \mathbb{E}^{\mathbb{V}}_X/M$ mapping any $(d, x) \in \mathrm{Pl}(X)$ to the unique morphism $id^{\bullet}_d \to M$ with lower border x, and any $(d', x') \in \mathrm{Pl}_M(X')$ to $\alpha^{x',M}$. Since P is a poset, p is faithful. It is furthermore full by Proposition 4.27. Finally, for any $(w, \alpha) : v \to M$ in $\mathbb{E}^{\mathbb{V}}_X/M$,

- either $|v| = 0$ and there is a unique player $x : d \to X$ such that (w, α) is the (unique) morphism $id^{\bullet}_d \to M$ with lower border x,
- or $|v| = 1$ and there exists a unique player $(d', x') \in X'$ such that $(w, \alpha) = (id, \alpha^{x',M})$ (let $x = \mathrm{cod}(\alpha)$; $|M_{|x}| = 1$, so by Proposition 4.25 $|w| = 0$).

This entails that p is essentially surjective on objects, hence an equivalence. Thus, $\overline{S}(M)$ is also a limit of

$$P^{op} \simeq (\mathbb{E}^{\mathbb{V}}_X/M)^{op} \xrightarrow{\mathrm{dom}} (\mathbb{E}^{\mathbb{V}}_X)^{op} \xrightarrow{S} \mathsf{ford} \hookrightarrow \mathsf{set}.$$

But now, because S is definite, this functor maps any $(d, x) \in \mathrm{Pl}(X)$ to a singleton, hence $\overline{S}(M)$ is also a limit of

$$\mathrm{Pl}_M(X') \hookrightarrow P^{op} \simeq (\mathbb{E}^{\mathbb{V}}_X/M)^{op} \xrightarrow{\mathrm{dom}} (\mathbb{E}^{\mathbb{V}}_X)^{op} \xrightarrow{S} \mathsf{ford} \hookrightarrow \mathsf{set},$$

i.e., isomorphic to $\prod_{(d',x') \in \mathrm{Pl}_M(X')} S(v^{x',M}, (x')^M)$, as desired. □

Proof of Proposition 5.23. If $|v^{x',M}| = 0$, then $S(v^{x',M}, (x')^M)$ is a singleton. Thus, the second map of Lemma 5.24 is bijective, hence so is ψ_M. □

The moral of Proposition 5.23 is that transitions $(X, S) \xleftarrow{M} (X', S')$ in $\mathcal{S}_{\mathbb{D}}$ are precisely given by full quasi-moves $M : X' \nrightarrow X$ such that there exists a state $\sigma \in \overline{S}(M)$ with

$$S' = (S \cdot M)_{|\psi_M(\sigma)},$$

for all $(d', x') \in \mathrm{Pl}(X')$.

We now give more syntactic characterisations of transitions, starting with transitions from states of the shape (d, S). Recall the syntax for strategies below Theorem 5.7.

Proposition 5.25. *If $S = \langle (S_b)_{b \in [\mathbb{B}]_d} \rangle$ is a definite strategy on $d \in \mathbb{I}$, and if for all $b \in [\mathbb{B}]_d$, $S_b = \bigoplus_{i \in n_b} D^b_i$ for definite D^b_i, then for any $M : X' \nrightarrow d$ we have $(d, S) \xleftarrow{M} (X', S')$ iff*

- *for all $(d', x') \in \mathrm{Pl}_M(X')$, there exists $i_{x'} \in n_{v^{x',M}}$ such that $S' \cdot x' = D^{v^{x',M}}_{i_{x'}}$,*

- *and for all* $(d', x') \in \mathrm{Pl}(X') \setminus \mathrm{Pl}_M(X')$, $S' \cdot x' = S$.

Let us now characterise transitions from arbitrary positions in terms of their restrictions to individuals. Recalling Notation 5.17, we have:

Proposition 5.26. *We have* $(X, S) \xleftarrow{M} (X', S')$ *iff for all* $(d, x) \in \mathrm{Pl}(X)$,

$$(d, S \cdot x) \xleftarrow{M_{|x}} (D_{x,M}, S' \cdot h_{x,M}).$$

Putting both previous results together, we obtain:

Corollary 5.27. *Let, for all* $(d, x) \in \mathrm{Pl}(X)$, $S \cdot x = \langle (S^x_b)_{b \in [\mathbb{B}]_d} \rangle$ *and for all* $b \in [\mathbb{B}]_d$, $S^x_b = \bigoplus_{i \in n^x_b} D^{x,b}_i$ *for definite* $D^{x,b}_i$.

Then, for any $M : X' \nrightarrow X$, *we have* $(X, S) \xleftarrow{M} (X', S')$ *iff*

- *for all* $(d', x') \in \mathrm{Pl}_M(X')$, *there exists* $i_{x'} \in n^{(x')^M}_{v^{x'}, M}$ *such that* $S' \cdot x' = D^{(x')^M, v^{x'}, M}_{i_{x'}}$,
- *and for all* $(d', x') \in \mathrm{Pl}(X') \setminus \mathrm{Pl}_M(X')$, $S' \cdot x' = S \cdot (x')^M$.

5.3. **Process terms.** In the previous section, starting from a playground $\mathbb{D}$, we have constructed an LTS $\mathcal{S}_{\mathbb{D}}$ of strategies. We now begin the construction of the LTS $\mathcal{T}_{\mathbb{D}}$ of *process terms* announced in Section 1.3, starting with process terms themselves.

Definition 5.28. For any X, let $[\mathbb{F}]_X$ be the set of isomorphism classes of full moves with codomain X, in $\mathbb{D}_H(X)$, and let χ denote the map

$$\begin{array}{rcl} [\mathbb{F}]_d & \to & \mathcal{P}_f([\mathbb{B}]_d) \\ M & \mapsto & \{[b] \in [\mathbb{B}]_d \mid \exists \alpha \in \mathbb{D}_H(b, M)\}. \end{array}$$

Let $[\mathbb{F}^1]_d$ denote the subset of $[\mathbb{F}]_d$ consisting of (isomorphism classes of) full moves $M : X' \nrightarrow d$ such that $\mathrm{Pl}_M(X')$ is a singleton (and hence so is $\chi(M)$). Let $[\mathbb{F}^+]_d$ denote the complement subset.

The map χ is easily checked to be well-defined.

We state one more axiom to demand that basic sub-moves of a full move $[M] \in [\mathbb{F}]_d$ may not be sub-moves of other full moves.

Axiom. (P10) (Basic vs. full) For any $d \in \mathbb{I}$ and $M, M' \in [\mathbb{F}]_d$, if $M \neq M'$, then $\chi(M) \cap \chi(M') = \varnothing$.

Let *process terms* be infinite terms in the typed grammar:

$$\frac{\ldots\ d_i \vdash T_i\ \ldots\ (\forall i \in n)}{d \vdash \sum_{i \in n} M_i.T_i}\ (n \in \mathbb{N}; \forall i \in n, M_i \in [\mathbb{F}^1]_d \text{ and } \chi[M_i] = \{b_i : d_i \nrightarrow d\})$$

$$\frac{\ldots\ d' \vdash T_b\ \ldots\ (\forall (b : d' \nrightarrow d) \in \chi[M])}{d \vdash M\langle (T_b)_{b \in \chi[M]} \rangle}\ (M \in [\mathbb{F}^+]_d).$$

The first rule is a guarded sum, in a sense analogous to guarded sum in CCS. It should be noted that guards have to be full moves with only one non-trivial view. There is good reason for that, since allowing general moves as guards would break bisimilarity between process terms and strategies. To understand this, consider a hypothetical guarded sum

$R = (P|Q) + (P'|Q')$. Since this has no interaction before the choice is made, R behaves, in CCS, just like an internal choice $(P|Q) \oplus (P'|Q')$. However, our translation to strategies does not translate guarded sum as an internal choice, with right, since other guarded sums, e.g., $a.P + b.Q$ should certainly not be translated this way. Instead, R would be translated as something equivalent to $(P|Q) \oplus (P'|Q) \oplus (P|Q') \oplus (P'|Q')$, which is clearly not bisimilar to R in general.

We could easily include internal choice in the grammar, since strategies do model it, directly. We refrain from doing so for simplicity.

Definition 5.29. Let $\mathsf{T}_{\mathbb{D}}$ be the set of process terms.

Example 5.30. For $\mathbb{D}^{CCS}$, the obtained syntax is equivalent to

$$\frac{\dots\ \Gamma \cdot \alpha_i \vdash P_i \dots}{\Gamma \vdash \sum_i \alpha_i.P_i} \qquad \frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P|Q}\ .$$

where

- Γ ranges over natural numbers;
- $\alpha ::= a \mid \overline{a} \mid \heartsuit \mid \nu$ (for $a \in \Gamma$);
- $\Gamma \cdot \alpha$ denotes $(\Gamma + 1)$ if $\alpha = \nu$ and just Γ otherwise.

This grammar obviously contains CCS, and we let $\theta : \mathrm{ob}(CCS) \hookrightarrow \mathsf{T}_{\mathbb{D}^{CCS}}$ be the injection.

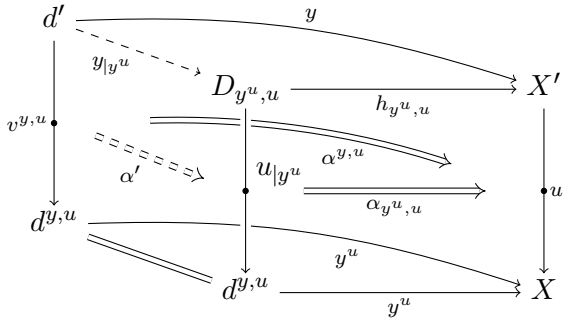
5.4. **The labelled transition system for process terms.** We now define the LTS $\mathcal{T}_{\mathbb{D}}$. States, i.e., vertices of the graph underlying this LTS, are pairs (X, T) of a position X and a family T of process terms, indexed by the players of X, i.e., $T \in \prod_{(d,x)\in \mathrm{Pl}(X)} (\mathsf{T}_{\mathbb{D}})_d$, where $(\mathsf{T}_{\mathbb{D}})_d$ is the set of process terms of type d.

To define edges, we need a lemma. For any play $u : X' \nrightarrow X$ and $x : d \to X$, recalling Notation 5.17, consider the map

$$\begin{array}{rcccc} r^u & : & \sum_{(d,x)\in \mathrm{Pl}(X)} \mathrm{Pl}(D_{x,u}) & \to & \mathrm{Pl}(X') \\ & & ((d,x),(d',x')) & \mapsto & h_{x,u} \circ x' \end{array}$$

sending any $(d, x) \in \mathrm{Pl}(X)$ and $x' : d' \to D_{x,u}$ to $d' \xrightarrow{x'} D_{x,u} \xrightarrow{h_{x,u}} X'$.

Consider also the map i^u in the other direction sending any $y : d' \to X'$ to the pair $((d^{y,u}, y^u), (d', y_{|y^u}))$, where $y_{|y^u}$ is the (domain in $\mathbb{D}_V$ of the) unique α' making the diagram



commute (by (P1)). This map i^u is well-defined by uniqueness of y^u and cartesianness of $\alpha_{y^u,u}$.

Lemma 5.31. *The maps i^u are r^u are mutually inverse.*

Proof. Straightforward. □

Let us return to the definition of our LTS. We first say that for any full quasi-move $M : D \to\!\!\!\!\bullet\, d$, a process term $d \vdash T$ has an M-transition to (D, T'), for $T' \in \prod_{(d',x') \in \mathrm{Pl}(D)} (\mathsf{T}_{\mathbb{D}})_{d'}$, when one of the following holds:

(i) $\exists M' \in [\mathbb{F}^+]$, $T = M'\langle T'' \rangle$, and, for all $(d', x') \in \mathrm{Pl}(D)$,
- if $v^{x',M}$ is a basic move, then $v^{x',M} \in \chi(M')$ and $T'_{d',x'} = T''_{v^{x',M}}$;
- otherwise $|v^{x',M}| = 0$ (hence $d' = d$), and $T'_{d',x'} = T$;

(ii) $[M] \in [\mathbb{F}^1]$, $T = \sum_{i \in n} M_i.T_i$, $M_{i_0} = [M]$ for some $i_0 \in n$, and for all players $x' : d' \to D$
- if $v^{x',M} \in \chi(M)$, then $T'_{d',x'} = T_{i_0}$,
- and otherwise ($|v^{x',M}| = 0$), $T'_{d',x'} = T$;

(iii) $|M| = 0$ and for all $(d', x') \in \mathrm{Pl}(D)$, $T'_{d',x'} = T$ (which, again, makes sense by Lemma 4.17).

We denote such a transition by $T \xleftarrow{M} (D, T')$.

Remark 5.32. The first case *(i)* allows $\chi(M) = \varnothing$, but if $\chi(M) \neq \varnothing$, then $[M] = M'$ by (P10). Also, let us mention that $\chi(M) \neq \varnothing$ does not imply $|M| = 0$ in general, although it does in $\mathbb{D}^{CCS}$.

Definition 5.33. Let $\mathcal{T}_{\mathbb{D}}$ be the graph with pairs (X, T) as vertices, and as edges $(X', T') \to (X, T)$ full quasi-moves $M : X' \to\!\!\!\!\bullet\, X$ such that for all $(d, x) \in \mathrm{Pl}(X)$, $T_{d,x} \xleftarrow{M_{|x}} (D_{x,M}, (T' \circ \Sigma_{h_{x,M}}))$. Here, we let $\Sigma_{h_{x,M}}$ denote composition with $h_{x,M} : D_{x,M} \to X'$, viewed as a map $\mathrm{Pl}(D_{x,M}) \to \mathrm{Pl}(X')$.

$\mathcal{T}_{\mathbb{D}}$ is viewed as an LTS over $\mathbb{Q}$, by mapping $(X, T) \xleftarrow{M} (X', T')$ to $X \xleftarrow{M}\!\!\!\!\bullet\, X'$.

Example 5.34. For $\mathbb{D}^{CCS}$, the obtained LTS differs subtly, but significantly from the usual LTS for CCS. In order to explain this clearly, let us introduce some notation. First, let *evaluation contexts* be generated by the grammar

$$\frac{}{\Gamma; x : n \vdash x(a_1, \ldots, a_n)} \qquad \frac{\Gamma; \Delta_1 \vdash \mathsf{e}_1 \qquad \Gamma; \Delta_2 \vdash \mathsf{e}_2}{\Gamma; \Delta_1, \Delta_2 \vdash \mathsf{e}_1 | \mathsf{e}_2},$$

where, in the first rule, $\forall i \in n, a_i \in \Gamma$, and in the second $\mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) = \varnothing$. Here, x ranges over a fixed set of *variables*, and $\Delta, \ldots$ range over finite maps from variables to natural numbers. Evaluation contexts are considered equivalent up to associativity and commutativity of $|$. Positions are essentially a combinatorial, direct representation of such contexts.

Leaving the details aside, states in $\mathcal{T}_{\mathbb{D}^{CCS}}$ may be viewed as pairs (X, T) of an evaluation context X, plus, for each n-ary variable $x(a_1, \ldots, a_n)$ in X, a process term over n in the grammar of Example 5.30. Instead of separately writing the evaluation context and the map from its variables to process terms, we inline process terms between brackets in the context, thus avoiding variables. Moves are either put in context similarly, or located implicitly. E.g., for a state (X, T) where X contains two players respectively mapped by T to process terms P and Q, we would write $[P]|[Q]$. There is some ambiguity in this notation, e.g., in case some channels are absent from P: are they absent from the arity of P, or only unused? Since we use this notation mostly for clarifying examples, we will avoid such ambiguities.

Finally, we sometimes use brackets to denote the fact that some holes are filled with the given state. E.g., $X[[P]|[Q]]$ denotes a state X, where a hole has been replaced by a parallel composition of two holes, respectively filled with P and Q.

Returning to our comparison of $\mathcal{T}_{\mathbb{D}^{CCS}}$ and *CCS*, of course, a first difference is the fact that labels may contain several moves, as quasi-moves only locally have length 1.

A second difference is the presence of *heating* rules for parallel composition and channel creation, in a sense close to the chemical abstract machine [4]. For example, we have transitions $X[P|Q] \xleftarrow{\pi} X[[P]|[Q]]$.

There is a third important difference, related to channel creation. For instance, we have transitions

$$[\nu a.a.P] \xleftarrow{\nu} [a.P] \xleftarrow{\iota a} [P].$$

The second transition cannot occur in a closed-world setting, since the environment cannot know a. And it does not occur in *CCS* either.

A final difference is that labels contain too much information to be relevant for behavioural equivalences. E.g., they contain the whole evaluation context in which the transition takes place, as well as which players are involved.

The second difference, i.e., the presence of heating rules, is not really problematic, and merely forces us to use weak bisimulations rather than strong ones. All other defects will be corrected below.

5.5. **Translation and a first correctness result.** We conclude this section on the general theory of playgrounds by establishing a strong, functional bisimulation from process terms to strategies.

Mimicking (3.2) (page 23), our translation from process terms to definite strategies (*qua* families over $\mathbb{I}$) is defined coinductively by

$$\begin{array}{rcl} [\![\sum_{i\in n} M_i.T_i]\!] & = & \langle b \mapsto \bigoplus_{\{i\in n | b\in\chi(M_i)\}} [\![T_i]\!]\rangle \\ [\![M\langle (T_b)_{b\in\chi(M)}\rangle]\!] & = & \left\langle b \mapsto \left\{ \begin{array}{ll} [\![T_b]\!] & \text{if } b\in\chi(M) \\ \varnothing & \text{otherwise} \end{array} \right. \right\rangle. \end{array} \tag{5.1}$$

Let us extend the map $[\![-]\!]\colon \mathsf{T}_{\mathbb{D}} \to \mathsf{S}_{\mathbb{D}}$ to a map $[\![-]\!]\colon \mathrm{ob}(\mathcal{T}_{\mathbb{D}}) \to \mathrm{ob}(\mathcal{S}_{\mathbb{D}})$, defined by $[\![X,T]\!] = (X, ([\![T_{d,x}]\!])_{(d,x)\in \mathrm{Pl}(X)})$, using Proposition 5.1.

Theorem 5.35. *The map $[\![-]\!]\colon \mathrm{ob}(\mathcal{T}_{\mathbb{D}}) \to \mathrm{ob}(\mathcal{S}_{\mathbb{D}})$ is a functional, strong bisimulation.*

Proof. The theorem follows from Proposition 5.26 and the next lemma. □

Lemma 5.36. *For any full quasi-move $M\colon X' \rightarrowtail\!\!\!\!\bullet\; d$, for any $T \in (\mathsf{T}_{\mathbb{D}})_d$ and $S' \in (\mathsf{S}_{\mathbb{D}})_{X'}$, we have*

$$(d, [\![T]\!]) \xleftarrow{M} (X', S') \qquad \textit{iff} \qquad \exists T', (T \xleftarrow{M} (X', T')) \wedge ((X', S') = [\![X', T']\!]).$$

Note the implicit typing: $T' \in \prod_{(d',x')\in \mathrm{Pl}(X')} (\mathsf{T}_{\mathbb{D}})_{d'}$. Also the second condition on the right is equivalent to $\forall x'\colon d' \to X', S'\cdot x' = [\![T'_{d',x'}]\!]$.

Proof. If $|M| = 0$, then both sides are equivalent to the fact that for all $x'\colon d' \to X'$, $S'\cdot x' = [\![T]\!]$.

Otherwise, we proceed by case analysis on T.

If $T = M'\langle (T''_b)_{b\in\chi(M')}\rangle$, then by (P10) both sides are equivalent to $\chi(M) \subseteq \chi(M')$, plus

- for all $(d', x') \in \mathrm{Pl}_M(X')$, $S'\cdot x' = [\![T''_{v^{x'},M}]\!]$, and

- for all $(d', x') \in \mathrm{Pl}(X') \setminus \mathrm{Pl}_M(X')$, $S' \cdot x' = [\![T]\!]$.

Indeed, for any $b \in \chi(M)$, $[\![T]\!] \cdot b = [\![T''_b]\!]$ is definite. We thus put $T'_{d',x'} = T''_{v^{x',M}}$ in the first case and $T'_{d',x'} = T$ in the second case.

If $T = \sum_{i \in n} M_i.T_i$, then both sides are equivalent to the existence of $i_0 \in n$ such that $M_{i_0} = [M]$ and

- for the unique $(d', x') \in \mathrm{Pl}_M(X')$, $S' \cdot x' = [\![T_{i_0}]\!]$, and
- for all $(d', x') \in \mathrm{Pl}(X') \setminus \mathrm{Pl}_M(X')$, $S' \cdot x' = [\![T]\!]$.

This uses (P10), since the left-hand side unfolds to the existence of $x' : d' \to X'$ such that $v^{x',M} \in \chi[M]$ and $[\![T]\!] \cdot v^{x',M} \neq \varnothing$, i.e., $v^{x',M} \in \chi(M_{i_0})$ for some $i_0 \in n$, by definition of $[\![T]\!]$. This entails in particular $[M] = M_{i_0}$ by (P10). □

6. Graphs and fair morphisms

In this section, we derive our main result. For this, we develop a notion of *graph with complementarity*, which aims at being a theory of LTSs over which fair testing makes sense. Although the theory would apply with any predicate $\perp$ compatible with $\approx_\Sigma$ equivalence classes (see below), the question of whether such a generalisation would have useful applications is deferred for now.

For any graph with complementarity A and relation $R : G \nrightarrow H$ over A, we exhibit sufficient conditions for R to be *fair*, i.e., to preserve and reflect fair testing equivalence. We then relate this theory to our semantics, and show that it entails our main result. For now, this section lies outside the scope of playground theory. Some aspects of it could be formalised there, but we leave the complete formalisation for further work. Because the only playground involved is $\mathbb{D}^{CCS}$, we often omit sub or superscripts, e.g., in $\mathbb{D}$, $\mathcal{S}_\mathbb{D}$ (even just $\mathcal{S}$), etc.

Before we start, let us define $\mathbb{W}_{CCS}$ to be the set of *closed-world quasi-moves*, i.e., vertical morphisms in $\mathbb{D}$ which either are closed-world moves (Definition 3.22) or have length 0. Please note: quasi-moves must locally restrict to plays of length ≤ 1, whereas closed-world quasi-moves have length ≤ 1 globally. Let $\mathbb{D}^\mathbb{W}$ be the subbicategory of $\mathbb{D}_v$ generated by $\mathbb{W}_{CCS}$, and let Σ be the free reflexive graph on an endo-edge $\heartsuit$. Finally, let $\bar{\ell}_\mathbb{D} : \mathbb{D}^\mathbb{W} \to \mathsf{fc}(\Sigma)$ be the pseudo functor determined by the mapping $\ell_\mathbb{D} : \mathbb{W}_{CCS} \to \Sigma$ sending all closed-world quasi-moves to id except $\heartsuit$ moves, which are sent to $\heartsuit$.

6.1. **Graphs with complementarity.** A *relation* $A \nrightarrow B$ between two reflexive graphs A and B is a subgraph $R \hookrightarrow A \times B$. Such a relation R is *total* when, for all vertices, resp. edges, $x \in A$, there exists a vertex, resp. an edge $y \in B$, such that $(x, y) \in R$. It is *partially functional* if there is at most one such y. It is *functional* when it is total and partially functional. The *domain* of R is the subgraph of A consisting of vertices and edges related to something in B.

Definition 6.1. A *graph with complementarity* is a reflexive graph A, equipped with a subgraph $A^\mathbb{W}$, a relation $\rhd^A : A^2 \nrightarrow A^\mathbb{W}$, and a map $\ell^A : A^\mathbb{W} \to \Sigma$, such that the composite $A^2 \nrightarrow A^\mathbb{W} \to \Sigma$ is partially functional and symmetric.

We let $A^\frown = \mathrm{dom}(\rhd^A)$ and write $a \frown a'$ for $(a, a') \in A^\frown$. We further denote the map $A^\frown \hookrightarrow A^2 \nrightarrow A^\mathbb{W} \to \Sigma$ by $(a, b) \mapsto (a \Downarrow b)$, and deem edges in $A^\mathbb{W}$ *closed-world*.

Remark 6.2. $A^{\frown}$ has to be symmetric as the domain of a symmetric relation.

Definition 6.3. A morphism of graphs with complementarity is a morphism $f\colon A \to B$ of reflexive graphs such that

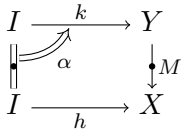
$$f(A^{\mathbb{W}}) \subseteq B^{\mathbb{W}} \qquad \ell^B \circ f^{\mathbb{W}} = \ell^A \qquad ((a_1, a_2) \triangleright^A a_3) \Rightarrow ((f(a_1), f(a_2)) \triangleright^A f(a_3)),$$

where $f^{\mathbb{W}}\colon A^{\mathbb{W}} \to B^{\mathbb{W}}$ is the restriction of f.

Proposition 6.4. *Graphs with complementarity and morphisms between them form a category* GCompl.

We now introduce the graph $\mathbb{IQ}$, which as announced in the introduction will serve as a base for making $\mathcal{S}_{\mathbb{D}CCS}$ and $\mathcal{T}_{\mathbb{D}CCS}$ into graphs with complementarity. It is an *interfaced* variant of $\mathbb{Q}$, hence its name.

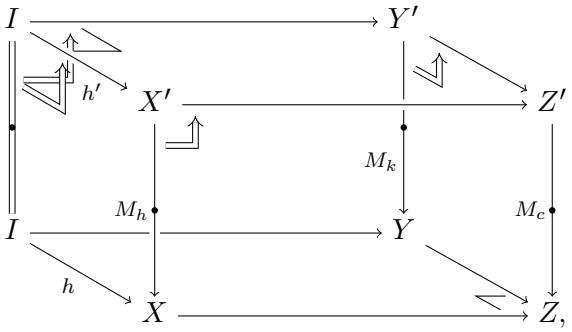
Example 6.5. Let $\mathbb{IQ}$ be the graph with as vertices all horizontal morphisms $h\colon I \to X$ from some interface to some position, and whose edges $k \to h$ are given by diagrams



(6.1)

in $\mathbb{D}_H$, where M is either a full move or an identity, such that if M is an input or an output, then the corresponding channel is in the image of I. $\mathbb{IQ}$ forms a reflexive graph with identities given by the case where $M = id^\bullet$, which forms a graph with complementarity as follows.

Let $(\mathbb{IQ})^{\mathbb{W}}$ consist of all closed-world quasi-moves in $\mathbb{IQ}$. For any $h\colon I \to X$, $k\colon J \to Y$, and $c\colon K \to Z$, let $(h, k) \triangleright^{\mathbb{IQ}} c$ iff $I = J = K$, $Z = h +_I k$, and c is the corresponding map $I \to Z$. On edges, for any $M_h\colon h' \to h$, $M_k\colon k' \to k$, and $M_c\colon c' \to c$, let $(M_h, M_k) \triangleright^{\mathbb{IQ}} M_c$ iff there exists a diagram



(6.2)

where M_c is a closed-world quasi-move and double cells with a 'double pullback' mark are cartesian, as below Axiom (P1) (page 25). (One easily shows that the upper square is also a pushout.) Then $(\mathbb{IQ})^{\frown}$, consists of all pairs (M_h, M_k) for which there exists a diagram of the shape (6.2).

Let $\ell^{\mathbb{IQ}}$ be the composite $(\mathbb{IQ})^{\mathbb{W}} \hookrightarrow \mathbb{W}_{CCS} \xrightarrow{\ell_{\mathbb{D}}} \Sigma$. It thus maps tick moves to $\heartsuit$ and all other closed-world moves to id. The composite $(\mathbb{IQ})^2 \overset{\triangleright^{\mathbb{IQ}}}{\dashrightarrow} (\mathbb{IQ})^{\mathbb{W}} \to \Sigma$ is indeed partially functional and symmetric.

There is an obvious morphism $\chi\colon \mathbb{IQ} \to \mathbb{Q}$ of reflexive graphs.

Example 6.6. Recall the alphabet $\mathbb{A}$ for CCS. It also forms a graph with complementarity, as follows. Let $\mathbb{A}^{\mathbb{W}}$ consist of all vertices and of all $\heartsuit$ and id edges. Let $\mathbb{A}^{\frown}$ consist, on vertices, of the diagonal, i.e., all pairs (n, n). On edges, let $e \frown e'$ when $\mathrm{dom}(e) \frown \mathrm{dom}(e')$ and:

- one of e and e' is in $\mathbb{A}^{\mathbb{W}}$, the other being an identity,
- or one of e and e' is an input on some $i \in \mathrm{dom}(e)$, the other being an output on i.

Define now our relation $\triangleright^{\mathbb{A}}$ to be the graph of the map sending all coherent pairs $e \frown e'$ to id, except when one is a $\heartsuit$, in which case the pair is sent to $\heartsuit : n \to n$. The axioms are easily satisfied.

Let $\xi : \mathbb{IQ} \to \mathbb{A}$ map any vertex $h : I \to X$ to $n = I(\star)$, and any edge (6.1) to

- id_n if M is an identity, a synchronisation, a fork, or a channel creation,
- $\heartsuit_n$ if M is a tick move,
- i if M is an input on $h_\star(i)$,
- $\bar{i}$ if M is an output on $h_\star(i)$.

This map ξ is a morphism of graphs with complementarity.

We have the following general way of constructing graphs with complementarity. For any graph with complementarity A and morphism of reflexive graphs $p : G \to A$, consider the following candidate complementarity structure on G.

Let $G^{\mathbb{W}} = G \times_A A^{\mathbb{W}}$ denote the pullback

$$\begin{array}{ccc} G^{\mathbb{W}} & \xrightarrow{p^{\mathbb{W}}} & A^{\mathbb{W}} \\ \downarrow & \lrcorner & \downarrow \\ G & \xrightarrow[p]{} & A. \end{array} \tag{6.3}$$

Further, let ℓ^G be the composite $G^{\mathbb{W}} \xrightarrow{p^{\mathbb{W}}} A^{\mathbb{W}} \xrightarrow{\ell^A} \Sigma$, and let $(x, y) \triangleright^G z$ iff $(p(x), p(y)) \triangleright^A p(z)$ (for both vertices and edges). In other words, $\triangleright^G$ is the relational composite

$$G^2 \xrightarrow{p^2} A^2 \overset{\triangleright^A}{\mapsto} A^{\mathbb{W}} \xleftarrow{p^{\mathbb{W}}} G^{\mathbb{W}},$$

where the backwards $p^{\mathbb{W}}$ arrow denotes the converse of the graph of $p^{\mathbb{W}}$.

Proposition 6.7. *For any subrelation $R \subseteq \triangleright^G$, if R is a symmetric relation, then $(G, G^{\mathbb{W}}, R, \ell^G)$ forms a graph with complementarity, and p is a morphism of graphs with complementarity to A.*

Proof. By standard relational algebra, the composite relation

$$G^2 \overset{\triangleright^G}{\mapsto} G^{\mathbb{W}} \xrightarrow{p^{\mathbb{W}}} A^{\mathbb{W}},$$

which is equal to

$$G^2 \xrightarrow{p^2} A^2 \overset{\triangleright^A}{\mapsto} A^{\mathbb{W}} \xleftarrow{p^{\mathbb{W}}} G^{\mathbb{W}} \xrightarrow{p^{\mathbb{W}}} A^{\mathbb{W}},$$

is included in

$$G^2 \xrightarrow{p^2} A^2 \overset{\triangleright^A}{\mapsto} A^{\mathbb{W}}.$$

Composing with ℓ^A, we obtain that $(\ell^G \circ \triangleright^G) \subseteq (\ell^A \circ \triangleright^A \circ p^2)$, which is straightforwardly symmetric and partially functional. A subrelation of a partially functional relation is automatically partially functional, so $\ell^G \circ R$ is partially functional. It is symmetric because R is, hence the result. □

Example 6.8. *CCS* forms a graph with complementarity over $\mathbb{A}$ by the last proposition, taking R to relate

- all pairs $(n \vdash P, n \vdash Q)$ to $n \vdash P \mid Q$ on vertices,
- any transitions $(\Gamma \vdash P_1) \overset{\alpha}{\leftarrow} (\Gamma \vdash P_1')$ and $(\Gamma \vdash P_2) \overset{id}{\leftarrow} (\Gamma \vdash P_2)$ with $(\Gamma \vdash P_1 \mid P_2) \overset{\alpha}{\leftarrow} (\Gamma \vdash P_1' \mid P_2)$, and symmetrically,
- and any two transitions $(\Gamma \vdash P_1) \overset{\alpha}{\leftarrow} (\Gamma \vdash P_1')$ and $(\Gamma \vdash P_2) \overset{\overline{\alpha}}{\leftarrow} (\Gamma \vdash P_2')$ with $(\Gamma \vdash P_1 \mid P_2) \overset{id}{\leftarrow} (\Gamma \vdash P_1' \mid P_2')$.

Proposition 6.9. *Suppose given a choice, for all $x, y \in G$ and $a \in A$ such that $(p(x), p(y)) \rhd^A a$, of a vertex $[x,y]_a \in G$ such that $p([x,y]_a) = a$, satisfying the following condition: for all edges $e_x : x' \to x$ and $e_y : y' \to y$ in G, and $e_a : a' \to a$ in A, if $(p(e_x), p(e_y)) \rhd^A e_a$, then there exists a $[e_x, e_y]_{e_a} : [x', y']_{a'} \to [x,y]_a$ such that $p([e_x, e_y]_{e_a}) = e_a$.*

Then, $(G, G^{\mathbb{W}}, \rhd^G, \ell^G)$ forms a graph with complementarity, and p is a morphism of graphs with complementarity.

Proof. Recalling the beginning of the proof of Proposition 6.7, the hypothesis implies that the inclusion

$$(G^2 \overset{\rhd^G}{\rightarrowtail} G^{\mathbb{W}} \xrightarrow{p^{\mathbb{W}}} A^{\mathbb{W}}) \subseteq (G^2 \xrightarrow{p^2} A^2 \overset{\rhd^A}{\rightarrowtail} A^{\mathbb{W}})$$

is actually an equality.

Composing with ℓ^A, we obtain that $\ell^G \circ \rhd^G = \ell^A \circ \rhd^A \circ p^2$, which is straightforwardly symmetric and partially functional. The morphism p is a morphism of graphs with complementarity by construction. □

Definition 6.10. Let $\mathcal{S}^{\mathbb{IQ}} = \Delta_\chi(\mathcal{S})$ and $\mathcal{T}^{\mathbb{IQ}} = \Delta_\chi(\mathcal{T})$ be the pullbacks of $\mathcal{S} \to \mathbb{Q}$ and $\mathcal{T} \to \mathbb{Q}$ along $\chi : \mathbb{IQ} \to \mathbb{Q}$.

Example 6.11. $\mathcal{S}^{\mathbb{IQ}}$ and $\mathcal{T}^{\mathbb{IQ}}$ form graphs with complementarity over $\mathbb{IQ}$ by Proposition 6.9. The canonical relation $\rhd^{CCS}$ does not satisfy the condition of Proposition 6.9, however. Indeed, e.g., any non-silent transition $(\Gamma \vdash P_1) \overset{\alpha}{\leftarrow} (\Gamma \vdash P_1')$ and silent but non-identity transition $(\Gamma \vdash P_2) \overset{id}{\leftarrow} (\Gamma \vdash P_2')$ are not coherent in *CCS*, although their images under the projection to $\mathbb{A}$ are so. (Amalgamating two such transitions in *CCS* requires a path of length 2, as will be used below.) What saves $\mathcal{S}^{\mathbb{IQ}}$ and $\mathcal{T}^{\mathbb{IQ}}$ from this issue is that projecting to $\mathbb{IQ}$ does not hide away, e.g., synchronisations.

6.2. **Modular graphs and fair testing equivalence.** We now introduce the notion of *modular* graph, which is appropriate for defining fair testing. We could actually introduce fair testing for arbitrary graphs with complementarity, but the extra generality would make little sense.

For any graph with complementarity G, $G^{\frown}$ forms an LTS over Σ, through $G^{\frown} \overset{\Downarrow}{\to} \Sigma$.

Definition 6.12. G is *modular* iff for all $(x,y) \rhd^G z$ we have both:

(1) for all $e : z' \to z$, there exists $e_x : x' \to x$ and $e_y : y' \to y$ such that $(e_x, e_y) \rhd^G e$; and
(2) for all $e_x : x' \to x$ and $e_y : y' \to y$ such that $e_x \frown e_y$ there exists $e : z' \to z$ such that $(e_x, e_y) \rhd^G e$.

Remark 6.13. The second condition is almost redundant: in any graph with complementarity G, there exists e' such that $(e_x, e_y) \triangleright^G e'$, but the target of e' may be any u such that $(x, y) \triangleright^G u$; it does not have to be z.

Proposition 6.14. *G is* modular *iff $\triangleright^G$ is a strong bisimulation over Σ.*

We here implicitly view $\triangleright^G$ as a relation $G^{\frown} \nrightarrow G^{\mathbb{W}}$.

Proof. Since $\triangleright^G$ is a relation over Σ, it is enough to prove that both projections are graph fibrations, which is directly equivalent to modularity. □

Example 6.15. $\mathcal{S}^{\mathbb{IQ}}$ and $\mathcal{T}^{\mathbb{IQ}}$, as well as *CCS*, are modular.

We now define fair testing in any modular graph, and compare with both semantic fair testing equivalence ($\sim_f$) for strategies and standard fair testing equivalence ($\sim_{f,s}$) for CCS processes. Recall that $\sim_\Sigma$ denotes strong bisimilarity over Σ.

Lemma 6.16. *For any modular graph with complementarity G and $x, y, z, t \in G$, if $(x, y) \triangleright^G z$ and $(x, y) \triangleright^G t$, then $z \sim_\Sigma t$.*

Proof. We have $z \sim_\Sigma (x, y) \sim_\Sigma t$. □

Any modular graph may be equipped with a choice of z such that $(x, y) \triangleright^G z$, for all $x \frown y$. We denote such a choice by $[x, y]$. By the lemma, the choice of z does not matter as long as we only consider properties invariant under $\sim_\Sigma$. Here, we only need the standard predicate for fair testing.

Definition 6.17. For any reflexive graph G over Σ, let $\perp^G$ denote the set of all $x \in G$ such that for all $x \Leftarrow x'$ there exists $x' \overset{\heartsuit}{\Longleftarrow} x''$.

When G is a graph with complementarity, we often denote $\perp^{G^{\mathbb{W}}}$ by $\perp^G$. There is no confusion because G is not even a graph over Σ in general.

In any modular graph with complementarity G, let, for any $x \in G$, $x^{\frown} = \{y \mid x \frown y\}$, and let $x \bowtie y$ iff $x^{\frown} = y^{\frown}$.

Definition 6.18. For any $x, y \in G$, let $x \sim_f^G y$ iff $x \bowtie y$ and for all $z \in x^{\frown}$, $[x, z] \in \perp^G$ iff $[y, z] \in \perp^G$.

We may at last define fair relations:

Definition 6.19. For all modular graphs with complementarity G and H, and full relations $R : G \nrightarrow H$, let R *preserve fair testing equivalence* when, for all $x \, R \, x'$ and $y \, R \, y'$, $(x \sim_f^G y)$ implies $(x' \sim_f^H y')$. R *reflects fair testing equivalence* when the converse implication holds. R is *fair* when it preserves and reflects fair testing equivalence.

Modularity enables a first, easy characterisation of fair testing.

Proposition 6.20. *If G is modular, then for any $x \frown y$, $[x, y] \in \perp^G$ iff $(x, y) \in \perp^{G^{\frown}}$.*

Proof. A direct consequence of Proposition 6.14. □

We now prove that the general definition of fair testing equivalence instantiates correctly for $\mathcal{S}^{\mathbb{IQ}}$ and *CCS*. First, we easily have

Proposition 6.21. *For any two CCS processes P and Q over n, $P \sim_{f,s} Q$ iff $P \sim_f^{CCS} Q$.*

Proof. Straightforward. □

We now wish to compare $\perp^{\mathcal{S}^{\mathbb{IQ}}}$, as defined in this section, and the semantic $\perp\!\!\!\perp$. As an intermediate step, we consider the following, bare $\perp\!\!\!\downarrow$, which lives over $\mathbb{Q}$, but is defined in terms of LTSs (as opposed to successful states of strategies). Let $\mathcal{S}^{\mathbb{W}}$ be the restriction of $\mathcal{S}$ to closed-world transitions, i.e., the pullback of $\mathcal{S} \to \mathbb{Q}$ along the inclusion $\mathbb{W}_{CCS} \hookrightarrow \mathbb{Q}$; this is an LTS over Σ via $\ell_{\mathbb{D}}$. Let $\perp\!\!\!\downarrow = \perp^{\mathcal{S}^{\mathbb{W}}}$ denote the set of pairs $(X, S) \in \mathcal{S}$ such that for all $(X, S) \Leftarrow (X', S')$ there exists $(X', S') \overset{\heartsuit}{\Longleftarrow} (X'', S'')$.

Lemma 6.22. *For all $(X, S) \in \mathcal{S}$, $\overline{S} \in \perp\!\!\!\perp_X$ iff $(X, S) \in \perp\!\!\!\downarrow$.*

This essentially amounts to checking that the notions of closed-world, successful, and unsuccessful play (Definition 3.22), correspond with closed-world, successful, and unsuccessful transition sequences. The former are defined in terms of plays and moves therein, while the latter rest upon the map $\ell_{\mathbb{D}} : \mathbb{W}_{CCS} \to \Sigma$.

We first observe:

Lemma 6.23. *For any two closed-world plays W, W' over X, and $\alpha : W \to W'$ in $\mathbb{D}_H(X)$, α is an isomorphism, and it is unique.*

Proof of Lemma 6.22. Let $S \in \mathsf{S}_X$ and assume $\overline{S} \in \perp\!\!\!\perp_X$. Let $S \Leftarrow S'$ (over Σ). This means that there exists a path p

$$X = X_0 \xleftarrow{M_1} X_1 \xleftarrow{M_2} \dots X_n = X',$$

such that, omitting positions,

$$S = S_0 \xleftarrow{M_1} S_1 \xleftarrow{M_2} \dots S_n = S',$$

and p is mapped by $\ell^\star_{\mathbb{D}}$ to the path of length n consisting only of *id* edges. This implies by induction the existence of $\sigma \in \overline{S}(W)$, where $W = M_1 \bullet \dots \bullet M_n$ is closed-world and unsuccessful, such that $S' = (S \cdot W)_{|\psi(\sigma)}$. Because $\overline{S} \in \perp\!\!\!\perp_X$, there exists a successful, closed-world play W', a morphism $f : W \to W'$ in $\mathbb{E}(X)$, and $\sigma' \in \overline{S}(W')$ such that $\sigma' \cdot f = \sigma$. By Lemma 6.23, W' is isomorphic to an extension of W with closed-world moves, say $W' \cong W \bullet M_{n+1} \bullet \dots \bullet M_{n+m}$. By induction on m, we obtain a path

$$S' = S_n \xleftarrow{M_{n+1}} S_{n+1} \xleftarrow{M_{n+2}} \dots S_{n+m},$$

where $S_{n+m} = (S \cdot W')_{|\psi(\sigma')}$. Because W' is successful, there exists $i \in m$ such that $\ell_{\mathbb{D}}(M_{n+i}) = \heartsuit$, hence $S' \overset{\heartsuit}{\Longleftarrow} S_{n+i}$. Thus, $(X, S) \in \perp\!\!\!\downarrow$.

Conversely, assume $(X, S) \in \perp\!\!\!\downarrow$. Let W be an unsuccessful, closed-world play over X and $\sigma \in \overline{S}(W)$. Picking a decomposition $W = M_1 \bullet \dots \bullet M_n$ of W, we obtain a path p

$$S = S_0 \xleftarrow{M_1} S_1 \dots \xleftarrow{M_n} S_n = S'$$

in $\mathcal{S}$ such that $S' = (S \cdot W)_{|\psi(\sigma)}$, which yields $S \Leftarrow S'$. Because $(X, S) \in \perp\!\!\!\downarrow$, there exists $S' \Leftarrow S'' \overset{\heartsuit}{\leftarrow} S'''$, with underlying path

$$S' = S_n \xleftarrow{M_{n+1}} S_{n+1} \dots \xleftarrow{M_{n+m}} S_{n+m} = S'' \xleftarrow{M_{n+m+1}} S'''$$

in $\mathcal{S}^{\mathbb{W}}$, such that $\ell_{\mathbb{D}}(M_{n+i}) = id$ for all $i \in m$ and $\ell_{\mathbb{D}}(M_{n+m+1}) = \heartsuit$. But by definition this means that $S''' = (S' \cdot W')_{|\psi(\sigma')}$ for some

$$\sigma' \in \overline{S'}(W') = \overline{(S \cdot W)_{|\psi(\sigma)}}(W') = \{\sigma'' \in \overline{S}(W \bullet W') \mid \sigma'' \cdot f = \sigma\},$$

where $W' = M_{n+1} \bullet \ldots \bullet M_{n+m+1}$ and $f : W \to (W \bullet W')$ is the extension. By construction, $\sigma' \cdot f = \sigma$. Hence, $\overline{S} \in \perp\!\!\!\perp_X$. □

We furthermore have:

Lemma 6.24. *For any vertex $h : I \to X$ of $\mathbb{IQ}$ and $S \in \mathsf{S}_X$, $(I, h, S) \in \perp^{\mathcal{S}^{\mathbb{IQ}}}$ iff $(X, S) \in \downarrow$.*

Proof. The map $\chi^{\mathbb{W}} : \mathbb{IQ}^{\mathbb{W}} \to \mathbb{Q}^{\mathbb{W}}$ is a strong, functional bisimulation, because for any $h : I \to X$ and closed-world move $M : Y \rightarrowtail X$, there exists a diagram (6.1). Thus, the projection $(\mathcal{S}^{\mathbb{IQ}})^{\mathbb{W}} \to \mathcal{S}^{\mathbb{W}}$ is a strong, functional bisimulation by Proposition 2.11. □

Remark 6.25. Interfaces are pretty irrelevant here, and indeed we could have decreed that closed-world moves only relate vertices with empty interfaces in $\mathbb{IQ}$. This is unnecessary here, though, so we stick to the simpler definition, but it will be crucial for the π-calculus.

This entails:

Corollary 6.26. *For any $h : I \to X$, $h' : I \to X'$, $S \in \mathsf{S}_X$, and $S' \in \mathsf{S}_{X'}$, $(I, h, S) \sim_f (I, h', S')$ iff $(I, h, S) \sim_f^{\mathcal{S}^{\mathbb{IQ}}} (I, h', S')$.*

Proof. We have

$$\begin{array}{c}
(I, h, S) \sim_f (I, h', S') \\
\Updownarrow \text{(by definition)} \\
\forall Y, k : I \to Y, T \in \mathsf{S}_Y, (\overline{[S, T]} \in \perp\!\!\!\perp_{X +_I Y} \Leftrightarrow \overline{[S', T]} \in \perp\!\!\!\perp_{X' +_I Y}) \\
\Updownarrow \text{(by Lemma 6.22)} \\
\forall Y, k : I \to Y, T \in \mathsf{S}_Y, ((X +_I Y, [S, T]) \in \downarrow \Leftrightarrow (X' +_I Y, [S', T]) \in \downarrow) \\
\Updownarrow \text{(by Lemma 6.24)} \\
\forall Y, k : I \to Y, T \in \mathsf{S}_Y, ((I \to X +_I Y, [S, T]) \in \perp^{\mathcal{S}^{\mathbb{IQ}}} \Leftrightarrow (I \to X' +_I Y, [S', T]) \in \perp^{\mathcal{S}^{\mathbb{IQ}}}) \\
\Updownarrow \text{(by definition)} \\
(I, h, S) \sim_f^{\mathcal{S}^{\mathbb{IQ}}} (I, h', S'),
\end{array}$$

which concludes the proof. □

6.3. **Adequacy.** Until now, our study of graphs with complementarity and fair testing therein is intrinsic, i.e., fair testing equivalence in a modular graph with complementarity G does not depend on any alphabet. We now address the question of what an alphabet should be, for G. The main idea is that such an alphabet A should be a graph with complementarity, and that viewing it as an alphabet for G is the same as providing a morphism $p : G \to A$ in GCompl, satisfying a certain condition called *adequacy*. To understand the role of this condition, one should realise that edges in G may be much too fine a tool for checking fair testing equivalence. E.g., in $\mathcal{S}^{\mathbb{IQ}}$, they include information about which players played which move. Thus, although it is true that weak bisimilarity implies fair testing equivalence, this property is essentially useless for fair testing, because too few strategies are weakly bisimilar. Any morphism $p : \mathcal{S}^{\mathbb{IQ}} \to A$ induces an *a priori* coarser version of fair testing for $\mathcal{S}^{\mathbb{IQ}}$, where one only looks at labels in A. *Adequacy* is a sufficient condition for this latter version to coincide with the original. This will in particular entail that weak bisimilarity over A is finer than fair testing equivalence.

Adequacy relies on the following:

Definition 6.27. Consider, for any $p : G \to A$ and $q : H \to A$ the pullback

$$\begin{array}{ccc} G \diamond_A H & \longrightarrow & A^{\frown} \\ \downarrow & \lrcorner & \downarrow \\ G \times H & \xrightarrow[p \times q]{} & A^2. \end{array}$$

We call $G \diamond_A H$ the *blind composition* of G and H over A, viewed as an LTS over Σ via $G \diamond_A H \to A^{\frown} \to \Sigma$.

Recall from Section 2.2 that $\approx_A$ denotes weak bisimilarity for reflexive graphs over A.

Definition 6.28. Let $p : G \to A$ be a morphism of graphs with complementarity. We say that p is *adequate* iff

- the graph of $\mathrm{ob}\, G^{\frown} \hookrightarrow \mathrm{ob}(G \diamond_A G)$ is included in $\approx_\Sigma$, and
- for all $x, y \in \mathrm{ob}(G)$, $x \frown y$ iff $p(x) \frown p(y)$.

Concretely, any transition $(e_1, e_2) \in G^{\frown}$ is matched, without any hypothesis on G, by (e_1, e_2) itself. Conversely, having a transition $(x_1, x_2) \xleftarrow{e_1, e_2} (x_1', x_2')$ in $G \diamond_A G$ means that $p(e_1) \Downarrow p(e_2) = \sigma$. Adequacy demands that there exists a path $(r_1, r_2) : (x_1, x_2) \leftarrow^\star (x_1'', x_2'')$ in $G^{\frown}$, such that $\widetilde{r_1 \Downarrow^\star r_2} = \widetilde{(\sigma)}$, and $(x_1'', x_2'') \approx_\Sigma (x_1', x_2')$, where the left-hand side is in $G^{\frown}$ and the right-hand side is in $G \diamond_A G$.

Recall the map $\xi : \mathbb{IQ} \to \mathbb{A}$ from Example 6.6. Via this map, $\mathcal{S}^{\mathbb{IQ}}$ and $\mathcal{T}^{\mathbb{IQ}}$ form LTSs and even graphs with complementarity over $\mathbb{A}$.

Proposition 6.29. *The maps from CCS, $\mathcal{S}^{\mathbb{IQ}}$, and $\mathcal{T}^{\mathbb{IQ}}$ to $\mathbb{A}$ are adequate.*

Proof. For all three graphs $p : G \to \mathbb{A}$ over $\mathbb{A}$, both $G^{\frown}$ and $G \diamond_A G$ form graphs over $\mathbb{A}^{\mathbb{W}}$, because $\rhd^{\mathbb{A}} : \mathbb{A}^2 \rightarrowtail\!\!\!\!\to \mathbb{A}^{\mathbb{W}}$ is actually partially functional. In each case, the graph of $\mathrm{ob}\, G^{\frown} \hookrightarrow \mathrm{ob}(G \diamond_A G)$ is a weak bisimulation over $\mathbb{A}^{\mathbb{W}}$, because for all e and e' in G, if $p(e) \frown p(e')$, then either $e \frown e'$, or both interleavings are coherent, i.e., $(e, id) \frown (id, e')$ and $(id, e) \frown (e', id)$, pointwise. (Here, e.g., (e, id) denotes the *path* $\cdot \xleftarrow{e} \cdot \xleftarrow{id} \cdot$.) □

The only subtle point is that this only holds thanks to the restrictions put on edges of $\mathbb{IQ}$. E.g., consider the graph the graph $\mathbb{IQ}_-$ with the same vertices as $\mathbb{IQ}$, and edges $(I \xrightarrow{k} Y) \to (I \xrightarrow{h} X)$ given just as for $\mathbb{IQ}$, except that we do not require existence of a diagram (6.1). Pullback yields a graph $\mathcal{S}_-$ over $\mathbb{IQ}_-$. Extending ξ to $\xi' : \mathbb{IQ}_- \to \mathbb{A}$ in the obvious way, we obtain a graph over $\mathbb{A}$. Consider now the moves $o_{2,1}, \iota_{2,1} : [2] \rightarrowtail\!\!\!\!\to [2]$, let $I = 2 \cdot \star$, and let f be one of the two embeddings $I \to [2]$, say the one which is an inclusion at $\star$, f' being the other. Recalling labels in $\mathbb{A}$ from Definition 2.20, we have edges $(I, f, [2]) \xleftarrow{\overline{1}} (I, f, [2])$ and $(I, f, [2]) \xleftarrow{1} (I, f', [2])$, and $(I, f, [2]) \frown (I, f, [2])$. However, the two edges are not coherent, because any attempt to construct a diagram (6.2) (with here $h = h' = k = f$, and $k' = f'$) fails (even if we forget about the vertical identity). This is the very reason we use $\mathbb{IQ}$ instead of $\mathbb{IQ}_-$.

We have the following two easy properties of blind composition.

Proposition 6.30. *For any modular G, adequate $p : G \to A$, and $x \frown y$ in G, we have $[x, y] \in \perp^G$ iff $(x, y) \in \perp^{G \diamond_A G}$.*

Proof. We have $[x, y] \approx_\Sigma ((x, y) \in G^{\frown}) \approx_\Sigma ((x, y) \in G \diamond_A G)$. □

Proposition 6.31. *For any H over A, modular G, adequate $p\colon G \to A$, $x_1, x_2 \in G$, and y in H, if $x_2 \approx_A y$, then*

$$[x_1, x_2] \in \perp^G \text{ iff } (x_1, y) \in \perp^{G \diamond_A H}.$$

Proof. By Proposition 6.30, it is enough to prove that the right-hand side is equivalent to $(x_1, x_2) \in \perp^{G \diamond_A G}$, which is straightforward by hypothesis. □

We conclude this section by stating the main property of blind composition, Proposition 6.37 below, which will be used extensively in the next section.

To start with, recall the following notation from Section 2.2.1. There, considering a morphism $p\colon G \to A$ of reflexive graphs, we defined $x \ {}_A\overset{r}{\Longleftarrow} x'$, for $x, x' \in \mathrm{ob}(G)$ and $r\colon p(x) \leftarrow p(x')$ in $A^\star$. Namely, this denotes any path $r'\colon x \leftarrow^\star x'$ in G, such that $\widetilde{p^\star(r')} = \widetilde{r}$.

In order to state Proposition 6.37, we now need to equip $\mathsf{fc}(A)$ with complementarity structure, but we cannot do it over the graph Σ, because closed-world paths may contain more than one $\heartsuit$ edge, hence cannot all be mapped to Σ. We thus define *categories* with complementarity.

The notions of relation, partial functionality, functionality, totality, and domain on reflexive graphs carry over to categories, e.g., a relation $A \nrightarrow B$ is a subcategory $R \subseteq A \times B$. The only subtlety is that the definitions imply certain functoriality properties. E.g., for any composites $g \circ f$ in A and $g' \circ f'$ in B, if $(f, f') \in R$ and $(g, g') \in R$, because R, as a subcategory, is stable under composition, we have for free that $(g \circ f, g' \circ f') \in R$. Similarly, if $(x, y) \in R$ for objects $x \in A$ and $y \in B$, then $(id_x, id_y) \in R$. We thus rename partial functionality and functionality into partial functoriality and functoriality in this setting.

Definition 6.32. A *category with complementarity* is a category A, equipped with a subcategory $A^{\mathbb{W}}$, a relation $\rhd^A\colon A^2 \nrightarrow A^{\mathbb{W}}$, and a functor $\ell^A\colon A^{\mathbb{W}} \to \mathsf{fc}(\Sigma)$, such that the composite $A^2 \nrightarrow A^{\mathbb{W}} \to \mathsf{fc}(\Sigma)$ is partially functorial and symmetric.

Again, we let $A^{\frown} = \mathrm{dom}(\rhd^A)$ and write $a \frown a'$ for $(a, a') \in A^{\frown}$. We further denote the map $A^{\frown} \hookrightarrow A^2 \nrightarrow A^{\mathbb{W}} \to \mathsf{fc}(\Sigma)$ by $(a, b) \mapsto (a \Downarrow b)$, and deem morphisms in $A^{\mathbb{W}}$ *closed-world*.

Defining functors with complementarity in the obvious way, we obtain:

Proposition 6.33. *Categories with complementarity form a (locally small) category* CCompl.

Consider the functor $\mathsf{UCompl}\colon \mathsf{CCompl} \to \mathsf{GCompl}$ mapping any category with complementarity $\mathbb{C}$ to its underlying graph, say G, which we equip with complementarity structure as follows. First, define $G^{\mathbb{W}}$ and ℓ^G by the pullback

$$\begin{array}{ccc} G^{\mathbb{W}} & \xrightarrow{\ i\ } & \mathbb{C}^{\mathbb{W}} \\ {\scriptstyle \ell^G}\downarrow & \lrcorner & \downarrow{\scriptstyle \ell^{\mathbb{C}}} \\ \Sigma & \xrightarrow[\ \eta\ ]{} & \mathsf{fc}(\Sigma). \end{array}$$

Furthermore, let $\rhd^{\mathbb{C}}$ consist of all triples (x, y, z) of vertices (resp. edges) such that $z \in G^{\mathbb{W}}$ and $(x, y) \rhd^{\mathbb{C}} z$ (which is a pullback of $(\rhd^{\mathbb{C}}) \hookrightarrow \mathbb{C}^2 \times \mathbb{C}^{\mathbb{W}}$ along $\mathbb{C}^2 \times G^{\mathbb{W}} \hookrightarrow \mathbb{C}^2 \times \mathbb{C}^{\mathbb{W}}$). This clearly equips G with complementarity structure and extends to the announced functor UCompl.

This functor does not appear to have a left adjoint, because complementarity in G may behave badly w.r.t. composition in $\mathsf{fc}(G)$. However, we may define the following candidate structure on $\mathsf{fc}(G)$. Consider any graph with complementarity G, and let us

start by defining a complementarity structure on $G^\star$. Let $(G^\star)^{\mathbb{W}}$ denote the subcategory of closed-world paths in G, i.e., $(G^\star)^{\mathbb{W}} = (G^{\mathbb{W}})^\star$. Accordingly, let $\ell^{G^\star}$ be the composite

$$(G^{\mathbb{W}})^\star \xrightarrow{(\ell^G)^\star} \Sigma^\star \xrightarrow{\simeq} \mathsf{fc}(\Sigma).$$

Finally, consider the functor

$$(G^2)^\star \xrightarrow{\langle \pi^\star, (\pi')^\star \rangle} (G^\star)^2.$$

It yields a relation $(G^2)^\star \rightarrowtail (G^\star)^2$, whose converse we use to define $\rhd^{G^\star}$ as the composite relation

$$(G^\star)^2 \overset{\langle \pi^\star, (\pi')^\star \rangle^\dagger}{\rightarrowtail} (G^2)^\star \overset{(\rhd^G)^\star}{\rightarrowtail} G^\star.$$

Concretely, $\rhd^{G^\star}$ is $\rhd^G$ on objects, and on paths, we have $(r_1, r_2) \rhd^{G^\star} r$ iff all three paths r_1, r_2, and r have the same length n and $(r_1^i, r_2^i) \rhd^G r^i$ for all $i \in n$. This clearly makes $G^\star$ into a category with complementarity.

Let us now define our candidate complementarity structure on $\mathsf{fc}(G)$, for any $G \in \mathsf{GCompl}$. Let first $\mathsf{fc}(G)^{\mathbb{W}}$ be the image of $(G^\star)^{\mathbb{W}} \hookrightarrow G^\star \xrightarrow{\simeq} \mathsf{fc}(G)$, i.e., all id-free, closed-world paths. This in particular induces a functor $(G^\star)^{\mathbb{W}} \to (\mathsf{fc}(G))^{\mathbb{W}}$, with which $\ell^{(G^\star)}$ is obviously compatible, hence we define $\ell^{\mathsf{fc}(G)}$ to be the induced functor. Finally, let $\rhd^{\mathsf{fc}(G)}$ be the following relational composite, where the backwards arrow denotes a converse:

$$(\mathsf{fc}(G))^2 \xleftarrow{(\simeq)^2} (G^\star)^2 \overset{\rhd^{G^\star}}{\rightarrowtail} (G^\star)^{\mathbb{W}} \to \mathsf{fc}(G)^{\mathbb{W}}.$$

Concretely, $(\rho_1, \rho_2) \rhd^{\mathsf{fc}(G)} \rho_3$ iff there exist $(r_1, r_2) \rhd^{G^\star} r_3$ such that $\widetilde{r_i} = \rho_i$ for $i = 1, 2, 3$. Intuitively, ρ_1 and ρ_2 are coherent if upon insertion of identities at appropriate places they become pointwise coherent.

The relational composite $\mathsf{fc}(G)^2 \overset{\rhd^{\mathsf{fc}(G)}}{\rightarrowtail} \mathsf{fc}(G)^{\mathbb{W}} \xrightarrow{\ell^{\mathsf{fc}(G)}} \Sigma$ is obviously symmetric and furthermore partially functional on objects, so in order to equip $\mathsf{fc}(G)$ with complementarity structure, it only misses partial functoriality on morphisms.

Definition 6.34. Let GCompl_+ denote the full subcategory of GCompl spanning objects G such that the above composite is partially functorial on morphisms, which we call *functorial* graphs with complementarity.

Example 6.35. A sufficient condition for a graph with complementarity G to be functorial is to satisfy

(i) for any two edges e and e', and object x, if $e \supset e'$, $e \neq id$, and $e \supset id_x$, then e' is an identity;

(ii) for all edges e and e', $(e \Downarrow \widetilde{id}); (\widetilde{id} \Downarrow e')$ and $(id \Downarrow \widetilde{e'}); (\widetilde{e} \Downarrow id)$ are defined at the same time and then equal.

The three graphs with complementarity *CCS*, $\mathcal{S}^{\mathbb{IQ}}$, and $\mathcal{T}^{\mathbb{IQ}}$ satisfy these conditions, hence are functorial.

The forgetful functor UCompl of course lands into GCompl_+ and we view it as a functor $\mathsf{CCompl} \to \mathsf{GCompl}_+$ from now on.

Proposition 6.36. *The above construction of* $\mathsf{fc}(G)^{\mathbb{W}}$, $\ell^{\mathsf{fc}(G)}$, *and* $\rhd^{\mathsf{fc}(G)}$ *extends to a left adjoint to* UCompl, *which coincides with* fc *on underlying graphs.*

We henceforth denote the left adjoint by fc.

Proof. Proving that this is left adjoint to UCompl reduces to showing that the composite

$$\mathsf{GCompl}_+(G, \mathsf{UCompl}(C)) \hookrightarrow \mathsf{Gph}(G, U(C)) \xrightarrow{\cong} \mathsf{Cat}(\mathsf{fc}(G), C)$$

factors through $\mathsf{CCompl}(\mathsf{fc}(G), C) \hookrightarrow \mathsf{Cat}(\mathsf{fc}(G), C)$, and conversely the composite

$$\mathsf{CCompl}(\mathsf{fc}(G), C) \hookrightarrow \mathsf{Cat}(\mathsf{fc}(G), C) \xrightarrow{\cong} \mathsf{Gph}(G, U(C))$$

factors through $\mathsf{GCompl}_+(G, \mathsf{UCompl}(C)) \hookrightarrow \mathsf{Gph}(G, U(C))$, which is routine. □

We may now state the main property of blind composition:

Proposition 6.37. *For any graphs with complementarity G and H over A, and transition sequences $x \ {}_A\!\overset{\rho_x}{\Longleftarrow} x'$ and $y \ {}_A\!\overset{\rho_y}{\Longleftarrow} y'$ respectively in G and H, if $(\rho_x, \rho_y) \rhd^{\mathsf{fc}(A)} \rho$, then $(x, y) \ {}_A\!\overset{\rho}{\Longleftarrow} (x', y')$ in $G \diamond_A H$.*

Proof. Let $p : G \to A$ and $q : H \to A$ be the given projections. Let also $(r_1^i, r_2^i) \rhd^A r_3^i$ for all $i \in n$ witness the fact that $(\rho_x, \rho_y) \rhd^{\mathsf{fc}(A)} \rho$. It is enough to prove $(x, y) \ {}_{A^\star}\!\overset{r_3}{\longleftarrow} (x', y')$, which is in fact a trivial induction on the length of r_3 using the definition of $G \diamond_A H$. □

6.4. **Trees.** Returning to our main question, we know by Theorem 5.35 that the graph morphism $\mathcal{T} \to \mathcal{S}$ is a functional, strong bisimulation over $\mathbb{Q}$. Hence, by Proposition 2.11, we have:

Proposition 6.38. *The graph morphism $\mathcal{T}^{\mathbb{IQ}} \to \mathcal{S}^{\mathbb{IQ}}$ is a functional, strong bisimulation over $\mathbb{IQ}$, and thus also over $\mathbb{A}$.*

In this section, we introduce a criterion for a relation $R : G \nrightarrow H$ between modular graphs with complementarity over some adequate alphabet A, which essentially ensures that if $R \subseteq \approx_A$, then R is fair. This will reduce our main question to proving that the full relation induced by the map $CCS \hookrightarrow \mathcal{T}^{\mathbb{IQ}}$ is included in weak bisimilarity over $\mathbb{A}$, which we do in Section 6.5.

Our criterion will rest upon the notion of A-tree, for any graph with complementarity A, which is directly inspired by the work of Brinksma et al. on *failures* [48].

Let the set $\mathcal{H}^A$ of *A-trees* consist of possibly infinite terms in the grammar

$$\frac{\dots \qquad v_i \vdash t_i \qquad \dots \qquad (\forall i \in n)}{v \vdash \sum_{i \in n} a_i.t_i} \ (n \in \mathbb{N})$$

where for all $i \in n$, $a_i : v_i \to v$ in A is not silent, i.e., $a_i \in A^{\mathbb{W}}$ implies $\ell^A(a_i) \neq id$. A-trees form a reflexive graph over A with edges determined by

$$(v \vdash \sum_{i \in n} a_i.t_i) \xleftarrow{a_i} (v_i \vdash t_i).$$

Definition 6.39. A modular graph $p : G \to A$ over A *has enough A-trees* iff for all $x \in G$, $v \in A$ such that $p(x) \supset v$, for all A-trees $v \vdash t$, there exists $x_t \in G$ such that $x \supset x_t$ and x_t is weakly bisimilar to t (over A).

Remark 6.40. In the case where $x \supset x'$ iff $p(x) \supset p(x')$, this is equivalent to requiring that for all $a \in A$ and A-tree t over a, there exists $x_t \in G$ such that $p(x_t) = a$ and $x_t \approx_A t$.

Example 6.41. *CCS*, $\mathcal{S}^{\mathbb{IQ}}$, and $\mathcal{T}^{\mathbb{IQ}}$ have enough $\mathbb{A}$-trees, and Remark 6.40 applies.

A-trees yield a new testing equivalence, called *A-tree* equivalence, as follows.

Definition 6.42. For any modular $p{:}\, G \to A$, let $\sim_f^{G|A}$ be the relation defined by $x \sim_f^{G|A} y$ iff $x \bowtie y$ and for all $v \in A$ such that $p(x) \subset v$ and A-trees $t \in \mathcal{H}_v^A$,

$$(x,t) \in \perp^{G \diamond_A \mathcal{H}^A} \text{ iff } (y,t) \in \perp^{G \diamond_A \mathcal{H}^A}.$$

A graph with complementarity A *has enough ticks* iff for all $a \in A$, there exists an edge $\heartsuit_a{:}\, a' \to a$ such that $\ell^A(\heartsuit_a) = \heartsuit$. Furthermore, A is *inertly silent* iff for all $e{:}\, b \to a$ in $A^{\mathbb{W}}$ such that $\ell^A(e) = id$, we have $a = b$ and $e = id_a$.

Definition 6.43. A graph with complementarity A is a *nice alphabet* iff it has enough ticks, and is finitely branching and inertly silent.

Example 6.44. $\mathbb{A}$ is a nice alphabet, but $\mathbb{IQ}$ is not, because it is not inertly silent.

The main property of A-trees is:

Proposition 6.45. *Consider any modular G and adequate $p{:}\, G \to A$, where G has enough A-trees and A is a nice alphabet. Then, $\sim_f^G = \sim_f^{G|A}$.*

We start with some preparation. Let a path in A be *loud* iff it contains no silent (=identity if A is inertly silent) edge, and $\heartsuit$*-free* iff no edge is in $(\ell^A)^{-1}(\heartsuit)$. Let the set $\mathcal{F}_a$ of *failures* over $a \in A$ consist of all pairs (p, L), where $p{:}\, a' \to^\star a$ is any loud, $\heartsuit$-free path in A and $L \subseteq A^\star$ is a set of loud paths such that for all $q \in L$, $\mathrm{cod}(q) = a'$.

We define a map $\mathbf{fl}{:}\, \mathcal{F}_a \to \mathcal{H}_a^A$ to A-trees over a, for all a, by induction on p, followed by coinduction on L:

$$\begin{array}{rcl} (e \circ p, L) & \mapsto & e.(\mathbf{fl}(p,L)) + \heartsuit_a.0 \\ (\epsilon, L) & \mapsto & \mathbf{fl}(L) \\ L & \mapsto & \sum_{\{e \in A(-,a) \mid L \cdot e \neq \varnothing\}} e.\mathbf{fl}(L \cdot e) \end{array}$$

where $L \cdot e$ is the set of paths p such that $(e \circ p) \in L$. Note in particular that if $L = \varnothing$ or $\{\epsilon\}$, then $\mathbf{fl}(L) = 0$. The sum is finite at each stage because A is finitely branching, and we use the fact that A has enough ticks.

Proof of Proposition 6.45. It is straightforward to show that $\sim_f^G \subseteq \sim_f^{G|A}$, by Proposition 6.30. For the converse, assume $x \bowtie y$ and $x \not\sim_f^G y$. This means that there exists z such that $x \subset z$ and $y \subset z$, and, w.l.o.g., $(y,z) \in \perp^{G \diamond_A G}$ and $(x,z) \notin \perp^{G \diamond_A G}$.

By the latter, we obtain a transition sequence $(x,z) \Leftarrow (x',z')$, such that for no (x'',z'') we have $(x',z') \overset{\heartsuit}{\Leftarrow} (x'',z'')$. Let r be the given path witnessing $(x,z) \Leftarrow (x',z')$. Its second projection $(\pi')^\star(r)$ is mapped by $p^\star$ to a path in A, from which we remove all identity edges (which are also all silent edges by A being inertly silent) to obtain $\rho = \widetilde{(p \circ \pi')^\star}(r)$, a loud, $\heartsuit$-free path in A. Further let $L \subseteq A^\star$ be the set of all $\widetilde{p^\star(r')}$ for paths $r'{:}\, z' \leftarrow^\star z''$. Let $t = \mathbf{fl}(\rho, L)$. We show $(x,t) \notin \perp^{G \diamond_A \mathcal{H}^A}$ and $(y,t) \in \perp^{G \diamond_A \mathcal{H}^A}$.

For the first point, $t \overset{\rho}{\Leftarrow} t'$, with $t' = \mathbf{fl}(\epsilon, L)$, hence $(x,t) \Leftarrow (x',t')$, by Proposition 6.37. Now, assume $(x',t') \overset{\heartsuit}{\Leftarrow} b''$. By definition of $G \diamond_A \mathcal{H}^A$, we split this into $x' \overset{\rho_1}{\Leftarrow} x''$ and $t' \overset{\rho_2}{\Leftarrow} t''$, with $b'' = (x'', t'')$. But then $z' \overset{\rho_2}{\Leftarrow} z''$ by construction of t, and hence $(x',z') \overset{\heartsuit}{\Leftarrow} (x'',z'')$ (by Proposition 6.37), contradicting $(x,z) \notin \perp^{G \diamond_A G}$.

Let us now show $(y,t) \in \perp^{G \diamond_A \mathcal{H}^A}$. For any $(y,t) \Leftarrow (y',t')$, we have accordingly $t \overset{\rho'}{\Leftarrow} t'$. By construction, ρ' is in the prefix closure of $\rho \circ L$,i.e.,

$$\rho' \in \{r \in A^\star \mid \exists r' \in A^\star, l \in L, r \circ r' = \rho \circ l\}.$$

- If ρ' is a strict prefix of ρ, then by construction $t' \overset{\heartsuit}{\Leftarrow} 0$ and we are done by Proposition 6.37, since $(id, \heartsuit) \rhd^{\mathbb{A}} \heartsuit$.
- Otherwise, ρ is a prefix of ρ'. Let ρ'' be the unique path such that $\rho' = \rho \circ \rho''$. We have $\rho'' \in L$, hence by construction of L there exists z'' such that $z \overset{\rho}{\Leftarrow} z' \overset{\rho''}{\Leftarrow} z''$, and thus $(y,z) \Leftarrow (y',z'')$, by Proposition 6.37. By $(y,z) \in \perp^{G \diamond_A G}$, there exists $(y',z'') \overset{\heartsuit}{\Leftarrow} (y'',z''')$, which projects to $y' \overset{\rho_y}{\Leftarrow} y''$ and $z'' \overset{\rho_z}{\Leftarrow} z'''$. But then $t' \overset{\rho_z}{\Leftarrow} t''$, hence $(y',t') \overset{\heartsuit}{\Leftarrow} (y'',t'')$, by Proposition 6.37 again, which concludes the proof. □

Corollary 6.46. *For any nice alphabet A, modular G and H, adequate $p : G \to A$ and $q : H \to A$, and relation $R : G \nrightarrow H$ over A such that $R \subseteq \approx_A$, if G and H have enough A-trees and R preserves and reflects $\bowtie$, then for any xRx' and yRy', we have $x \sim_f^G y$ iff $x' \sim_f^H y'$.*

Proof. We have

$$\begin{array}{c} x \sim_f^G y \\ \Updownarrow \text{(by Proposition 6.45)} \\ x \bowtie y \text{ and } \forall v \in p(x)^\frown . \forall t \in \mathcal{H}_v^A . (x,t) \in \perp^{G \diamond_A \mathcal{H}^A} \Leftrightarrow (y,t) \in \perp^{G \diamond_A \mathcal{H}^A} \\ \Updownarrow \text{(by weak bisimilarity over } A) \\ x' \bowtie y' \text{ and } \forall v \in p(x)^\frown . \forall t \in \mathcal{H}_v^A . (x',t) \in \perp^{H \diamond_A \mathcal{H}^A} \Leftrightarrow (y',t) \in \perp^{H \diamond_A \mathcal{H}^A} \\ \Updownarrow \text{(by Proposition 6.45 again)} \\ x' \sim_f^H y'. \end{array}$$

□

6.5. **Main result.** We now provide the missing piece to our main result, and then conclude.

Lemma 6.47. *The graph of $\theta : \mathrm{ob}\, CCS \to \mathrm{ob}\, \mathcal{T}^{\mathbb{IQ}}$ is included in weak bisimilarity over $\mathbb{A}$.*

Proof. We would like, for any $h : I \to X$ and family $P \in \prod_{n \in \mathbb{N}} \prod_{x \in X[n]} CCS_n$, to define a process term $h[P]$ with interface $I(\star)$, which would amount to

$$(|_n |_{x \in X[n]} P_x[l \mapsto x \cdot s_l]),$$

but restricting all channels in $X(\star) \setminus h(I(\star))$. When h is not an inclusion, this is a bit tricky, because in our De Bruijn-like syntax $\Gamma \vdash \nu.P$ may be understood as $\Gamma \vdash \nu(\Gamma + 1).P$. That is, ν-bound channels are always strictly greater than names in Γ.

The correct way of doing this is to use subtraction, i.e., restrict channels in $X(\star) - I(\star)$, and accordingly rename channels in the body. Formally, let γ_h be the unique non-decreasing isomorphism $(X(\star) \setminus h(I(\star))) \to (X(\star) - I(\star))$ (which exists thanks to h being monic), and let $h[P]$ be

$$I(\star) \vdash \nu^{X(\star) - I(\star)} . \left(|_n |_{x \in X[n]} P_x \left[\begin{array}{ll} l \mapsto \epsilon a . (h_\star(a) = x \cdot s_l) & \text{if } x \cdot s_l \in h_\star(I(\star)) \\ l \mapsto \gamma_h(x \cdot s_l) & \text{otherwise} \end{array} \right] \right),$$

where ϵ is Hilbert's definite description operator, i.e., $\epsilon a . A(a)$ denotes the unique a such that $A(a)$ holds, and $\nu^n.P$ denotes $\nu. \ldots . \nu.P$, n times.

Definition 6.48. Let $\mathcal{I} : \mathrm{ob}\, CCS \nrightarrow \mathrm{ob}\, \mathcal{T}^{\mathbb{IQ}}$ consist, for any $P \in \prod_{n \in \mathbb{N}} \prod_{x \in X[n]} CCS_n$, of all pairs $(h[P], (I, h, \theta(P)))$.

Let $\mathcal{R}$ be the composite relation

$$\mathrm{ob}\, CCS \xrightarrow{\equiv} \mathrm{ob}\, CCS \xrightarrow{\mathcal{I}} \mathrm{ob}\, \mathcal{T}^{\mathbb{IQ}}.$$

We show that $\mathcal{R}$ is an expansion [51, Chapter 6], which implies that it is a weak bisimulation. Hence, since the graph of θ is included in $\mathcal{R}$, this entails the desired result. Let $x \overset{\widehat{\alpha}}{\leftarrow} x'$ iff

- either α is an identity and $x \overset{x}{\Longleftarrow}{}' $ in zero or one step,
- or α is not an identity and $x \overset{\alpha}{\Longleftarrow} x'$.

Recall:

Definition 6.49. $\mathcal{R}$ is an expansion iff for all $P\,\mathcal{R}\,T$,

- if $P \overset{\alpha}{\leftarrow} P'$, then there exists T' such that $P'\,\mathcal{R}\,T'$ and $T \overset{\alpha}{\Longleftarrow} T'$; and
- if $T \overset{\alpha}{\leftarrow} T'$, then there exists P' such that $P \overset{\widehat{\alpha}}{\leftarrow} P'\,\mathcal{R}\,T'$.

First, one easily shows that transitions in CCS are dealt with by 'heating' the right-hand side until it may match the given transition.

Conversely, we show below in (1) that for any transition $(I,h,\theta(P)) \overset{M}{\leftarrow} (I,k,T')$, for $M\colon k \to h$ in $\mathbb{IQ}$, where M is either a fork or a channel creation, then $T' = \theta(P')$, for some $P' \in \prod_{n\in\mathbb{N}} \prod_{y\in Y[n]} CCS_n$, and $h[P] \equiv k[P']$.

Thus, any such transition, which is silent, is matched by the empty transition sequence, as in

$$\begin{array}{ccccc} Q & \equiv & h[P] & \mathcal{I} & (I,h,\theta(P)) \\ \| & & \| & & \uparrow M \\ Q & \equiv & k[P'] & \mathcal{I} & (I,k,T'). \end{array}$$

Similarly, for any transition $(I,h,\theta(P)) \overset{M}{\leftarrow} (I,k,T')$ not falling in the previous cases, we prove below in (2) that there exists $P' \in \prod_{n\in\mathbb{N}} \prod_{y\in Y[n]} CCS_n$ and Q' such that $h[P] \xleftarrow{\xi(M)} Q' \equiv k[P']$. Thus, any such transition is matched as in

$$\begin{array}{ccccccc} & & Q & \equiv & h[P] & \mathcal{I} & (I,h,\theta(P)) \\ & {\scriptstyle\xi(M)}\nearrow & & {\scriptstyle\xi(M)}\nearrow & & & \uparrow M \\ Q'' & \equiv & Q' & \equiv & k[P'] & \mathcal{I} & (I,k,T'), \end{array}$$

where Q'' is obtained by $\equiv$ being a bisimulation.

(1) As announced, let us now consider the case of a transition $(I,h,\theta(P)) \overset{M}{\leftarrow} (I,k,T')$, for $M\colon k \to h$ in $\mathbb{IQ}$, where M is either a fork or a channel creation. Consider first the case where M is a fork. Let $x_1,\ldots,x_n$ be the players of X, let $m_1,\ldots,m_n$ be their respective arities, and let $i_0 \in n$ be the forking player. Let, for any $i \in n+1$,

$$\mu(i) = \begin{cases} i & \text{if } i < i_0 \\ i_0 & \text{if } i = i_0 \text{ or } i = i_0+1 \\ i-1 & \text{if } i > i_0+1 \end{cases}$$

and

$$P'_i = \begin{cases} P_i & \text{if } i < i_0 \\ P^1_{i_0} & \text{if } i = i_0 \\ P^2_{i_0} & \text{if } i = i_0 + 1 \\ P_{i-1} & \text{if } i > i_0 + 1, \end{cases}$$

where $P_{i_0} = P^1_{i_0} | P^2_{i_0}$. For all $j \in n+1$, we have that y_j is an avatar of $x_{\mu(j)}$ (i.e., $x_{\mu(j)} = (y_j)^M$), and $P'_j = P_{\mu(j)}$ if $\mu(j) \neq i_0$, while $P_{i_0} = P'_{i_0} \mid P'_{i_0+1}$.

Thanks to the restriction of edges

$$\begin{array}{ccccc} & & I & & \\ & {\scriptstyle h}\swarrow & \downarrow{\scriptstyle u} & \searrow{\scriptstyle k} & \\ X & \xrightarrow[t]{} & M & \xleftarrow[s]{} & Y \end{array}$$

in $\mathbb{IQ}$, for any $j \in n+1$, if $\mu(j) = i$, $l \in m_i$ and $a, b \in I(\star)$, we have that if $h_\star(a) = x_i \cdot s_l$ and $k_\star(b) = y_j \cdot s_l$, then, since $s \circ y_j \circ s_l = t \circ x_i \circ s_l$, both squares

$$\begin{array}{ccc} \star & \xrightarrow{a,b} & I \\ {\scriptstyle s_l}\downarrow & & \downarrow{\scriptstyle u} \\ [m_i] & \xrightarrow[s\circ y_j, t\circ x_i]{} & M \end{array}$$

commute, hence $a = b$ by monicity of u.

So, for all $j \in n+1$ and $l \in m_i$, for $i = \mu(j)$, we have $x_i \cdot s_l \in h_\star(I(\star))$ iff $y_j \cdot s_l \in k_\star(I(\star))$, in which case

$$\epsilon a.(h_\star(a) = x_i \cdot s_l) = \epsilon b.(k_\star(b) = y_j \cdot s_l).$$

Furthermore, we have a commuting diagram

$$\begin{array}{ccccc} & & \overset{\delta_M}{\frown} & & \\ X(\star) \setminus h(I(\star)) & \xrightarrow{\cong} & M(\star) \setminus u(I(\star)) & \xleftarrow{\cong} & Y(\star) \setminus k(I(\star)) \\ {\scriptstyle \gamma_h}\downarrow & & & & \downarrow{\scriptstyle \gamma_k} \\ X(\star) - I(\star) & & \xrightarrow[\delta'_M]{} & & Y(\star) - I(\star), \end{array}$$

of bijections, where δ_M and δ'_M are obtained by composition and the arrows marked $\cong$ are the respective restrictions of t and s. This diagram is such that for all $j \in n+1$ and $i = \mu(j)$, $l \in m_i$, if $x_i \cdot s_l \notin h(I(\star))$, then $\delta_M(x_i \cdot s_l) = y_j \cdot s_l$. We have

$$h[P] = \nu^{X(\star)-I(\star)} \cdot \left(|_{i \in n} P_i \left[\begin{array}{ll} l \mapsto \epsilon a.(h_\star(a) = x_i \cdot s_l) & \text{if } x_i \cdot s_l \in h_\star(I(\star)) \\ l \mapsto \gamma_h(x_i \cdot s_l) & \text{otherwise} \end{array} \right] \right)$$

and

$$k[P'] = \nu^{Y(\star)-I(\star)} \cdot \left(|_{j \in n+1} P'_j \left[\begin{array}{ll} l \mapsto \epsilon b.(k_\star(b) = y_j \cdot s_l) & \text{if } y_j \cdot s_l \in k_\star(I(\star)) \\ l \mapsto \gamma_k(y_j \cdot s_l) & \text{otherwise} \end{array} \right] \right).$$

Via the renaming δ'_M, we have

$$\begin{array}{rcl} h[P] & \equiv & \nu^{Y(\star)-I(\star)}.\left(|_{j\in n+1, j\neq i_0+1} P_{\mu(j)} \left[\begin{array}{ll} l \mapsto \epsilon b.(k_\star(b) = y_j \cdot s_l) & \text{if } y_j \cdot s_l \in k_\star(I(\star)) \\ l \mapsto \gamma_k(\delta_M(x_i \cdot s_l)) & \text{otherwise} \end{array}\right]\right) \\ & \equiv & \nu^{Y(\star)-I(\star)}.\left(|_{j\in n+1, j\neq i_0+1} P_{\mu(j)} \left[\begin{array}{ll} l \mapsto \epsilon b.(k_\star(b) = y_j \cdot s_l) & \text{if } y_j \cdot s_l \in k_\star(I(\star)) \\ l \mapsto \gamma_k(y_j \cdot s_l)) & \text{otherwise} \end{array}\right]\right) \\ & \xleftarrow{id} & \nu^{Y(\star)-I(\star)}.\left(|_{j\in n+1} P'_j \left[\begin{array}{ll} l \mapsto \epsilon b.(k_\star(b) = y_j \cdot s_l) & \text{if } y_j \cdot s_l \in k_\star(I(\star)) \\ l \mapsto \gamma_k(y_j \cdot s_l)) & \text{otherwise} \end{array}\right]\right) \\ & \equiv & k[P']. \end{array}$$

The case of a channel creation move is similar.

(2) Consider now any transition $(I, h, \theta(P)) \xleftarrow{M} (I, k, T')$, where M is an input or an output on some channel $c \in h_\star(I(\star))$, or a synchronisation, or a tick. Then, proceeding as for the forking move above, we may take $\mu = id$, and still obtain δ_M and δ'_M. In all cases, we have $T'_i = \theta(P'_i)$, for some family P' of CCS processes. E.g., if M is an input on c by x_{i_0}, then $P'_i = P_i$ for all $i \neq i_0$, and $P_{i_0} \equiv c.P'_{i_0} + P''$. We have $h[P] \xleftarrow{\xi(M)} Q$, where

$$Q = \nu^{X(\star)-I(\star)}.\left(|_{i\in n} P'_i \left[\begin{array}{ll} l \mapsto \epsilon a.(h_\star(a) = x_i \cdot s_l) & \text{if } x_i \cdot s_l \in h_\star(I(\star)) \\ l \mapsto \gamma_h(x_i \cdot s_l) & \text{otherwise} \end{array}\right]\right),$$

which via the renaming δ'_M, is structurally congruent to

$$\begin{array}{l} \nu^{Y(\star)-I(\star)}.\left(|_{i\in n} P'_i \left[\begin{array}{ll} l \mapsto \epsilon b.(k_\star(b) = y_i \cdot s_l) & \text{if } y_i \cdot s_l \in k_\star(I(\star)) \\ l \mapsto \gamma_k(\delta_M(x_i \cdot s_l)) & \text{otherwise} \end{array}\right]\right) \\ \equiv \nu^{Y(\star)-I(\star)}.\left(|_{i\in n} P'_i \left[\begin{array}{ll} l \mapsto \epsilon b.(k_\star(b) = y_i \cdot s_l) & \text{if } y_i \cdot s_l \in k_\star(I(\star)) \\ l \mapsto \gamma_k(y_i \cdot s_l) & \text{otherwise} \end{array}\right]\right) \\ \equiv k[P'], \end{array}$$

which concludes the proof. □

This leads to our first full abstraction result:

Corollary 6.50. *The composite* $\mathrm{ob}(CCS) \hookrightarrow \mathrm{ob}(\mathcal{T}^{\mathbb{IQ}}) \to \mathrm{ob}(\mathcal{S}^{\mathbb{IQ}})$ *is included in weak bisimilarity.*

Proof. By the previous lemma, Proposition 6.38, and the fact that weak bisimulations are closed under composition. □

Corollary 6.51. *The composite* $\mathrm{ob}\, CCS \xrightarrow{\theta} \mathrm{ob}\, \mathcal{T}^{\mathbb{IQ}} \xrightarrow{[\![-]\!]} \mathrm{ob}\, \mathcal{S}^{\mathbb{IQ}}$ *is fair, and we have for all CCS processes* P *and* Q *over any common* n*:*

$$P \sim_{f,s} Q \qquad \textit{iff} \qquad [\![\theta(P)]\!] \sim_f [\![\theta(Q)]\!].$$

Proof. We have:

$$\begin{array}{c} P \sim_{f,s} Q \\ \Updownarrow \text{(by Proposition 6.21)} \\ P \sim_f^{CCS} Q \\ \Updownarrow \text{(by Corollaries 6.46 and 6.50, and Example 6.41)} \\ [\![\theta(P)]\!] \sim_f^{\mathcal{S}^{\mathbb{IQ}}} [\![\theta(Q)]\!] \\ \Updownarrow \text{(by Corollary 6.26)} \\ [\![\theta(P)]\!] \sim_f [\![\theta(Q)]\!], \end{array}$$

as desired. □

7. CCS as a playground

At last, we prove that $\mathbb{D}^{CCS}$ forms a playground. We rewind to the beginning of Section 4.1, to state things a bit more formally.

7.1. **A pseudo double category.** Recall from HP the notion of dimension in $\mathbb{C}$: $\star$ is the sole object of dimension 0, all $[n]$'s have dimension 1, all $o_{n,i}$, $\iota_{n,i}$, π^l_n, π^r_n, $\heartsuit_n$, and ν_n have dimension 2, all π_n have dimension 3, and all $\tau_{n,i,m,j}$ have dimension 4. By extension, a presheaf F has dimension i if F is empty over objects of dimension strictly greater than i. We call *interfaces* the presheaves of dimension 0 (i.e., empty beyond dimension 0), *positions* the finite presheaves of dimension 1.

We start by viewing the base pseudo double category of our playground, $\mathbb{D}^{CCS}$, as a sub-pseudo double category of the following pseudo double category $\mathbb{D}^{CCS,0}$.

Definition 7.1. Let $\mathbb{D}^{CCS,0}$ have:

- as objects all positions,
- horizontal category $\mathbb{D}^{CCS,0}_h$ the subcategory of $\widehat{\mathbb{C}^f}$ consisting of positions and monic arrows between them;
- vertical (bi)category $\mathbb{D}^{CCS,0}_v$ the sub-bicategory of $\mathsf{Cospan}(\widehat{\mathbb{C}^f})$ consisting of positions and cospans of monic arrows between them;
- and all commuting diagrams

$$\begin{array}{ccc} X & \hookrightarrow^{h} & X' \\ {\scriptstyle s}\downarrow & & \downarrow{\scriptstyle s'} \\ U & \hookrightarrow_{k} & V \\ {\scriptstyle t}\uparrow & & \uparrow{\scriptstyle t'} \\ Y & \underset{l}{\hookrightarrow} & Y' \end{array} \quad \text{as double cells} \quad \begin{array}{ccc} X & \xrightarrow{h} & X' \\ {\scriptstyle U}\downarrow & \overset{(h,k,l)}{\Longrightarrow} & \downarrow{\scriptstyle V} \\ Y & \xrightarrow[l]{} & Y', \end{array} \tag{7.1}$$

  with all $\hookrightarrow$ arrows monic.

Horizontal composition of double cells is induced by composition in $\widehat{\mathbb{C}^f}$. Vertical composition of double cells is induced by pushout in $\widehat{\mathbb{C}^f}$. It is of course the vertical direction here which is pseudo.

Proposition 7.2. *$\mathbb{D}^{CCS}$ is the pseudo double category obtained by restricting $\mathbb{D}^{CCS,0}$ to vertical morphisms which are either equivalences or finite composites of moves.*

Since $\mathbb{D}^{CCS}$ is again the only involved (candidate) playground in this section, we often omit the superscript. E.g., $\mathbb{D}^0$ denotes $\mathbb{D}^{CCS,0}$.

The rest of Section 7 is devoted to proving:

Theorem 7.3. *$\mathbb{D}$, equipped with*

- *as individuals, all positions of the shape $\mathbb{C}(-,[n])$, i.e., all strictly representable presheaves,*
- *moves as moves, seeds as basic moves, and full moves as full moves.*

forms a playground.

We start with a combinatorial correctness criterion for characterising plays $U : X \nrightarrow Y$ among general cospans $X \hookrightarrow U \hookleftarrow Y$, which we then put to use in proving the theorem. Our convention for plays $X \hookrightarrow U \hookleftarrow Y$ is that the (candidate) final position is always on the left.

7.2. **Correctness.** We prove a few properties of plays, which we then find are sufficient for a cospan to be a play.

Given a play $X \hookrightarrow U \hookleftarrow Y$, we start by forgetting the cospan structure and exhibiting some properties of U alone.

Definition 7.4. A *core* of a presheaf $U \in \widehat{\mathbb{C}^f}$ is an element of dimension > 1 which is not the image (under the action of some morphism of $\mathbb{C}$) of any element of higher dimension.

Here is a first easy property of plays. Observing that for all seeds $Y \hookrightarrow M \hookleftarrow X$, M is a representable presheaf, we put:

Definition 7.5. A presheaf U is *locally 1-injective* iff for any seed $Y \hookrightarrow M \hookleftarrow X$ with interface I and core $\mu \in U(M)$, if two elements of M are identified by the Yoneda morphism $\mu : M \to U$, then they are in (the image of) $I(\star)$.

The name 'locally 1-injective' is designed to evoke the fact that $M \to U$ is injective above dimension 0.

Proposition 7.6. *Any play U is locally 1-injective.*

Proof. Choose a decomposition of U into moves; μ corresponds to precisely one such move, say M', obtained, by definition, from some seed M as a pushout (3.1). By construction of pushouts in presheaf categories, M' is obtained from M by identifying some channels according to $I \to Z$. □

We now extract from any presheaf a graph, which represents its candidate causal structure. Observe that, in $\mathbb{C}$, for any object μ of dimension > 1 (i.e., a move), all morphisms from a player, i.e., an object of the shape $[n]$, to μ have exactly one of the shapes $f \circ s \circ f'$ and $f \circ t \circ f'$. In the former case, the given player belongs in the final position of μ and we say that it is a *source* of μ; in the latter, it belongs in the initial position and we call it a *target*. We extend these notions to arbitrary presheaves.

Definition 7.7. In any U, the *sources* of a core μ are the players x with a morphism, in $\int U$ (the category of elements of U, recalled in Section 3.1), of the shape $x \xrightarrow{f \circ s \circ f'} \mu$ to μ; its *targets* are the players y with a morphism of the shape $y \xrightarrow{f \circ t \circ f'} \mu$.

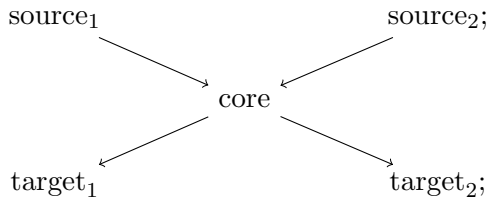
Example 7.8. In the representable π_n, there is one target, $l \circ t$ (or equivalently $r \circ t$), and two sources, $s_1 = l \circ s$ and $s_2 = r \circ s$, respectively the left and right players obtained by forking. Another example is $\tau_{n,i,m,j}$, which has two targets, the sender $\epsilon \circ t$ and the receiver $\rho \circ t$, and two sources $\epsilon \circ s$ and $\rho \circ s$.

Definition 7.9. A channel $a \in M(\star)$ is *created* by a seed $Y \overset{s}{\hookrightarrow} M \overset{t}{\hookleftarrow} X$ iff $a \in Y(\star) \setminus X(\star)$.

Recall that in $\mathbb{C}$, the channels known to a player $[n]$ are represented by morphisms $s_1, \ldots, s_n : \star \to [n]$, so that in a presheaf $U \in \widehat{\mathbb{C}^f}$, the channels known to $x \in U[n]$ are $x \cdot s_1$, $\ldots$, and $x \cdot s_n$.

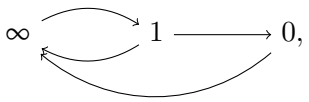
Given a presheaf U, we construct its *causal (simple) graph* G_U as follows:

- its vertices are all channels, players, and cores in U;
- there is an edge to each core from its sources and one from each core to its targets, as in



- there is an edge $x \to x \cdot s_i$ for all $x \in U[n]$ and $i \in n$;
- there is an edge $a \to \mu$ for each channel a created by μ.

This graph is actually a binary relation, since there is at most one edge between any two vertices. It is also a coloured graph, in the sense that it comes equipped with a morphism to the graph L:



mapping cores to ∞, players to 1, and channels to 0. (Observe in particular that there are no edges from channels to players nor from cores to channels.) For any simple graph G, equipped with a morphism $l\colon G \to L$, we call vertices of G channels, players, or cores, according to their label.

Definition 7.10. Seen as an object of Gph/L, G is *source-linear* iff for any cores μ, μ', and other vertex (necessarily a player or a channel) x, $\mu \leftarrow x \to \mu'$ in G, then $\mu = \mu'$. G is *target-linear* iff for any cores μ, μ' and player x, if $\mu \to x \leftarrow \mu'$ in G, then $\mu = \mu'$. G is *linear* iff it is both source-linear and target-linear.

Proposition 7.11. *For any play $Y \overset{s}{\hookrightarrow} U \overset{t}{\hookrightarrow} X$, G_U is linear.*

Proof. By induction on any decomposition of U into moves. □

Proposition 7.12. *For any play as above, G_U is acyclic (in the directed sense).*

Proof. Again by induction on any decomposition of U. □

Definition 7.13. A player x in U is *final* iff it is not the target of any move, i.e., for no move $\mu \in U$, $x = \mu \cdot t$.

Lemma 7.14. *A player is final in U iff it has no edge from any core in G_U.*

Definition 7.15. A player is *initial* in U when it is not the source of any move, i.e., for no move $\mu \in U$, $x = \mu \cdot s$. A channel is initial when it is not created by any move.

Lemma 7.16. *A player is initial in U iff it has no edge to any core in G_U.*

Now, here is the expected characterisation:

Theorem 7.17. *A cospan $Y \overset{s}{\hookrightarrow} U \overset{t}{\hookrightarrow} X$ is a play iff*

(i) U is locally 1-injective,

(ii) X contains precisely the initial players and channels in U,

(iii) *Y contains all channels, plus precisely the final players in U,*
(iv) *and G_U is linear and acyclic.*

Of course, we have almost proved the 'only if' direction, and the rest is easy, so only the 'if' direction remains to prove. The rest of this section is devoted to this. First, let us familiarise ourselves with removing elements from a presheaf. For two morphisms of presheaves $U \xrightarrow{f} V \xleftarrow{g} W$, we denote by $U \setminus W$ the topos-theoretic difference $U \cap \neg W$ of (the images of) f and g in the lattice $\mathrm{Sub}(V)$ of subobjects of V. This differs in general from what we denote $U - W$, which is the set of elements in V which are in the image of U but not that of W, i.e., $\sum_{c\in\mathbb{C}} U(c) \setminus W(c)$. More generally, for any morphism of presheaves $f\colon U \to V$ and set W, let $U - W = \sum_{c\in\mathbb{C}} \mathrm{Im}(U(c)) \setminus W$. $U - W$ is generally just a set, not a presheaf; i.e., its elements are not necessarily stable under the action of morphisms in $\mathbb{C}$. Proposition 7.19 below exhibits a case where they are, which is useful to us.

Definition 7.18. For any seed $Y \hookrightarrow M \hookleftarrow X$, let the *past* $\mathrm{past}(M) = M - Y$ of M be the set of its elements not in the image of Y. For any such M, presheaf U, and core $\mu \in U(M)$, let $\mathrm{past}(\mu) = \mathrm{Im}(\mathrm{past}(M))$ consist of all images of $\mathrm{past}(M)$.

To explain the statement a bit more, by Yoneda, we see μ as a map $M \to U$, so we have a set-function

$$\mathrm{past}(M) \hookrightarrow \textstyle\int M \to \int U.$$

Observe that $\mathrm{past}(\mu)$ is always a set of players and moves only, since channels present in X always are in Y too.

Given a core $\mu \in U$, an important operation for us will be

$$U \setminus\!\setminus \mu = \bigcup\{V \hookrightarrow U \mid \textstyle\int V \cap \mathrm{past}(\mu) = \varnothing\}.$$

$U \setminus\!\setminus \mu$ is thus the largest subpresheaf of U not containing any element of the past of μ. The good property of this operation is:

Proposition 7.19. *If μ is a maximal core in G_U (i.e., there is no path to any further core) and G_U is target-linear, then $U \setminus\!\setminus \mu = U - \mathrm{past}(\mu)$, i.e., $(U \setminus\!\setminus \mu)(c) = U(c) \setminus \mathrm{past}(\mu)$ for all c.*

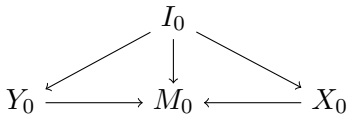
Proof. The direction $(U \setminus\!\setminus \mu)(c) \subseteq U(c) \setminus \mathrm{past}(\mu)$ is by definition of $\setminus\!\setminus$. Conversely, it is enough to show that $c \mapsto U(c) \setminus \mathrm{past}(\mu)$ forms a subpresheaf of U, i.e., that for any $f\colon c \to c'$ in $\mathbb{C}$, and $x \in U(c') \setminus \mathrm{past}(\mu)$, $x \cdot f \notin \mathrm{past}(\mu)$. Assume on the contrary that $x' = x \cdot f \in \mathrm{past}(\mu)$. Then, of course f cannot be the identity. Furthermore, x' is either a player or a move; so, up to pre-composition of f with a further morphism, we may assume that x' is a player. But then, since f is non-identity, x must be a move, with x' being one of its sources or targets. Now, up to post-composition of f with a further morphism, we may assume that x is a core. So, there is either an edge $x \to x'$ or an edge $x' \to x$ in G_U. However, $x \neq \mu$, so $x \to x'$ is impossible by target-linearity of G_U, and $x' \to x$ is impossible by maximality of μ. □

Proof of Theorem 7.17. We proceed by induction on the number of moves in U. If it is zero, then U is a position; by *(ii)*, t is an iso, and by *(iii)* so is s, hence the cospan is a play. For the induction step, we first decompose U into

$$Y \xhookrightarrow{s_2} U' \xhookrightarrow{t_2} Z \xhookrightarrow{s_1} M' \xhookrightarrow{t_1} X,$$

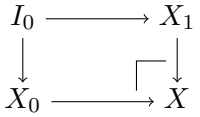
and then show that M' is a move and U' satisfies the conditions of the theorem.

So, first, pick a maximal core μ in G_U, i.e., one with no path to any other core. Let

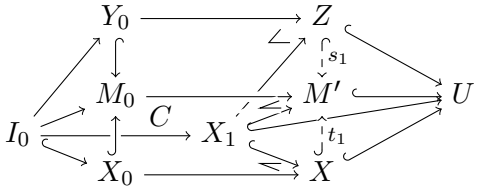


be the seed with interface corresponding to μ, so we have the Yoneda morphism $\mu : M_0 \to U$.

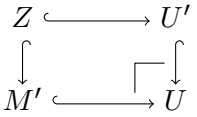
Let $U' = (U \setminus\!\setminus \mu)$, and $X_1 = X - \mathrm{Pl}(X_0)$. X_1 is a subpresheaf of X, since it contains all names. The square



is a pushout, since it just adds the missing players to X_1. Define now Z, M', s_1, and t_1 by the pushouts



and the induced arrows. We further obtain arrows to U by universal property of pushout, which are monic because $X \hookrightarrow U$ is, using *(i)*. We observe that $U = M' \cup U'$, i.e., the square



is a pushout, so U is indeed a composite as claimed, with $Z \hookrightarrow M' \hookleftarrow X$ a move by construction. So, it remains to prove that $Y \hookrightarrow U' \hookleftarrow Z$ satisfies the conditions. First, as a subpresheaf of U, U' is locally 1-injective and has a linear and acyclic causal graph, so satisfies *(i)* and *(iv)*. U' furthermore satisfies *(ii)* by construction of Z and source-linearity of G_U, and *(iii)* because removing $\mathrm{past}(\mu)$ cannot make any non-final player final. □

7.3. **CCS as a pre-playground.** We now start proving:

Theorem 7.20. $\mathbb{D}$ *forms a playground.*

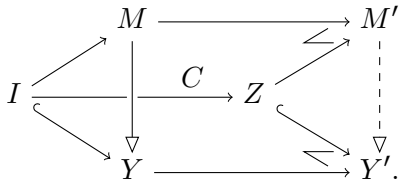
Axioms (P2)–(P4) are easy, as well as (P6), (P9) and (P10). Furthermore, once (P1) is clear, (P5) is also easy. This leaves (P1) and the decomposition axioms.

For (P1), i.e., the fact that $\mathrm{cod} : \mathbb{D}_H \to \mathbb{D}_h$ is a fibration, we introduce the notion of 'history' for plays. For a presheaf $U \in \widehat{\mathbb{C}^f}$, let $\llcorner U \lrcorner$ be its restriction to dimension 3, i.e., $\llcorner U \lrcorner(\tau_{n,i,m,j}) = \varnothing$ for all n, i, m, j, and $\llcorner U \lrcorner(c) = U(c)$ on other objects. Further let $\mathrm{El}(U) = \sum_{c \in \mathrm{ob}(\mathbb{C})} \llcorner U \lrcorner(c)$ be the set of elements of $\llcorner U \lrcorner$. We have a category $\mathrm{El}(\widehat{\mathbb{C}^f})$, whose objects are those of $\widehat{\mathbb{C}^f}$, and whose morphisms $U \to U'$ are set-functions $\mathrm{El}(U) \to \mathrm{El}(U')$. We denote such morphisms with special arrows $U \multimap U'$. There is a forgetful functor $\mathrm{El} : \widehat{\mathbb{C}^f} \to \mathrm{El}(\widehat{\mathbb{C}^f})$, which we implicitly use in casting arrows $U \to U'$ to arrows $U \multimap U'$.

Definition 7.21. Consider any seed $X \hookrightarrow M \hookleftarrow Y$ which is not a synchronisation, where Y is the initial position and X is the final position. Then Y is a representable position, say $[n]$, and we let the *history* of M be the map $p_M : \mathrm{El}(M) \to \mathrm{El}(Y)$ sending

- all channels in $\mathrm{El}(M) \cap \mathrm{El}(Y)$ to themselves,
- all other elements to $id_{[n]}$.

The history $p_{M'}$ of a move M' is the map obtained by pushout of the history of its generating seed M, as in



This defines the history of moves. We have:

Proposition 7.22. *For any move $X \overset{s}{\hookrightarrow} M \overset{t}{\hookleftarrow} Y$, we have $p_M \circ t = id$.*

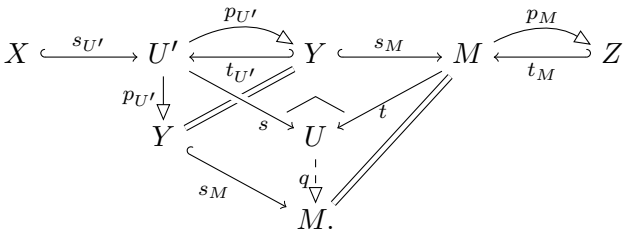
We graphically represent histories by arrows between the presheaves, as p in

$$X \underset{s}{\hookrightarrow} U \underset{t}{\overset{p}{\rightleftarrows}} Y. \tag{7.2}$$

We now define the history of sequences of moves, which we here call *sequential plays*. We denote such a sequence $X_n \xrightarrow{M_n} X_{n-1} \dots X_1 \xrightarrow{M_1} X_0$ by $(M_n, \dots, M_1)$.

Definition 7.23. Define now the history of a sequential play $X \to (M_n, \dots, M_1) \leftarrow Y$, letting $U = M_1 \bullet \dots \bullet M_n$ be the corresponding play, to be the map $U \mathrel{-\!\!\triangleright} Y$ defined by induction on n as follows:

- if $|U| = 0$, then t is an isomorphism, and the history is the inverse of the corresponding bijection on elements;
- if $|U| = 1$, then U is a move M and its history is that of M;
- if $|U| > 1$, then $U = (U', M)$ for some move M and sequential play U'; letting $p_{U'}$ be the history of U' obtained by induction hypothesis, we let $p_U = p_M \circ q$, where q is defined by universal property of pushout in



Proposition 7.24. *For any sequential plays $U_1, U_2 : X \mathrel{-\!\!\bullet\!\!\to} Y$ with isomorphic compositions, we have $p_{U_1} = p_{U_2}$.*

Proof. For any presheaf U such that G_U is source-linear and acyclic, consider the function $h_U : \mathrm{El}(U) \to \mathrm{El}(U)$ mapping

- initial players and channels to themselves,

- non-initial players and channels to the (unique by source-linearity of G_U) core that created them,
- elements of dimension 2 to their image under t,
- elements of higher dimensions to the image of one of their images in dimension 2 (which all map to the same element by a simple case analysis).

Observe that this map is ultimately idempotent because it is strictly increasing w.r.t. G_U, and let H_U be the corresponding idempotent function.

It is easy to see that if $X \hookrightarrow U \hookleftarrow Y$ is a move, then $\mathrm{Im}(H_U) = Y$ and $p_U = H_U$.

Furthermore, for all composable plays $X \overset{U'}{\dashrightarrow} Y \overset{U}{\dashrightarrow} Z$, we have $H_{U \bullet U'} = H_U \circ H^U_{U'}$, where $H^U_{U'} : \mathrm{El}(U \bullet U') \to \mathrm{El}(U)$ is the extension of $H_{U'}$ to $\mathrm{El}(U \bullet U')$ which is the identity on $\mathrm{El}(U) \setminus \mathrm{El}(U')$. Because $\mathrm{Im}(H_{U'}) = Y$, this indeed goes to $\mathrm{El}(U)$.

When U is a move, this is actually equivalent to the diagrammatic definition of $p_{M \bullet U'}$, which entails by induction that for any play U, $p_U = H_U$, which does not depend on the decomposition of U into moves. □

Just as for moves, the target map is a section of the history:

Proposition 7.25. *For any play $X \hookrightarrow U \overset{t}{\hookleftarrow} Y$, we have $p_U \circ t = id_Y$.*

Proposition 7.26. *Any double cell (h, k, l) as on the left below*

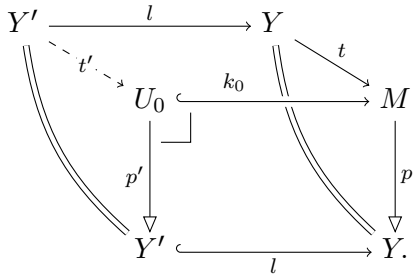
$$\begin{array}{ccc} X & \xrightarrow{h} & X' \\ {\scriptstyle s}\downarrow & & \downarrow{\scriptstyle s'} \\ U & \xrightarrow{k} & V \\ {\scriptstyle t}\uparrow & & \uparrow{\scriptstyle t'} \\ Y & \xrightarrow[l]{} & Y' \end{array} \qquad \begin{array}{ccc} U & \xrightarrow{k} & V \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle p'} \\ Y & \xrightarrow[l]{} & Y' \end{array}$$

is compatible with histories $p : U \twoheadrightarrow Y$ and $p' : U' \twoheadrightarrow Y'$, in the sense that the square on the right commutes.

The important point for us is:

Proposition 7.27. *The vertical codomain functor* $\mathrm{cod} : \mathbb{D}_H \to \mathbb{D}_h$ *is a fibration.*

Proof. We first consider the restriction of cod to the full subcategory of $\mathbb{D}_H$ consisting of moves and isomorphisms. Given a move $X \overset{s}{\hookrightarrow} M \overset{t}{\hookleftarrow} Y$ and a morphism $l : Y' \to Y$ in $\mathbb{D}_h$, consider the pullback (in sets) and the induced arrow t':



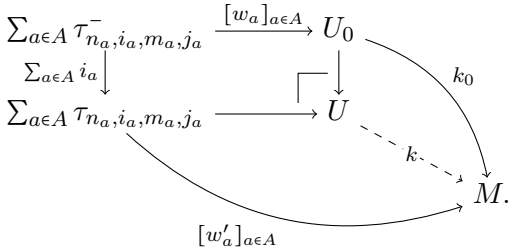
Now, consider U_0 as a presheaf over $\mathbb{C}_3$ by giving each element the type of its image under k_0, and checking that U_0, viewed as an $\mathrm{ob}(\mathbb{C}_3)$-indexed family of subsets of M, is stable

under the action of morphisms in $\mathbb{C}_3$. This, in passing, equips k_0 and t' with the structure of maps in $\widehat{\mathbb{C}}^f$.

Furthermore, let the (n,i,m,j)-*horn* (see, e.g., Joyal and Tierney [29] for the origin of our terminology) $\tau^-_{n,i,m,j}$ be the representable presheaf on $\tau_{n,i,m,j}$, minus the element $id_{\tau_{n,i,m,j}}$, and consider the family A of commuting squares

$$\begin{array}{ccc} \tau^-_{n,i,m,j} & \xrightarrow{w} & U_0 \\ {\scriptstyle i}\downarrow & & \downarrow{\scriptstyle k_0} \\ \tau_{n,i,m,j} & \xrightarrow[w']{} & M, \end{array}$$

where i is the inclusion. Define then U and k by pushout as in



Informally, U is U_0, where we add all the $\tau_{n,i,m,j}$'s that exist in M and whose horn is in U_0. We have by construction $\mathrm{El}(U) = \mathrm{El}(U_0)$, so p' is indeed a left inverse to $t'{:}\mathrm{El}(Y') \to \mathrm{El}(U)$.

Finally, define X', h, and s' by the pullback

$$\begin{array}{ccc} X' & \xrightarrow{h} & X \\ {\scriptstyle s'}\downarrow & & \downarrow{\scriptstyle s} \\ U & \xrightarrow[k]{} & M. \end{array}$$

This altogether yields a vertical morphism

$$X' \underset{s'}{\hookrightarrow} U \underset{t'}{\overset{p'}{\rightleftarrows}} Y',$$

in $\mathbb{D}^0_v$. A tedious case analysis (made less tedious by $l{:}Y' \hookrightarrow Y$ being monic) shows that, because M is a move, U is either a move or isomorphic to Y'. So it is in $\mathbb{D}_v$. U is our candidate cartesian lifting of M along l. More generally, for any play $X \overset{s}{\hookrightarrow} U \overset{t}{\hookrightarrow} Y$, choose a decomposition into moves. We obtain a candidate cartesian lifting $X' \overset{s'}{\hookrightarrow} U' \overset{t'}{\hookrightarrow} Y'$ for U, with morphism (h,k,l) to U, along any $l{:}Y' \hookrightarrow Y$ by taking the successive candidates for each move in the obvious way, and composing them.

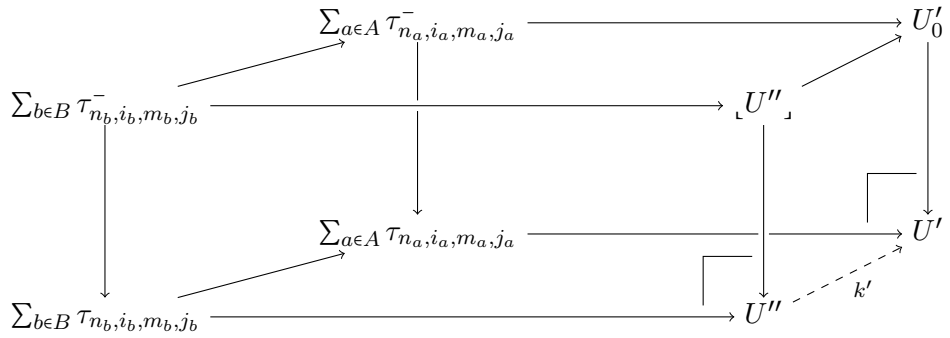
To show that this indeed yields a cartesian lifting, consider any vertical morphism $X'' \overset{s''}{\hookrightarrow} U'' \overset{t''}{\hookrightarrow} Y''$ and diagram

$$\begin{array}{ccc} X'' & \xrightarrow{h''} & X \\ {\scriptstyle s''}\downarrow & & \downarrow{\scriptstyle s} \\ U'' & \xrightarrow[k'']{} & U \\ {\scriptstyle t''}\uparrow & & \uparrow{\scriptstyle t} \\ Y'' & \xrightarrow[l'']{} & Y, \end{array}$$

together with a map $l' : Y'' \to Y'$ such that $l \circ l' = l''$. By Proposition 7.26, letting p'' be the history of U'', the diagram

$$\begin{array}{ccc} U'' & \xrightarrow{k''} & U \\ {\scriptstyle p''}\downarrow & & \downarrow{\scriptstyle p} \\ Y'' & \xrightarrow[l'']{} & Y \end{array}$$

commutes, so by universal property of pullback, we obtain a map $k'_0 : \mathrm{El}(U'') \to \mathrm{El}(U')$, such that $k_0 \circ k'_0 = k''_0$, where k''_0 is the restriction of k'' to dimensions < 4. Furthermore, the expected map $k' : U'' \to U'$, is given by universal property of pushout in



where B is the family of all commuting squares

$$\begin{array}{ccc} \tau^-_{n,i,m,j} & \xrightarrow{w} & \ulcorner U'' \urcorner \\ {\scriptstyle i}\downarrow & & \downarrow{\scriptstyle k_0} \\ \tau_{n,i,m,j} & \xrightarrow[w']{} & U''. \end{array}$$

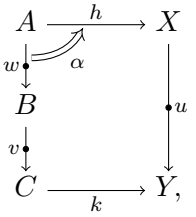
Finally, the desired map $h' : X'' \to X'$ follows from universal property of X' as a pullback, and the square

$$\begin{array}{ccc} U'' & \xrightarrow{k'} & U' \\ {\scriptstyle t''}\uparrow & & \uparrow{\scriptstyle t'} \\ Y'' & \xrightarrow[l']{} & Y' \end{array}$$

commutes by uniqueness in the universal property of U' as a pullback. □

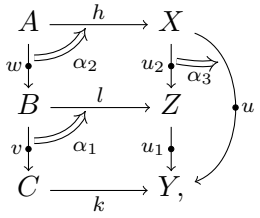
7.4. **Towards CCS as a playground.** In this section, we prove an intermediate result for proving the decomposition axioms.

Consider a double cell α of the shape

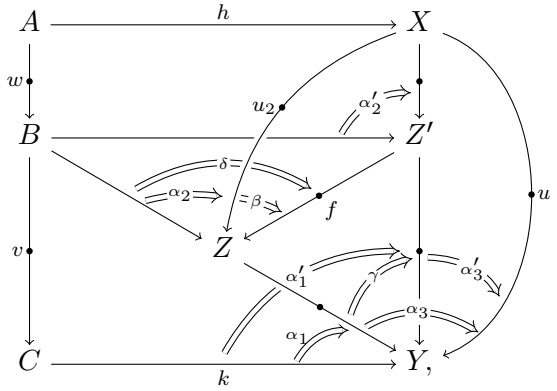


where v is a view. Let now D_α denote the category with

- objects all tuples $T = (Z, l, u_1, u_2, \alpha_1, \alpha_2, \alpha_3)$ such that



  equals α and α_3 is an isomorphism;
- with morphisms $T \to T'$ given by tuples $(U, f, \beta, \gamma, \delta)$ (where f is vertical) such that



  commutes, i.e., $\gamma \circ (\alpha_1 \bullet \delta) = \alpha'_1$, $\beta \circ \alpha_2 = \delta \bullet \alpha'_2$, and $\alpha'_3 \circ (\gamma \bullet u'_2) \circ (u_1 \bullet \beta) = \alpha_3$, and β and γ are isomorphisms;
- composition and identities are obvious.

So, objects of D_α are decompositions of u permitting corresponding decompositions of α. The rest of this section is a proof of:

Lemma 7.28. *D_α has a weak initial object, i.e., an object T such that for any object T' there is a morphism $T \to T'$.*

We start by extending the assignment $U \mapsto G_U$ to a functor, at least for source-linear U. Let SLin denote the full subcategory of $\widehat{\mathbb{C}}$ spanning source linear presheaves. The assignment $U \mapsto G_U$ actually extends to a functor $G_{\_} : \mathsf{SLin} \to \mathsf{Gph}/L$, as follows. Let, first, for any move $x \in U$, the *core associated to* x, $\mathrm{core}(x)$, be the unique core reachable from x in $\int U$, i.e., the unique core μ for which there exists f in $\mathbb{C}$ such that $\mu \cdot f = x$. Now, for any $\alpha : U \to U'$ in $\widehat{\mathbb{C}}$, let $G_\alpha : G_U \to G_{U'}$ map any core x in G_U to $\mathrm{core}(\alpha(x)) \in G_{U'}$, and any non-core vertex

$x \in G_U$ to $\alpha(x) \in G_{U'}$. By naturality, this indeed defines a unique morphism of simple graphs over L.

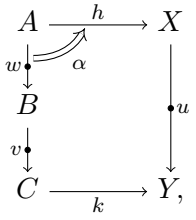
Proposition 7.29. *$G_-\colon \mathsf{SLin} \to \mathsf{Gph}/L$ is a functor.*

We continue with some properties of $\mathbb{D}$.

Definition 7.30. A *filiform* play is any play U such that the restriction of G_U to cores and players is a filiform graph, i.e., a graph of the shape $\cdot \to \cdot \to \cdots$

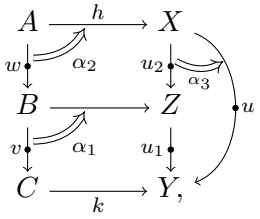
E.g., all views are filiform.

Lemma 7.31. *Any epimorphic (in $\mathbb{D}_H$, hence isomorphic) double cell*



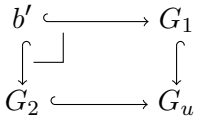
(7.3)

where v is filiform decomposes as



with α_3 an isomorphism, α_1 and α_2 epimorphic, uniquely up to isomorphism. In this case, u_1 is filiform.

Proof. B has just one player, say b. Let $b' = \alpha(b)$. Because α is epi, α induces a morphism $G_\alpha\colon G_{v \bullet w} \to G_u$ of graphs, which is also epi. So, G_u may be decomposed as a pushout



with $G_1 = \mathrm{Im}_{G_\alpha}(G_v)$ and $G_2 = \mathrm{Im}_{G_\alpha}(G_w)$. From this one deduces a decomposition of u and α. □

Lemma 7.32. *For any vertically composable α and β, if $\alpha \bullet \beta$ is epi, then so are α and β.*

Proof. Easy. □

Proof of Lemma 7.28. The double cell α induces morphisms of graphs $G_v \to G_u \leftarrow G_w$, by Proposition 7.29. Let

$$u_1 = \bigcap\{u' \subseteq u \mid (Y \subseteq u') \wedge (\mathrm{Im}_\alpha(G_v) \subseteq G_{u'})\}.$$

Thus, $v \to u$ factors as $v \to u_1 \to u$. Let Z be the position containing all channels of u_1, and all final players of u_1. Further let $\uparrow Z$ denote the full subgraph of G_u containing all vertices x with a path to some vertex of Z. Let then

$$u_2 = \bigcap\{u'' \subseteq u \mid G_{u''} \supseteq \uparrow Z\}.$$

The union $u_1 \cup u_2$ is u, i.e., the square

$$\begin{array}{ccc} Z & \longrightarrow & u_1 \\ \downarrow & \lrcorner & \downarrow \\ u_2 & \longrightarrow & u \end{array}$$

is a pushout, i.e., $u_2 \bullet u_1 \cong u$ in $\mathsf{Cospan}(\widehat{\mathbb{C}}^f)$. So it only remains to prove that $Z \to u_1 \leftarrow X$ and $Y \to u_2 \leftarrow Z$ are plays, for which we use Theorem 7.17. First, u_1 and u_2, as subpresheaves of u, both are locally 1-injective. Furthermore, G_{u_1} and G_{u_2}, as subgraphs of a linear and acyclic graph, are also linear and acyclic. Now, by definition of Z, $Z \to u_1$ contains all channels and the final players of u_1. Further, since $X \subseteq u_1$, being initial in u implies being initial in u_1, so $Z \to u_1 \leftarrow X$ indeed is a play. Symmetrically, no player of u_1 not in Z is final, so $Y \subseteq u_2$, and hence $Y \to u_2$ indeed contains all channels and final players. Finally, the players and channels of Z are precisely the initial players and channels of u_2.

It remains to show that the induced decomposition of α is weakly initial. But any decomposition, inducing a decomposition $u_1' \bullet u_2'$ of u, should satisfy $Y \subseteq u_1'$, $\mathrm{Im}_\alpha(G_v) \subseteq G_{u_1'}$, and $G_{u_2'} \subseteq \uparrow Z$, so, ignoring isomorphisms for readability, $u_1 \subseteq u_1'$ and $u_2' \subseteq u_2$, as desired. □

7.5. **CCS as a playground.** We are now ready to prove the decomposition axioms, which entail Theorem 7.3. They are proved in Lemmas 7.35 and 7.34 below.

Let us start with the following easy lemma.

Lemma 7.33. *If $u = u_2 \bullet u_1$, then, in G_u*

- *no player of u_1 is reachable from any core of u_2;*
- *no core of u_1 is reachable from any element of u_2.*

Proof. For the first point, cores of u_2 only reach initial channels of u_1.

For the second point, we further observe that channel and players of u_2 only reach initial players and channels of u_1, hence no core. □

The easiest decomposition axiom is (P8).

Lemma 7.34. *$\mathbb{D}$ satisfies (P8).*

Proof. Although the statement is complicated, this is rather easy: α restricts to a map of presheaves $f : b \to (M \bullet u)$, on which we proceed by case analysis.

If $\mathrm{Im}(f) \subseteq M$, then by Lemma 7.28 and correctness we are in the left-hand case. Otherwise, assume that a move $\mu' \in M$ is in the image of α, say of a move $\mu \in w$. We have a path $\mu \to b$ in $b \bullet w$, hence a path $\mathrm{core}(\mu') \to \alpha(b)$ in $M \bullet u$, contradicting Lemma 7.33. □

Let us now attack the last axiom.

Lemma 7.35. *$\mathbb{D}$ satisfies (P7).*

We need a few lemmas.

Lemma 7.36. *For any plays $A \overset{u_1}{\bullet\!\!\to} B \overset{u_2}{\bullet\!\!\to} C$, for any player or channel $x \in u_2$ and core $\mu \in u_1$, there is no edge $x \to \mu$ in $u_2 \bullet u_1$.*

Proof. The existence of $e : x \to \mu$ implies $x \in B$, hence x initial in u_1, which contradicts the very existence of e. □

Lemma 7.37. *Morphisms of plays preserve finality.*

Proof. If a player is final in the domain, then it is in the final position, hence has an image in the final position of the codomain, hence is final there. □

Lemma 7.38. *For any map $\alpha : u \to w$ in $\mathbb{D}_H$, for any player x in u and edge $e' : \mu' \to \alpha(x)$ from a core in w, there exists a core $\mu \in u$ and an edge $e : \mu \to x$ in u such that $G_\alpha(e) = e'$.*

Proof. Let first $X \to u \leftarrow Y$ and $X' \to w \leftarrow Y'$ be the considered morphisms.

Then, observe that x is not final in u, for otherwise it would be in X, hence $\alpha(x)$ would be in X' and final, contradicting the existence of e'.

So there exists $e : \mu \to x$ in u. But now, by target-linearity, $G_\alpha(\mu) = \mu'$, which entails the result. □

Lemma 7.39. *In any double cell* (7.1)*, both squares are pullbacks.*

Proof. X must consist precisely of all final players and channels of G_U, which must also be final in G_V, so finality in G_U implies finality in G_V. Conversely, any player or channel mapped to a final one in G_V has to be final. So X is a pullback of U and X'. The lower square being a pullback follows from similar reasoning. □

Proof of Lemma 7.35. Consider any α, and construct C, u_1, u_2, and the morphisms in Figure 5, as follows. First, let u_1 be the pullback $u \times_w w_1$, and then $C = u_1 \times_{w_1} Y$. Let then $u_2 = u \times_w w_2$, and the arrow $C \to u_2$ be induced by universal property of pullback. By the pullback lemma, $C = u_2 \times_{w_2} Y$. Because presheaf categories are adhesive [32], $\widehat{\mathbb{C}^f}$ is, and, $Y \to w_1$ being monic, we have a Van Kampen square. Thus, by the main axiom for adhesive categories, u is a pushout $u_1 +_C u_2$, i.e., $u \cong u_2 \bullet u_1$ in $\mathsf{Cospan}(\widehat{\mathbb{C}^f})$. Letting α_i be the arrow $u_i \to w_i$, for $i = 1, 2$, this yields the desired decomposition of α.

We still need to show that $A \to u_1 \leftarrow C$ and $C \to u_2 \leftarrow B$ are plays, and that the obtained decomposition is unique. Uniqueness follows from adhesivity of $\widehat{\mathbb{C}}$ and Lemma 7.39. Indeed, any decomposition looks like Figure 5, except that u_1, u_2, and C are not *a priori* obtained by pullback. But by Lemma 7.39, both back faces have to be pullbacks, hence so are the front faces by adhesivity.

Let us finally show that u_1 and u_2 are plays. It is easy to see that non-linearity or non-acyclicity of G_{u_1} (resp. G_{u_2}) would entail non-linearity or non-acyclicity of u or w_1 (resp. or w_2). Local 1-injectivity is also easy.

Let us now prove the missing conditions for $A \to u_1 \leftarrow C$.

a) Any player x of u_1 in the image of A is final, for otherwise its image in w_1 would be in the image of X and non-final.

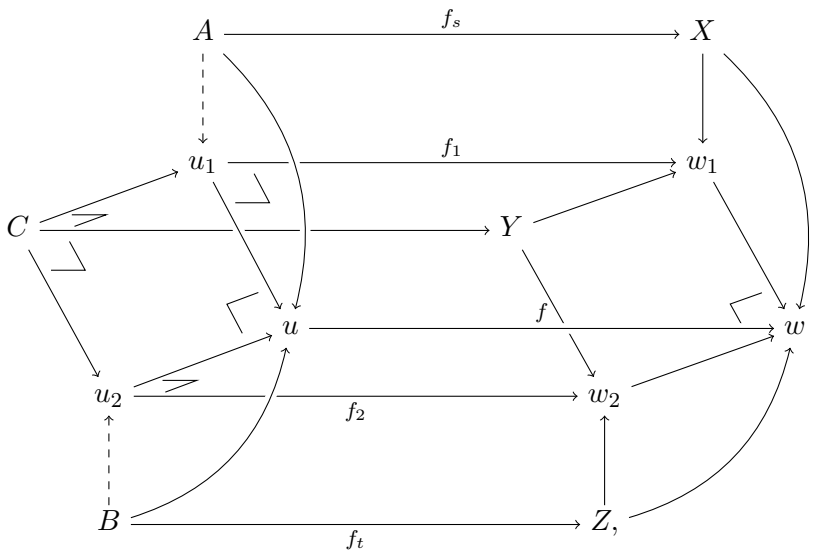


Figure 5: Proof of Lemma 7.35

b) Conversely, if a player $x \in u_1$ is final but not in A, then its image in u must be non-final by Theorem 7.17, because $u_1 \to u$ is monic. But then there is a core μ of u_2 with a path $\mu \to x$ in G_u, whose images in w yield a path from a core of w_2 to a player of w_1, contradicting Lemma 7.33. So A contains precisely the final players of u_1.

c) Now, if a channel $x \in u_1$ is not in A, then its image in u must be in A, hence $u_1 \to u$ cannot be mono, so neither can $w_1 \to w$, so neither can $Y \to w_2$, contradiction.

d) Finally, by construction, C contains precisely the initial players and channels of u_1.

Now, for $C \to u_2 \leftarrow B$.

a) By universal property of pullback, C contains all channels of u_2.

b) For players, clearly, for any player x in C, x is final in u_2. Indeed, otherwise, there would be a path $\mu \to x$ from a core μ in u_2, yielding a path $f_2(\mu) \to f_2(x)$ in w_2. But since x is in C, $f_2(x) \in Y$, which hence contains a non-final player, contradiction.

c) Conversely, if x is final in u_2, then $x' = f_2(x)$ is final in w_2. Indeed, otherwise, there would be an edge $\mu' \to x'$ from a core in w_2, so, by Lemma 7.38, an edge $\mu \to x$ in u with $f(\mu) = \mu'$. But then, $\mu \in u_2$, so x cannot be final. This shows that x' is final in w_2. But then $x' \in Y$, so, because $C = u_2 \times_w Y$, $x \in C$.

d) Consider now any player or channel x initial in u_2. First, x is also initial in u: otherwise, there would be an edge $x \to \mu$ to a core in u, with $\mu \in u_1$, hence an edge $f(x) \to f(\mu)$ in w from a channel of w_2 to a core of w_1, which is impossible by Lemma 7.36. So x is initial in u, hence $x \in B$.

e) Now, for any player or channel $x \in B$, x is initial in u, hence x is *a fortiori* initial in u_2. □

8. Conclusion and perspectives

8.1. **Conclusion.** We have described a denotational semantics of CCS based on presheaves, with a strong game-semantical flavour. Some aspects of the approach look promising to us.

First, our result is encouraging for potential applications of Kleene coalgebra to programming language theory, i.e., ascribing a semantics to the 'rule of the game' rather than attempting to organise operational semantics into some categorical structure.

Second, our use of techniques from categorical combinatorics (e.g., defining positions and plays as finite presheaves) provide a high-level, yet rigorous toolbox for dealing with string diagrams. (Compare, e.g., with available definitions of linear logic proof nets or interaction nets.)

Third, our notion of play encompassing both views and closed-world plays, and its rich notion of morphism yields a convincing interplay between strategies (presheaves on views) and behaviours (presheaves on plays). In particular,

- passing from one to the other is handled by standard categorical constructions,
- the general syntax and LTS for strategies provides a link to syntactic approaches.

Other aspects of our model are not as satisfactory.

First of all, the notion of playground is very complicated. In work in progress on a similar approach for π-calculus, we bypass the intermediate LTS $\mathcal{T}_{\mathbb{D}}$ of process terms, because it does not help so much — strategies are already really close to π-calculus terms. This seems to hint that the main result of playground theory is actually the characterisation of strategies by the syntax of Section 5.1. The good point is: this result does not at all need all axioms for playgrounds.

A second negative point is that some proofs may probably be improved. E.g., our proof that $\theta\colon CCS \to \mathcal{T}_{\mathbb{D}^{CCS}}$ is included in weak bisimilarity is a bit of a nightmare, with no apparent good reason. Similarly, we know already that our constructions for showing the fibration axiom (P1) may be improved. Indeed, the trick we use to restore synchronisations after restriction rests upon a *factorisation system* [16, 28]. In our current work on π, we use factorisation systems to prove the fibration axiom in a much more direct way (which was prompted by the fact that the method used here does not apply).

8.2. **Perspectives.** Beyond these rather technical concerns, we plan to adapt our semantics to more complicated calculi like π, the Join and Ambients calculi, calculi with passivation, functional calculi, possibly with extra features (e.g., references, data abstraction, encryption), with a view to eventually generalising it, perhaps to some SOS format. In particular, adapting the approach to functional calculi should clarify the relationship with Hyland-Ong innocence. In work in progress mentioned above, we construct a playground for π, whose proof of full abstraction remains to be completed. More speculative directions include

- designing a general way of constructing playgrounds automatically from more elementary data; work in progress reveals that this is a very subtle task;
- defining a notion of morphisms for playgrounds, which should induce translation functions between strategies, and find sufficient conditions for such morphisms to preserve, resp. reflect testing equivalences;
- generalising playgrounds to apply them beyond programming language semantics; in particular, preliminary work shows that playgrounds easily account for cellular

automata; this raises the question of how morphisms of playgrounds would compare with various notions of simulations between cellular automata [10];
- incorporate quantitative aspects from Kleene coalgebra into playground theory; this may start by refining fair testing equivalence to keep track of the probability of passing each test successfully.

References

[1] Samson Abramsky and Paul-André Melliès. Concurrent games and full completeness. In LICS 1999 [37], pages 431–442. doi: 10.1109/LICS.1999.782638.

[2] Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Information and Computation*, 163(2):409–470, 2000. doi: 10.1006/inco.2000.2930.

[3] J. Adámek, J.; Rosicky. *Locally Presentable and Accessible Categories*. Cambridge University Press, 1994. doi: 10.1017/CBO9780511600579.

[4] Gérard Berry and Gérard Boudol. The chemical abstract machine. In *POPL*, pages 81–94, 1990. doi: 10.1145/96709.96717.

[5] Filippo Bonchi, Marcello M. Bonsangue, Jan J. M. M. Rutten, and Alexandra Silva. Deriving syntax and axioms for quantitative regular behaviours. In *CONCUR*, volume 5710 of *LNCS*, pages 146–162. Springer Verlag, 2009. doi: 10.1007/978-3-642-04081-8_11.

[6] Marcello M. Bonsangue, Jan J. M. M. Rutten, and Alexandra Silva. A Kleene theorem for polynomial coalgebras. In *FoSSaCS*, volume 5504 of *LNCS*, pages 122–136. Springer Verlag, 2009. doi: 10.1007/978-3-642-00596-1_10.

[7] Diletta Cacciagrano, Flavio Corradini, and Catuscia Palamidessi. Explicit fairness in testing semantics. *Logical Methods in Computer Science*, 5(2), 2009. doi: 10.2168/LMCS-5(2:15)2009.

[8] Simon Castellan, Pierre Clairambault, and Glynn Winskel. Concurrent Hyland-Ong games. GaLoP, 2014.

[9] Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984. doi: 10.1016/0304-3975(84)90113-0.

[10] Marianne Delorme, Jacques Mazoyer, Nicolas Ollinger, and Guillaume Theyssier. Bulking I: An abstract theory of bulking. *Theoretical Computer Science*, 412(30):3866–3880, 2011. doi: 10.1016/j.tcs.2011.02.023.

[11] Charles Ehresmann. Catégories structurées. *Annales scientifiques de l'Ecole Normale Supérieure*, 80(4):349–426, 1963.

[12] Charles Ehresmann. *Catégories et structures*. Dunod, 1965.

[13] Claudia Faggian and Mauro Piccolo. Partial orders, event structures and linear strategies. In *TLCA*, volume 5608 of *LNCS*, pages 95–111. Springer Verlag, 2009. doi: 10.1007/978-3-642-02273-9_9.

[14] Marcelo P. Fiore. Fibred models of processes: Discrete, continuous, and hybrid systems. In *IFIP TCS*, volume 1872 of *LNCS*, pages 457–473. Springer Verlag, 2000. doi: 10.1007/3-540-44929-9_32.

[15] FoSSaCS 2004. *FoSSaCS*, volume 2987 of *LNCS*, 2004. Springer Verlag.

[16] Peter Freyd and G. M. Kelly. Categories of continuous functors, I. *Journal of Pure and Applied Algebra*, 2:169–191, 1972. doi: 10.1016/0022-4049(72)90001-1.

[17] Richard H. G. Garner. *Polycategories*. PhD thesis, University of Cambridge, 2006.

[18] Dan R. Ghica and Andrzej S. Murawski. Angelic semantics of fine-grained concurrency. In FoSSaCS 2004 [15], pages 211–225. doi: 10.1007/978-3-540-24727-2_16.
[19] Daniele Gorla. Towards a unified approach to encodability and separation results for process calculi. *Information and Computation*, 208(9):1031–1053, 2010. doi: 10.1016/j.ic.2010.05.002.
[20] Marco Grandis and Robert Paré. Limits in double categories. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, 40(3):162–220, 1999.
[21] Marco Grandis and Robert Paré. Adjoints for double categories. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, 45(3):193–240, 2004.
[22] Russell Harmer and Guy McCusker. A fully abstract game semantics for finite nondeterminism. In LICS 1999 [37], pages 422–430. doi: 10.1109/LICS.1999.782637.
[23] Russell Harmer, Martin Hyland, and Paul-André Melliès. Categorical combinatorics for innocent strategies. In *LICS*, pages 379–388. IEEE Computer Society, 2007. doi: 10.1109/LICS.2007.14.
[24] Tom Hirschowitz and Damien Pous. Innocent strategies as presheaves and interactive equivalences for CCS. In *ICE*, pages 2–24, 2011. doi: 10.4204/EPTCS.59.2.
[25] Tom Hirschowitz and Damien Pous. Innocent strategies as presheaves and interactive equivalences for CCS. *Scientific Annals of Computer Science*, 22(1):147–199, 2012. doi: 10.7561/SACS.2012.1.147. Selected papers from ICE '11.
[26] J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000. doi: 10.1006/inco.2000.2917.
[27] Bart Jacobs. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam, 1999.
[28] André Joyal. Factorisation systems. `http://ncatlab.org/joyalscatlab`.
[29] André Joyal and Myles Tierney. Notes on simplicial homotopy theory. Course at the CRM, February 2008.
[30] André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation and open maps. In *LICS*, pages 418–427. IEEE Computer Society, 1993. doi: 10.1109/LICS.1993.287566.
[31] Joachim Kock. Polynomial functors and trees. *International Mathematics Research Notices*, 2011(3):609–673, 2011. doi: 10.1093/imrn/rnq068.
[32] Stephen Lack and Pawel Sobocinski. Adhesive categories. In FoSSaCS 2004 [15], pages 273–288. doi: 10.1007/978-3-540-24727-2_20.
[33] James Laird. Game semantics for higher-order concurrency. In *FSTTCS*, volume 4337 of *LNCS*, pages 417–428. Springer Verlag, 2006. doi: 10.1007/11944836_38.
[34] F. William Lawvere and Stephen H. Schanuel. *Conceptual mathematics - a first introduction to categories*. Cambridge University Press, 1997.
[35] Tom Leinster. *Higher Operads, Higher Categories*, volume 298 of *London Mathematical Society Lecture Notes*. Cambridge University Press, Cambridge, 2004.
[36] Tom Leinster. *Basic Category Theory*, volume 143 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 2014.
[37] LICS 1999. *14th Symposium on Logic in Computer Science*, 1999. IEEE Computer Society.
[38] Saunders Mac Lane. *Categories for the Working Mathematician*. Number 5 in Graduate Texts in Mathematics. Springer Verlag, 2nd edition, 1998.
[39] Saunders MacLane and Ieke Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Universitext. Springer, 1992.

[40] Paul-André Melliès. Asynchronous games 2: the true concurrency of innocence. In *Proc. CONCUR '04*, volume 3170 of *LNCS*, pages 448–465. Springer Verlag, 2004. doi: 10.1007/978-3-540-28644-8_29.

[41] Paul-André Melliès. Game semantics in string diagrams. In *LICS*, pages 481–490. IEEE, 2012. doi: .1109/LICS.2012.58.

[42] Paul-André Melliès and Samuel Mimram. Asynchronous games: Innocence without alternation. In *CONCUR*, volume 4703 of *LNCS*, pages 395–411. Springer Verlag, 2007. doi: 10.1007/978-3-540-74407-8_27.

[43] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980. doi: 10.1007/3-540-10235-3.

[44] Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[45] Hanno Nickau. Hereditarily sequential functionals. In *LFCS*, volume 813 of *LNCS*, pages 253–264. Springer Verlag, 1994. doi: 10.1007/3-540-58140-5_25.

[46] Robert Paré. Yoneda theory for double categories. *Theory and Applications of Categories*, 25(17):436–489, 2011.

[47] Julian Rathke and Pawel Sobocinski. Deconstructing behavioural theories of mobility. In *IFIP TCS*, volume 273 of *IFIP*, pages 507–520. Springer Verlag, 2008. doi: 10.1007/978-0-387-09680-3_34.

[48] Arend Rensink and Walter Vogler. Fair testing. *Information and Computation*, 205 (2):125–198, 2007. doi: 10.1016/j.ic.2006.06.002.

[49] Silvain Rideau and Glynn Winskel. Concurrent strategies. In *LICS*, pages 409–418. IEEE Computer Society, 2011. doi: 10.1109/LICS.2011.13.

[50] Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2012.

[51] Davide Sangiorgi and Jan Rutten, editors. *Advanced Topics in Bisimulation and Coinduction*. Number 52 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2011.

[52] Davide Sangiorgi and David Walker. *The π-calculus - a theory of mobile processes*. Cambridge University Press, 2001.

[53] Glynn Winskel. Strategies as profunctors. In *FoSSaCS*, volume 7794 of *LNCS*, pages 418–433. Springer Verlag, 2013. doi: 10.1007/978-3-642-37075-5_27.

[54] Glynn Winskel and Mogens Nielsen. *Handbook of Logic in Computer Science*, volume 4 of *Oxford science publications*, chapter Models for concurrency. Clarendon, 1995.

$\mathbb{C}$	base category, over which positions and plays are presheaves
$\star$, $[n]$, π_n^l, π_n^r, π_n, ν_n, $\heartsuit_n$, $\iota_{n,i}$, $o_{m,j}$, $\tau_{n,i,m,j}$, $[m]_{a_1,\dots\vdash c_1,\dots[n]}$	objects of $\mathbb{C}$
$\mathsf{Cospan}(-)$	bicategory of cospans of $-$
$\mathbb{D}_h$	category of positions and embeddings
$\mathbb{D}_v$	bicategory of positions and plays
$\mathbb{D}$	playground
$\mathbb{D}^{CCS}$	playground for CCS
$\mathbb{E}$	category of plays and extensions
B_X	category of behaviours on X
$\mathbb{E}^{\mathbb{V}}$	category of views and extensions
S_X	category of strategies on X
$\mathrm{Pl}(X)$	players of position X
$v^{x,u}$	view of $x\colon d \to X$ in $u\colon X \nrightarrow Y$
$x^u\colon d^{x,u}\colon Y$	initial player of x in u
$u_{\mid k}\colon D_{k,u} \nrightarrow Y$	restriction of $u\colon X' \nrightarrow X$ along $k\colon Y \to X$
$\mathrm{Pl}_M(X)$	players of position X whose view in $M\colon X \nrightarrow Y$ is non-trivial
S_x	projection of $S \in \mathsf{S}_X$ to $x \in \mathrm{Pl}(X)$
$[S,T]$	copairing of S and T
$S \cdot v$	residual of S after v
$S_{\mid\sigma}$	restriction of S to antecedents of σ
$\mathbb{Q}$	graph of full quasi-moves
$[\mathbb{B}]_d$	set of isomorphism classes of basic moves over d
$[\mathbb{F}]_X$	set of isomorphism classes of full moves over X
$\chi[M]$	set of basic b's s.t. $\exists\ b \to M$
$[\mathbb{F}^1]_X \subseteq [\mathbb{F}]_X$	subset of full moves M such that $\chi[M]$ is a singleton
$[\mathbb{F}^+]_X \subseteq [\mathbb{F}]_X$	subset of full moves M such that $\chi[M]$ is not a singleton
r^u, i^u	bijection, for all plays $u\colon X' \nrightarrow X$, $\sum_{(d,x)\in\mathrm{Pl}(X)} \mathrm{Pl}(D_{x,u}) \to \mathrm{Pl}(X')$
$d \vdash S$	strategy term
$d \vdash_{\mathfrak{D}} D$	definite strategy term
$d \vdash T$	process term
$(I,h,S)^\perp$	set of tests passed by (I,h,S)
$\sim_f^G$	fair testing eq. in graph w.c. G
$\sim_{f,s}$	standard fair testing eq. in CCS
$\sim_f$	semantic fair testing eq.
$\perp^G$	pole for fair testing eq. in G
$\perp\!\!\!\perp$	pole for semantic fair test. eq.
$\perp$	pole for CCS (Def. 2.22)
$\perp\!\!\!\!\perp$	intermediate pole (Lem. 6.22)
CCS	LTS for CCS
$\mathcal{S}$	LTS for strategies
T	set of process terms
$\mathcal{T}$	LTS for T: $\mathrm{ob}(\mathcal{T})=\mathsf{T}$
$(\!\vert - \vert\!)$	translation $CCS \to \mathcal{S}$
θ	translation $CCS \to \mathcal{T}$
$[\![-]\!]$	translation $\mathcal{T} \to \mathcal{S}$
$\mathbb{W}_{CCS}$	set of closed-world quasi-moves
$\mathbb{D}^{\mathbb{W}} \subseteq \mathbb{D}_v$	subbicat. of closed-world plays
$\ell_{\mathbb{D}}$	labelling of closed-world plays in $\{id, \heartsuit\}$: $\mathbb{D}^{\mathbb{W}} \to \mathsf{fc}(\Sigma)$
$A^{\mathbb{W}}$	'closed-world' subgraph of a graph with complementarity A
$\triangleright^A$	compatibility relation for A: $A^2 \nrightarrow A^{\mathbb{W}}$
$e \Downarrow e'$	notation for the composite $A^{\frown} \hookrightarrow A^2 \nrightarrow A^{\mathbb{W}} \to \Sigma$
$[x,y]$	choice of 'amalgamation' in G
$\chi\colon \mathbb{IQ} \to \mathbb{Q}$	subgraph of edges with double cell $id_I^\bullet \to M$
$\xi\colon \mathbb{IQ} \to \mathbb{A}$	mapping to CCS labels
G modular	$\triangleright^G$ strong bisim over Σ
$x^{\frown}$	$\{y \mid x \frown y\}$
$x \bowtie y$	$x^{\frown} = y^{\frown}$
$G \diamond_A H$	blind composition of G and H over A
adequacy of $G \to A$	(essentially) $\perp^{G^{\frown}} = \perp^{G \diamond_A G}$
$\mathcal{H}^A$	A-trees
$\mathcal{F}_a$	failures over $a \in A$
$\mathbf{fl}$	failures to A-trees: $\mathcal{F} \to \mathcal{H}^A$
nice alphabet	enough ticks, finitely branching, inertly silent (Def. 6.43)
core	move element of some presheaf, of maximal dimension
U locally 1-inj.	cores map inj. to U, except perhaps for channels in the interface
G_U	causal graph of U
$\mathrm{El}(-)$	elements $\searrow$ synchronisations
$\multimap$	map between $\mathrm{El}(-)$'s
horn $\tau^-_{n,i,m,j}$	synchro. minus $id_{\tau_{n,i,m,j}}$
Δ_f	change of base along f

Figure 6: Cheat sheet

Decidability of Identity-free Relational Kleene Lattices

Paul Brunet, Damien Pous

Decidability of Identity-free Relational Kleene Lattices

Paul Brunet & Damien Pous

Plume team – LIP, CNRS, ENS de Lyon, Inria, UCBL, Université de Lyon, UMR 5668

Abstract

Families of binary relations are important interpretations of regular expressions, and the equivalence of two regular expressions with respect to their relational interpretations is decidable: the problem reduces to the equality of the denoted regular languages.

Putting together a few results from the literature, we first make explicit a generalisation of this reduction, for regular expressions extended with converse and intersection: instead of considering sets of words (i.e., formal languages), one has to consider sets of directed and labelled graphs.

We then focus on identity-free regular expressions with intersection—a setting where the above graphs are acyclic—and we show that the corresponding equational theory is decidable. We achieve this by defining an automaton model, based on Petri Nets, to recognise these sets of acyclic graphs, and by providing an algorithm to compare such automata.

Introduction

Binary relations appear everywhere in mathematics and computer science, together with the operations of union ($\cup$), intersection ($\cap$), composition ($\circ$), converse ($.^\vee$), reflexive-transitive closure ($.^\star$), and the constants identity (Id) and empty relation ($\varnothing$). As such, an algorithm for deciding the equivalence of expressions built with these operators with respect to their relational interpretations is a very desirable goal. However such an algorithm has yet to be found.

Regular expressions [6], where only the operators $\cup, \circ, .^\star, \mathrm{Id}$, and $\varnothing$ are allowed, are the most famous example of a decidable fragment [9]. In this setting, it is now well-known that the equivalence of two expressions in all relational interpretations is equivalent to the equality of the regular languages denoted by these expressions in the usual sense (the letter x is interpreted as $\{x\}$). Several equational or semi-equational theories are known to be complete for this fragment [10–12].

The converse operation can also be added to regular expressions, and the resulting theory remains decidable (see [3, 7] or [4]). In this case, decidability is obtained by 1) reducing the problem of equivalence of two expressions to the equality of some regular sets of words over an extended alphabet, and 2) defining automata constructions to recognise these sets.

Freyd and Scedrov sketched an algorithm for representable allegories [8, page 208], that is, expressions with composition, intersection, converse, and identity, but without union or reflexive-transitive closure. Similar constructions were given independently by Andréka and Bredikhin [2], in a more comprehensive way. The key idea is the following: if we restrict ourselves to the above syntax (variables, composition, intersection, converse, identity), we get what is called *ground terms*. Such a term u can be represented as labelled directed graphs $G(u)$ with two distinguished vertices called the *input* and the *output*. A variable a corresponds to a graph with one edge labelled by a linking the input to the output. The identity is represented by the graph with a single vertex and no edges, The composition of two graphs with disjoints sets of vertices can be performed by identifying the output of the first graph and the input of the second one. The operation corresponding to the intersection

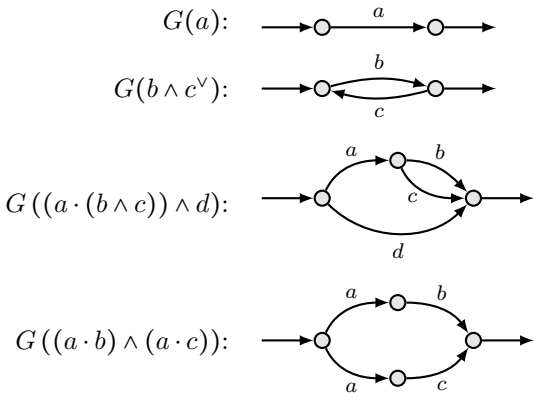


Figure 1: Graphs associated to some ground terms

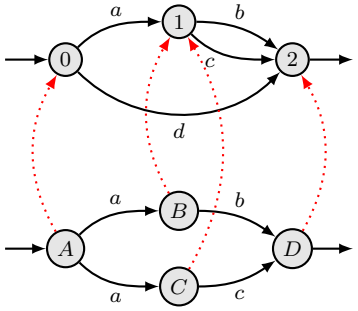


Figure 2: A graph homomorphism.

consists in merging the inputs of the two graphs, as well as their outputs. And finally, converse is obtained by swapping the inputs and the outputs. Some examples are given in Figure 1.

These graphs can be endowed with a preorder relation $G \blacktriangleleft F$, defined by the existence of a graph homomorphism from F to G (preserving inputs and outputs). For instance the graph corresponding to $(a \cdot (b \wedge c)) \wedge d$ is smaller than the graph of $(a \cdot b) \wedge (a \cdot c)$, thanks to the homomorphism depicted in Figure 2 using dotted arrows. Notice that this preorder has nothing to do with the respective sizes of the graphs: a graph may very well be smaller (in the sense of $\blacktriangleleft$) than another while having more vertices (and vice versa). The key result from Freyd and Scedrov [8, page 208], or Andréka and Bredikhin [2, Theorem 1], is that for any two ground terms u, v, u is contained in v under any relational interpretation if and only if $G(u) \blacktriangleleft G(v)$.

This is for ground terms; to handle the whole syntax, we need to add union and reflexive-transitive closure. It suffices for that to consider sets of graphs: to each expression e, one can associate a set of graphs $G(e)$. Writing $X^{\blacktriangleleft}$ for the downward closure of a set of graphs X by the relation $\blacktriangleleft$, we obtain the following generalisation of the above result: for any two expressions e and f, e is contained in f under any relational interpretation if and only if $G(e) \subseteq G(f)^{\blacktriangleleft}$. (Theorem 6 in the sequel—this result is almost there in the work by Andréka et al. [1], but this explicit formulation is new, to the best of our knowledge.)

This result encompasses the case of plain regular expressions, whose graphs are just words and for which the preorder $\blacktriangleleft$ reduces to isomorphism, but also the case of regular expressions with converse, whose graphs are words over a duplicated alphabet and for which the preorder $\blacktriangleleft$ can be reformulated in terms of the rewriting system proposed by Ésik et al. [3, 7].

Our main contribution is then to exploit this characterisation to obtain decidability for identity-free regular expressions with intersection, whose equational theory has been studied by Andréka et al. [1]. The reason why we need to exclude identity and converse is that in presence of intersection, they yield cyclic graphs, and we do not know how to handle such graphs. We hope to get rid of this assumption in future work.

The key concept which we introduce is a new kind of finite automaton, allowing us to recognise sets of graphs that are downward-closed w.r.t. the graph embedding relation $\blacktriangleleft$. To give some intuition about this automaton model, let us look at the example from Figure 2, and try to build sequentially a morphism h from $F = G((a \cdot b) \wedge (a \cdot c))$ to $G = G((a \cdot (b \wedge c)) \wedge d)$.

- We start by placing a token ⓐ on A. We know that for h to be a morphism, it has to preserve the input of the graph, so we map A to position 0 in G.

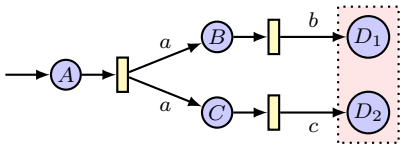


Figure 3: The automaton corresponding to the term $(a \cdot b) \wedge (a \cdot c)$.

- There are two outgoing edges from A, both labelled by a. We split token ⓐ into Ⓑ and Ⓒ, and move Ⓑ to position B and Ⓒ to position C. We then map the positions of both tokens to position 1 in G, which is consistent with h being a morphism, thanks to the arc $(0, a, 1)$.
- Now we try to move Ⓑ. B has one outgoing edge, labelled by b. We may move Ⓑ to D, and using the arc $(1, b, 2)$ in G we map D to position 2.
- Then we can look at Ⓒ. We have to move it to position D, and thanks to the arc $(1, c, 2)$ we can confirm the map of D to 2, and merge back Ⓑ and Ⓒ.

At the end, we have only one token, placed on the output of F, and during the procedure we have mapped all positions in F to positions in G, while preserving all labelled edges.

This kind of procedure is reminiscent of Petri nets [13–15]: at each step we relate tokens to positions in G, and fire transitions according to the edges of G. This is the basic idea behind the notion of Petri automata which we introduce in Section 2. For instance the Petri automaton we will construct for the term $(a \cdot b) \wedge (a \cdot c)$ is depicted in Figure 3, and the procedure sketched above can then be formally described as a reading of the graph G in this automaton.

Given an expression e, we show in Section 3 how to build a Petri automaton that recognises exactly the graphs in $G(e)^{\blacktriangleleft}$. We then show in Section 4 how to compare Petri automata. Several difficulties arise, that do not appear with classical word automata. Our solution nevertheless uses a standard coinductive approach, where we define an appropriate notion of simulation.

1. Expressions and languages

In this section we consider the full signature $\langle \wedge, \vee, \cdot, ^\star, ^\vee, \mathbb{0}, \mathbb{1} \rangle$ of Kleene lattices with conversion. We fix a set X of variables, and we denote by $\mathrm{Reg}_X^{\vee\wedge}$ the set of expressions build from variables in X with these connectives. These expressions are meant to be interpreted in relational models: $\cdot$ corresponds to the composition of relations; $\vee$ to the union; $\wedge$ to the intersection; R^* to the reflexive transitive closure of a relation R; and $R^\vee$ to the converse of R. The constants $\mathbb{0}$ and $\mathbb{1}$ are respectively interpreted as the empty relation and the identity relation. For any set S, we write $\mathcal{P}(S) \coloneqq \{P \mid P \subseteq S\}$ for the set of subsets of S. Let $A \to B$ be the functions from A to B and $A \dashrightarrow B$ the partial maps from A to B. $\mathrm{dom}(f)$ denotes the domain of a partial map f. If $\sigma : X \to \mathcal{P}(S \times S)$ is an interpretation of the alphabet X into some space of relations, we write $\widehat{\sigma}$ for the unique homomorphism extending σ from $\mathrm{Reg}_X^{\vee\wedge}$ to $\mathcal{P}(S \times S)$. We say that two expressions e and f are relationally equivalent, written $Rel \vDash e = f$, if for any relational interpretation σ we have $\widehat{\sigma}(e) = \widehat{\sigma}(f)$. Similarly, we write $Rel \vDash e \leqslant f$ if $\widehat{\sigma}(e) \subseteq \widehat{\sigma}(f)$ holds for any σ.

The *ground terms* are defined by the following sub-syntax:

$$u, v, w \in W_X \coloneqq x \in X \mid w \cdot w \mid w \wedge w \mid w^\vee \mid \mathbb{1} \ .$$

We let G range over 2-pointed labelled directed graphs, which we simply call *graphs* in the sequel. Those are tuples $\langle V, E, \iota, o \rangle$ with V a finite set of vertices, $E \subseteq V \times X \times V$ a set of edges labelled with X, and $\iota, o \in V$ the two distinguished vertices, respectively called *input* and *output*.

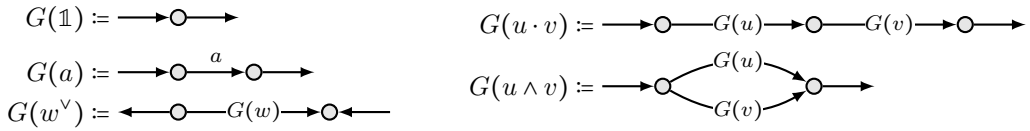


Figure 4: Graphs corresponding to ground terms.

To each ground term w, we associate such a graph $G(w)$. The graph for $\mathbb{1}$ has only one vertex, both input and output. The graph of a has one edge labelled by a linking its input to its output. The composition of two graphs with disjoint sets of vertices can be performed by identifying the output of the first graph and the input of the second one. The intersection on graphs consists in merging their inputs and merging their outputs. The converse consists simply in exchanging the input and the output of a graph. See Figure 4 for a graphical description of this construction. Those graphs were introduced independently by Freyd and Scedrov [8, page 208], and Andréka and Bredikhin [2].

Another useful notion is the notion of morphism between graphs:

Definition 1 (Graph morphism, preorder on ground terms)
A *graph morphism* from $\langle V_1, E_1, \iota_1, o_1\rangle$ to $\langle V_2, E_2, \iota_2, o_2\rangle$ is a map $h : V_1 \to V_2$ such that $h(\iota_1) = \iota_2$, $h(o_1) = o_2$, and $(p, x, q) \in E_1$ entails $(h(p), x, h(q)) \in E_2$. We denote by $\blacktriangleleft$ the relation on graphs defined by $G \blacktriangleleft G'$ if there exists a graph morphism from G' to G. This relation gives rise to a preorder on ground terms, written $\lhd$ and defined by $u \lhd v$ if $G(u) \blacktriangleleft G(v)$. *

Given a set S of graphs, we write $S^{\blacktriangleleft}$ for its downward closure w.r.t. $\blacktriangleleft$: $S^{\blacktriangleleft} \coloneqq \{G \mid G \blacktriangleleft G', G' \in S\}$. Similarly, we write $S^{\lhd}$ for the downward closure of a set of ground terms w.r.t. $\lhd$.

As explained in the introduction, the above preorder on ground terms precisely characterises inclusion under arbitrary relational interpretations:

Theorem 2 ([2, Theorem 1], or [8, page 208]). *For all ground terms $u, v \in W_X$, we have*

$$\mathit{Rel} \vDash u \leqslant v \Leftrightarrow u \lhd v \ .$$

To extend this result to the expressions we consider in this paper, we introduce the following generalisation of the language of a regular expression. Sets of words become sets of ground terms.

Definition 3 (Term language of an expression)
The *term language* denoted by an expression $e \in \mathrm{Reg}_X^{\vee\wedge}$, written $[\![e]\!]$, is the set of ground terms defined inductively as follows:

$$\begin{aligned}
[\![x]\!] &\coloneqq \{x\} & [\![e \cdot f]\!] &\coloneqq \{w \cdot w' \mid w \in [\![e]\!] \text{ and } w' \in [\![f]\!]\} \\
[\![e \vee f]\!] &\coloneqq [\![e]\!] \cup [\![f]\!] & [\![e \wedge f]\!] &\coloneqq \{w \wedge w' \mid w \in [\![e]\!] \text{ and } w' \in [\![f]\!]\} \\
[\![e^\star]\!] &\coloneqq \textstyle\bigcup_{n\in\mathbb{N}} \{w_1 \cdot \cdots \cdot w_n \mid \forall i, w_i \in [\![e]\!]\} & [\![e^\vee]\!] &\coloneqq \{w^\vee \mid w \in [\![e]\!]\} \\
[\![\mathbb{1}]\!] &\coloneqq \{\mathbb{1}\} & [\![\mathbb{0}]\!] &\coloneqq \varnothing \ .
\end{aligned}$$

*

We need a slight refinement of a lemma established by Andréka, Mikulás, and Németi [1]:

Lemma 4. *For all expression $e \in \mathit{Reg}_X^{\vee\wedge} X$, and all relational interpretations $\sigma : X \to \mathcal{P}(S \times S)$, we have*

$$\widehat{\sigma}(e) = \bigcup_{w\in[\![e]\!]} \widehat{\sigma}(w) = \bigcup_{w\in[\![e]\!]^{\lhd}} \widehat{\sigma}(w) \ .$$

Proof. The first equality is exactly [1, Lemma 2.1]; for the second one, we use the fact that $\widehat{\sigma}(w) \subseteq \widehat{\sigma}(u)$ whenever $w \lhd u$, thanks to Theorem 2 (i.e., [2, Theorem 1]). □

The above definitions make it possible to characterise inclusion under all relational interpretation in terms of downward-closed term languages. To obtain decidability, we need to go one step further, by considering graph languages.

Definition 5 (Graph language of an expression)
The *graph language* of an expression e, denoted by $G(e)$ is the set of graphs associated to the ground terms in $[\![e]\!]$: $G(e) \coloneqq \{G(w) \mid w \in [\![e]\!]\}$. *

We finally obtain the following characterisation, which allows us to reduce validity in *Rel* to an equality of graph languages.

Theorem 6. *The following properties are equivalent, for all expressions $e, f \in \mathrm{Reg}_X^{\vee\wedge}$:*

(i) $\mathit{Rel} \vDash e = f$,

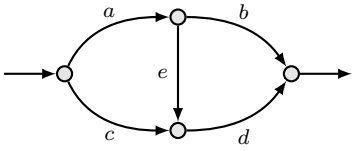
(ii) $[\![e]\!]^{\lhd} = [\![f]\!]^{\lhd}$,

(iii) $G(e)^{\blacktriangleleft} = G(f)^{\blacktriangleleft}$.

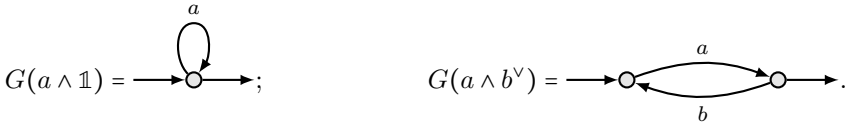
Proof. The implication (ii) ⇒ (i) follows easily from Lemma 4, and (iii) ⇒ (ii) is a matter of unfolding definitions. For (i) ⇒ (iii), we mainly use [2, Lemma 3]. □

The above statement can also be reformulated in terms of inclusions, to match the result announced in the introduction: $\mathit{Rel} \vDash e \leqslant v$ if and only if $[\![e]\!] \subseteq [\![f]\!]^{\lhd}$ if and only if $G(e) \subseteq G(f)^{\blacktriangleleft}$.

Also notice that while by definition $G(e)$ only contains graphs emanating from ground terms, this is not the case for its closure $G(e)^{\blacktriangleleft}$. For instance, $G((a \cdot b) \wedge (c \cdot d))^{\blacktriangleleft}$ contains the following graph, which is not the graph of any ground term.



The above result holds for the whole syntax of regular expressions with converse and intersection. However, in the remainder of the paper, we have to focus on expressions without converse and identity. This is because in combination with intersection, these two operations introduce cycles in the graphs associated to ground terms. Consider for instance the graphs for $a \wedge \mathbb{1}$ and $a \wedge b^{\vee}$:



Since reflexive-transitive closure $(\cdot^{\star})$ implicitly contains an occurrence of the identity, we also have to replace this operator with transitive closure $(\cdot^{+})$. We thus work we expressions from $\mathrm{Reg}_X^{\wedge -}$, defined with the following syntax: $e, f \in \mathrm{Reg}_X^{\wedge -} \coloneqq x \in X \mid e \wedge f \mid e \vee f \mid e \cdot f \mid e^{+} \mid \mathbb{0}$. Accordingly, ground terms are restricted to the following syntax: $u, v, w \in W_X^{-} \coloneqq x \in X \mid w \cdot w \mid w \wedge w$.

2. Petri Automata

Before getting to our definition of automata, we recall the standard notion of Petri net.

Definition 7 (Petri Net)
A *Petri Net* is a structure $N = \langle P, T, F, W, M_0 \rangle$ where:

- P and T are finite disjoints sets, respectively of *places* and *transitions*;
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs, called the *flow relation*;
- $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ is a weight function, such that $W(f) = 0$ if $f \notin F$;
- $M_0 : P \to \mathbb{N}$ is the initial marking.

Given a marking M in a net N, a transition $\tau \in T$ is *enabled* if for any place p such that (p, τ) is in the flow relation F, we have $W(p, \tau) \leqslant M(p)$. In that case, τ can *fire*, and it results in a new marking M' such that $M'(p) = M(p) - W(p, \tau) + W(\tau, p)$. A marking is called *accessible* if it can be obtained by successively firing transitions starting from M_0. *

To present examples in a simple way, we use the standard graphical representation of Petri nets: they are represented as graphs, with round nodes for places and rectangular nodes for transitions. The flow relation is simply represented by arrows; the places appearing in the initial marking have an additional incoming arrow.

A Petri net is said to be *one-bounded* if in any accessible marking M, no place is marked with more than one token. Such markings can be seen as finite sets of places; we call them *configurations* in the sequel, and we let ξ, Ξ range over them. Bounded nets form an interesting class of nets, because many problems which are undecidable in the general case become decidable in this setting: whereas general Petri nets have an infinite set of accessible markings, they are finitely many in a bounded net.

Our notion of *Petri Automata* is defined below. The main difference with regular Petri nets is the labelling with letters from the alphabet X of all arcs coming out of transitions. Note that this slightly differs from the usual notion of *labelled Petri net*, where the labels are put on transitions.

Definition 8 (Petri Automaton)
A *Petri automaton* is a structure $\langle N, L, \mathscr{F} \rangle$ where:

- $N = \langle P, T, F, W, M_0 \rangle$ is a *one-bounded* Petri net such that:
  - the weight $W(f)$ of all arcs f that appear in F is equal to 1;
  - all transitions $\tau \in T$, have at least one incoming arc and one outgoing arc, meaning that there are places $p, q \in P$ such that $(p, \tau) \in F$ and $(\tau, q) \in F$;
  - there is an initial place ι such that M_0 contains only one token, placed in ι.
- $L : (T \times P) \cap F \to X$ is a labelling function;
- $\mathscr{F} \subseteq \mathcal{P}(P)$ is a set of final configurations. *

A transition τ in a Petri automaton can be alternatively described by a pair $[\tau] = (s, t)$ where:

- $s = \{ p \mid (p, \tau) \in F \}$ is the *input of* τ, often denoted by ${}^\bullet\tau$, and
- $t = \{ (x, q) \mid (\tau, q) \in F \text{ and } x = L(\tau, q) \}$ is the *output of* τ. Notice that this differs from the usual notion of output of a transition in a Petri net: in the usual setting $\tau^\bullet$ is just the set of places reachable from the τ. Here we add to each place the label of the arc reaching it.

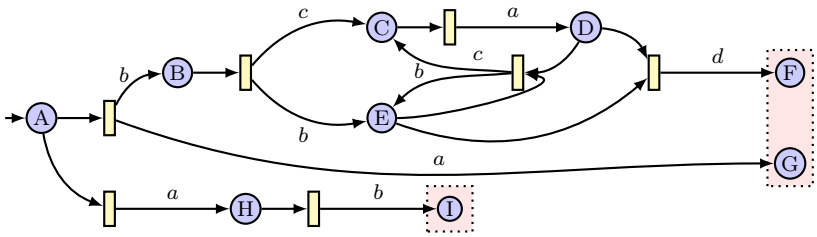


Figure 5: A Petri automaton. The initial state is A, and the final configurations are $\{I\}$ and $\{F, G\}$.

For commodity reasons, we will thus define Petri automata using quadruples $\mathscr{A} = \langle P, \mathscr{T}, \iota, \mathscr{F} \rangle$ with ι the initial place and $\mathscr{T} = \{(s,t) \mid \exists \tau \in T : [\tau] = (s,t)\}$.

Graphical representations of such automata are given in Figures 3 and 5. In these drawings a final configuration is represented by a dotted rectangle around the places contained in this configuration. Now we explain how to use Petri automata to define languages of graphs. We first describe what is a *run* of an automaton, and then how to use runs to read graphs.

Let $\mathscr{A} = \langle P, \mathscr{T}, \iota, \mathscr{F} \rangle$ be a Petri automaton. We write $\xi \xrightarrow{\tau}_{\mathscr{A}} \xi'$ when the configuration ξ' can be obtained by firing some transition τ in the configuration ξ. A set of transitions $T \subseteq \mathscr{T}$ is called *compatible* if their inputs are pairwise disjoint. If furthermore all transitions in T are enabled in a configuration ξ, one can observe that the configuration ξ' reached after firing them successively does not depend on the order in which they are fired. In that case we write $\xi \xrightarrow{T}_{\mathscr{A}} \xi'$.

Definition 9 (Run, accepting run, parallel run)
A *run* is a sequence $\xi = \langle (\xi_k)_{0 \leqslant k \leqslant n}, (\tau_k)_{0 \leqslant k < n} \rangle$ of configurations and transitions, such that $\xi_k \subseteq P$, $\tau_k \in \mathscr{T}$ and $\forall k < n,\ \xi_k \xrightarrow{\tau_k} \xi_{k+1}$. When $\xi_0 = \{\iota\}$ and $\xi_n \in \mathscr{F}$, we call ξ an *accepting run*.

A *parallel run* is defined similarly, as a sequence $\Xi = \langle (\Xi_k)_{0 \leqslant k \leqslant n}, (T_k)_{0 \leqslant k < n} \rangle$, where the $T_k \subseteq \mathscr{T}$ are compatible sets of transitions such that $\Xi_k \xrightarrow{T_k} \Xi_{k+1}$. ∗

(Note that a run ξ is uniquely determined by ξ_0 and the sequence (τ_k): all subsequent configurations can be computed deterministically.)

Example 10 (An accepting run in the automaton from Figure 5)
Consider the run $\xi = \langle (\xi_0, \xi_1, \xi_2, \xi_3, \xi_4, \xi_5, \xi_6), (\tau_0, \tau_1, \tau_2, \tau_3, \tau_4, \tau_5) \rangle$, with

$$\begin{array}{rl|rl} \xi_0 = & \{A\}, & \tau_0 = & (\{A\}, \{(b,B),(a,G)\}), \\ \xi_1 = & \{B,G\}, & \tau_1 = & (\{B\}, \{(c,C),(b,E)\}), \\ \xi_2 = \xi_4 = & \{C,E,G\}, & \tau_2 = \tau_4 = & (\{C\}, \{(a,D)\}), \\ \xi_3 = \xi_5 = & \{D,E,G\}, & \tau_3 = & (\{D,E\}, \{(c,C),(b,E)\}), \\ \xi_6 = & \{F,G\}. & \tau_5 = & (\{D,E\}, \{(d,F)\}). \end{array}$$

We can easily check, using the firing rules of a Petri net that:

$$\{A\} \xrightarrow{\tau_0} \{B,G\} \xrightarrow{\tau_1} \{C,E,G\} \xrightarrow{\tau_2} \{D,E,G\} \xrightarrow{\tau_3} \{C,E,G\} \xrightarrow{\tau_4} \{D,E,G\} \xrightarrow{\tau_5} \{F,G\}.$$

As $\{A\}$ is the initial configuration and $\{F,G\} \in \mathscr{F}$, this run is accepting. It can be represented graphically as in Figure 6. ■

As in finite-state automata, we now need to specify how to read a graph in an automaton. This is done by linking the intermediate configurations of a run to vertices in the graph, and by imposing conditions to match transitions with labelled edges of the graph.

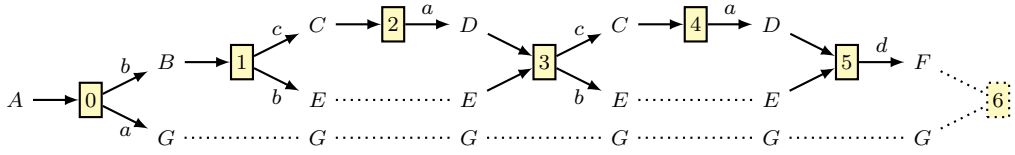


Figure 6: An accepting run in the automaton from Figure 5.

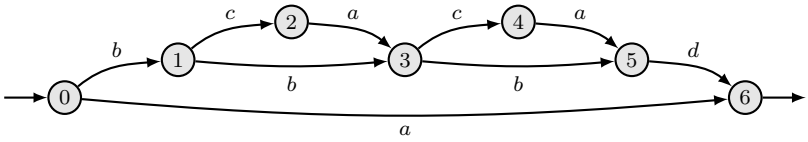


Figure 7: Trace of the run depicted in Figure 6.

Definition 11 (Reading along a run, parallel reading, language of a run)
A *reading* of $G = \langle V, E, \iota, o\rangle$ along a run $\xi = \langle (\xi_k)_{0\leqslant k\leqslant n}, (s_k, t_k)_{0\leqslant k<n}\rangle$ is a sequence $(\rho_k)_{0\leqslant k\leqslant n}$ such that for all k, ρ_k is a map from ξ_k to V_w, $\rho_0(\xi_0) = \{\iota\}$, $\rho_n(\xi_n) = \{o\}$, and $\forall k < n$, the following holds:

- all tokens in the input of the transition are mapped to the same vertex in the graph: $\forall p, q \in s_k,\ \rho_k(p) = \rho_k(q)$;
- the images of tokens in ξ_k that are not in the input of the transition are unchanged: $\forall p \in \xi_k \setminus s_k,\ \rho_k(p) = \rho_{k+1}(p)$,
- each pair in the output of the transition can be mapped to an edge of the graph with the same label: $\forall p \in s_k, \forall (x, q) \in t_k, (\rho_k(p), x, \rho_{k+1}(q)) \in E$.

Similarly, we define a *parallel reading* ρ along some parallel run $\Xi = \langle (\Xi_k)_{0\leqslant k\leqslant n}, (T_k)_{0\leqslant k<n}\rangle$ by requiring that: $\rho_0(\Xi_0) = \{\iota\}$, $\rho_n(\Xi_n) = \{o\}$, and $\forall k < n$ the following holds:

- $\forall p \in \Xi_k \setminus \bigcup_{(s,t)\in T_k} s,\ \rho_{k+1}(p) = \rho_k(p)$;
- $\forall (s, t) \in T_k, \forall p, q \in s, \rho_k(p) = \rho_k(q)$;
- $\forall (s, t) \in T_k, \forall p \in s, \forall (x, q) \in t, (\rho_k(p), x, \rho_{k+1}(q)) \in E$.

The *language of a run* ξ, denoted by $\mathscr{L}(\xi)$ is the set of graphs that can be read along ξ. ∗

The language of a Petri automaton is finally obtained by considering all accepting runs.

Definition 12 (Language recognised by a Petri automaton)
The language recognised by $\mathscr{A}$, written $\mathscr{L}(\mathscr{A})$, is the following set of graphs:

$$\mathscr{L}(\mathscr{A}) \coloneqq \bigcup_{\xi \text{ accepting in } \mathscr{A}} \mathscr{L}(\xi)\ .$$ ∗

The language of a run ξ can be characterised using a single graph which we call the *trace* of ξ: graphs are accepted by ξ exactly when they are smaller than the trace of ξ, according to $\blacktriangleleft$ (Lemma 14 below). For instance the trace of the run presented in Figure 6 is shown in Figure 7.

This trace is constructed by creating a vertex k for each transition $\tau_k = (s_k, t_k)$ of the run, plus a final vertex n. We add an edge (k, x, l) whenever there is some place q such that $(x, q) \in t_k$, and τ_l is the first transition after τ_k in the run with q among its inputs, or $l = n$ if there is no such transition in the run. Formally:

Definition 13 (Trace of a run)
Let $\xi = \langle (\xi_k)_{0\leqslant k\leqslant n}, (s_k,t_k)_{0\leqslant k<n}\rangle$ be run. For an index $k \leqslant n$ and a place q, let $\nu(k,q)$ be either the smallest index l such that $k \leqslant l$ and $q \in s_l$, or n if there is no such index. The *trace of* ξ is the graph $\mathrm{Tr}(\xi) \coloneqq \langle \{0,\dots,n\}, E_\xi, 0, n\rangle$ with $E_\xi \coloneqq \{(k,x,\nu(k+1,q)) \mid (x,q) \in t_k\}$. We write $\mathrm{Tr}(\mathscr{A})$ for the set of traces associated to the accepting runs of $\mathscr{A}$. ∗

Lemma 14. *For any accepting run ξ, we have $G \in \mathscr{L}(\xi)$ if and only if $G \blacktriangleleft Tr(\xi)$.*

Proof. Suppose there exists a graph morphism h from $\mathrm{Tr}(\xi)$ to G. Then we can build a reading by defining $\rho_k(p) = h(\nu(k,p))$ for $0 \leqslant k \leqslant n$ and $p \in \xi_k$. On the other hand, if we have a reading $(\rho_k)_{0\leqslant k\leqslant n}$ of G, we can build a morphism h by letting $h(k) = \rho_k(p)$ for any $p \in s_k$. As $(\rho_k)_k$ is a reading, h is well defined. □

As a corollary, we obtain the following characterisation of the language of a Petri automaton.

Corollary 15. $\mathscr{L}(\mathscr{A}) = Tr(\mathscr{A})^{\blacktriangleleft}$.

The left-hand side language is defined through readings along accepting runs, which is a local and incremental notion and which allows us to define *simulations* in Section 4. By contrast, the right-hand side language is defined globally, which eases the following construction of an automaton recognising the language of an expression.

3. From expressions to automata

We now show how to associate to any expression $e \in \mathrm{Reg}_X^{\wedge-}$ an automaton $\mathscr{A}(e)$ that recognises the language $G(e)^{\blacktriangleleft}$. In fact the produced automaton has an even stronger connection with e: the graphs in $G(e)$ are exactly the traces of accepting runs in $\mathscr{A}(e)$.

Definition 16
To each expression $e \in \mathrm{Reg}_X^{\wedge-}$, we associate a Petri automaton $\mathscr{A}(e)$ defined inductively as follows:

- $\mathscr{A}(x) \coloneqq \langle \{0,1\}, \{(\{0\},\{(x,1)\})\}, 0, \{\{1\}\}\rangle$
- $\mathscr{A}(\mathbb{0}) \coloneqq \langle \{0\}, \varnothing, 0, \varnothing\rangle$
- $\mathscr{A}(e_1 \vee e_2) \coloneqq \langle P_1 \cup P_2, \mathscr{T}, \iota_1, \mathscr{F}_1 \cup \mathscr{F}_2\rangle$ with $\mathscr{T} \coloneqq \mathscr{T}_1 \cup \mathscr{T}_2 \cup \{(\{\iota_1\},t) \mid (\{\iota_2\},t) \in \mathscr{T}_2\}$.
- $\mathscr{A}(e_1 \cdot e_2) \coloneqq \langle P_1 \cup P_2, \mathscr{T}, \iota_1, \mathscr{F}_2\rangle$ with $\mathscr{T} \coloneqq \mathscr{T}_1 \cup \mathscr{T}_2 \cup \{(f,t) \mid f \in \mathscr{F}_1 \text{ and } (\{\iota_2\},t) \in \mathscr{T}_2\}$.
- $\mathscr{A}(e_1^+) \coloneqq \langle P_1, \mathscr{T}, \iota_1, \mathscr{F}_1\rangle$ with $\mathscr{T} \coloneqq \mathscr{T}_1 \cup \{(f,t) \mid f \in \mathscr{F}_1 \text{ and } (\{\iota_1\},t) \in \mathscr{T}_1\}$.
- $\mathscr{A}(e_1 \wedge e_2) \coloneqq \langle P_1 \cup P_2, \mathscr{T}, \iota_1, \mathscr{F}\rangle$ with $\mathscr{F} \coloneqq \{f_1 \cup f_2 \mid f_1 \in \mathscr{F}_1,\ f_2 \in \mathscr{F}_2\}$ and

$$\mathscr{T} \coloneqq \{(s,t) \mid i \in \{1,2\},\ (s,t) \in \mathscr{T}_i,\ \iota_i \notin s\} \cup \{(\{\iota_1\}, t_1 \cup t_2) \mid (\{\iota_1\},t_1) \in \mathscr{T}_1,\ (\{\iota_2\},t_2) \in \mathscr{T}_2\}$$

(In the inductive cases, we assume that $\mathscr{A}(e_i) = \langle P_i, \mathscr{T}_i, \iota_i, \mathscr{F}_i\rangle$ for $i \in \{1,2\}$, with $P_1 \cap P_2 = \varnothing$.) ∗

We prove by induction on e that $\mathscr{A}(e)$ is indeed a Petri automaton; for the one-boundedness assumption, we add to the induction hypothesis the fact that for any configuration ξ accessible in $\mathscr{A}(e)$, if there is a final configuration $f \in \mathscr{F}$ such that $f \subseteq \xi$, then $f = \xi$. Another invariant is that the initial place never appears in a final configuration, nor in the outputs of any transition. Note that the place ι_2 becomes unreachable by construction in the cases for union, composition and intersection, so that it could safely be removed, together with the associated transitions. One can also check that the number of places in the produced automaton is linear in the size of the input expression.

Theorem 17 (Correctness)**.** *For all expression* $e \in Reg_X^{\wedge -}$, $\mathscr{L}(\mathscr{A}(e)) = G(e)^{\blacktriangleleft}$.

Proof. As explained above, we prove a stronger result, namely $\mathrm{Tr}(\mathscr{A}(e)) = G(e)$ (up to graph isomorphisms). This allows us to conclude thanks to Corollary 15. □

Remark 18. If e is an expression without intersection, it can be shown that the transitions in $\mathscr{A}(e)$ are all of the form $(\{p\}, \{(x, q)\})$, with only one input and one output. In consequence, the accessible configurations are singletons, and the resulting Petri automaton has the structure of a Non-deterministic Finite-state Automaton (NFA). Actually, in that case, the construction we described above is just a variation on Thompson's construction [16], with inlined epsilon transition elimination.

Combined with Theorem 6 from Section 1, the above theorem allows us to reduce the problem of deciding whether $Rel \vDash e = f$ to the problem of checking whether $\mathscr{L}(\mathscr{A}(e_1)) = \mathscr{L}(\mathscr{A}(e_2))$. By symmetry, it then suffices to decide inclusion of Petri automata languages.

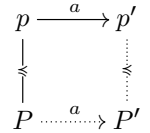
4. Comparing automata

We want to compare automata by testing if any graph accepted by $\mathscr{A}_1$ is also accepted by $\mathscr{A}_2$. Let us go back to standard non-deterministic finite-state automata (NFA), to find intuitions. In that setting an automaton over some alphabet Σ is defined by $\mathscr{A} = \langle Q, \iota, F, \Delta \rangle$ where Q is a finite set of states, ι is an initial state, F is a set of finite states F and Δ is a set of transitions of the form (p, a, q) where p and q are states and a is a letter. Consider two automata $\mathscr{A}_1 = \langle Q_1, \iota_1, F_1, \Delta_1 \rangle$ and $\mathscr{A}_2 = \langle Q_2, \iota_2, F_2, \Delta_2 \rangle$. $L(\mathscr{A}_1) \subseteq L(\mathscr{A}_2)$ means that for any word $w = a_1 \cdots a_n$ accepted by $\mathscr{A}_1$, w is also accepted by $\mathscr{A}_2$. Thus if there is an execution in $\mathscr{A}_1$ recognising w then there is an execution in $\mathscr{A}_2$ recognising w. One can then put them together side by side like so:

$$\begin{array}{lllllll} \iota_1 & \xrightarrow{a_1}_{\mathscr{A}_1} & p_1 & \xrightarrow{a_2}_{\mathscr{A}_1} & \cdots & \xrightarrow{a_n}_{\mathscr{A}_1} & p_n \in F_1 \\ \iota_2 & \xrightarrow{a_1}_{\mathscr{A}_2} & q_1 & \xrightarrow{a_2}_{\mathscr{A}_2} & \cdots & \xrightarrow{a_n}_{\mathscr{A}_2} & q_n \in F_2 \end{array}$$

Thus trying to prove that $L(\mathscr{A}_1) \subseteq L(\mathscr{A}_2)$ amounts to finding a method to build from any run in $\mathscr{A}_1$ a corresponding run in $\mathscr{A}_2$. This can be done by computing a *simulation* between the automaton $\mathscr{A}_1$ and the determinised automaton of $\mathscr{A}_2$. A simulation between these automata is then a relation $\preccurlyeq \subseteq Q_1 \times \mathcal{P}(Q_2)$ such that

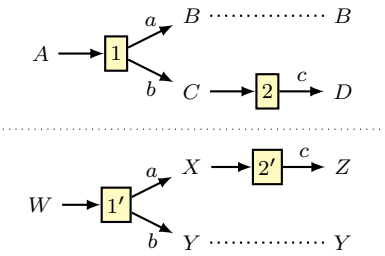
- $\iota_1 \preccurlyeq \{\iota_2\}$, and if $p \preccurlyeq P$ and $p \in F_1$, then $P \cap F_2 \neq \varnothing$;
- if $p \preccurlyeq P$ and $(p, a, p') \in \Delta_1$ then $p' \preccurlyeq P'$, where $P' \coloneqq \{p' \mid (p, a, p') \in \Delta_2,\ p \in P\}$.

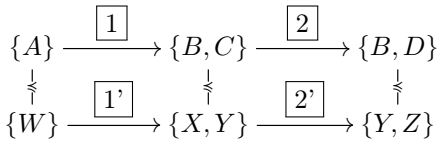


If such a relation can be found, then for any accepting execution in $\mathscr{A}_1$, we can use the relation to extract a corresponding execution in $\mathscr{A}_2$. It is also possible to prove that if the language of $\mathscr{A}_1$ is indeed included in the language of $\mathscr{A}_2$, then such a relation exists. This gives us an algorithm to decide the inclusion of languages, because the set of states of both automata being finite, there is only a finite number of candidates for $\preccurlyeq$. More realistically, one can define a coinductive algorithm that computes a simulation relation on-the-fly.

We follow a similar approach for Petri automata, but we need to make several important adjustments. Consider two automata $\mathscr{A}_1 = \langle P_1, \mathscr{T}_1, \iota_1, \mathscr{F}_1 \rangle$ and $\mathscr{A}_2 = \langle P_2, \mathscr{T}_2, \iota_2, \mathscr{F}_2 \rangle$, we try to show that for any graph G accepted by $\mathscr{A}_1$, G is recognised by $\mathscr{A}_2$. By Lemma 14, this amounts to proving that for any accepting run ξ in $\mathscr{A}_1$, $\mathrm{Tr}(\xi)$ is recognised by some accepting run ξ' in $\mathscr{A}_2$. Leaving non-determinism apart, the first idea that comes to mind is to find a relation between the configurations in $\mathscr{A}_1$ and the configurations in $\mathscr{A}_2$, that satisfy some conditions on the initial and final configurations, and such that if $\xi_k \preccurlyeq \xi'_k$ and $\xi_k \xrightarrow{\tau}_{\mathscr{A}_1} \xi_{k+1}$, then there is a configuration ξ'_{k+1} in $\mathscr{A}_2$ such that $\xi_{k+1} \preccurlyeq \xi'_{k+1}$,

$\xi'_k \xrightarrow{\tau'}_{\mathscr{A}_2} \xi'_{k+1}$, and these transition steps are compatible in some sense. However, such a definition will not give us the result we are looking for. Consider the two runs on the left-hand side:

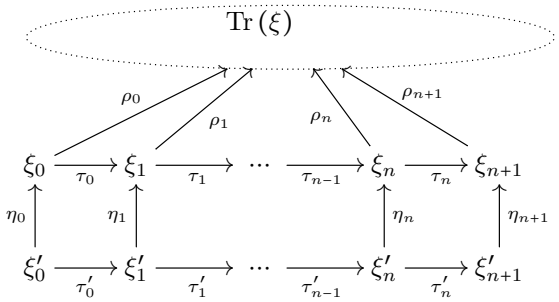




The trace of the first run corresponds to the ground term $a \wedge (b \cdot c)$, and the trace of the second one corresponds to $(a \cdot c) \wedge b$. These two terms are incomparable, but the relation $\preccurlyeq$ depicted on the right-hand side satisfies the previously stated conditions.

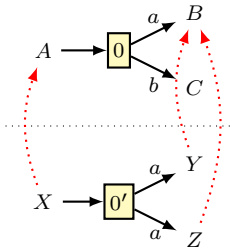
The problem here is that in Petri automata, runs are token firing games. To adequately compare two runs, we need to closely track the tokens. For this reason, we will relate a configuration ξ_k in $\mathscr{A}_1$ not only to a configuration ξ'_k in $\mathscr{A}_2$, but to a map η_k from ξ'_k to ξ_k. This will enable us to associate with each token situated on some place in P_2 another token placed on $\mathscr{A}_1$.

We want to find a reading of $\mathrm{Tr}\,(\xi)$ in $\mathscr{A}_2$, i.e. a run in $\mathscr{A}_2$ together with a sequence of maps associating places in $\mathscr{A}_2$ to positions in $\mathrm{Tr}\,(\xi)$. Consider the picture below. Since we already have a reading of $\mathrm{Tr}\,(\xi)$ along ξ (by defining $\rho_k(p) = \nu(k, p)$, as in the proof of Lemma 14), it suffices to find maps from the places in $\mathscr{A}_2$ to the places in $\mathscr{A}_1$ (the maps η_k): the reading of $\mathrm{Tr}\,(\xi)$ in $\mathscr{A}_2$ will be obtained by composing η_k with ρ_k.

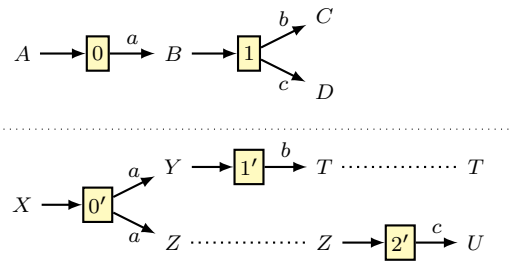


We need to impose some constraints on the maps (η_k) to ensure that $(\rho_k \circ \eta_k)_{0 \leqslant k \leqslant n}$ is indeed a correct reading in $\mathscr{A}_2$. First, we need to ascertain that a transition τ'_k in $\mathscr{A}_2$ may be fired from the reading state $\rho_k \circ \eta_k$ to reach the reading state $\rho_{k+1} \circ \eta_{k+1}$. Furthermore, as for NFA, we want transitions τ_k and τ'_k to be related: specifically, we require τ'_k to be included (via the morphisms η_k and η_{k+1}) in the transition τ_k. This is meaningful because transition τ_k contains a lot of information about the vertex k of $\mathrm{Tr}\,(\xi)$ and about ρ: the labels of the outgoing edges from k are the labels on the output of τ_k, and the only places that will ever be mapped to k in the reading ρ are exactly the places in the input of τ_k.

This already shows an important difference between the simulations for NFA and Petri automata. For NFA, we relate a transition $p \xrightarrow{a} p'$ to a transition $q \xrightarrow{a} q'$ with the same label a. Here the transitions $\xi_k \xrightarrow{\tau_k}_{\mathscr{A}_1} \xi_{k+1}$ may have different labels. Consider the step represented on the right, corresponding to a square in the above diagram. The output of $\boxed{0}$ has a label b that does not appear in $\boxed{0'}$, and $\boxed{0'}$ has two outputs labelled by a. Nevertheless this satisfies the conditions informally stated above, indeed, $a \wedge b \leqslant a \wedge a$ holds.



However this definition is not yet satisfactory. Consider the two runs below:



Their traces correspond respectively to the ground terms $a \cdot (b \wedge c)$ and $(a \cdot b) \wedge (a \cdot c)$. The problem is that $a \cdot (b \wedge c) \leqslant (a \cdot b) \wedge (a \cdot c)$, but with the previous definition, we cannot relate these runs: they do not have the same length. The solution here consists in grouping the transitions $\boxed{1'}$ and $\boxed{2'}$ together, and consider these two steps as a single step in a *parallel run*. This last modification gives us a notion of simulation we can really work with.

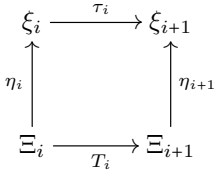
Definition 19 (Simulation)
A relation $\preccurlyeq \subseteq \mathcal{P}(P_1) \times \mathcal{P}(P_2 \rightharpoonup P_1)$ between the configurations of $\mathscr{A}_1$ and the partial maps from the places of $\mathscr{A}_2$ to the places of $\mathscr{A}_1$ is called a *simulation* between $\mathscr{A}_1$ and $\mathscr{A}_2$ if:

- $\{\iota_1\} \preccurlyeq \{[\iota_2 \mapsto \iota_1]\}$;
- if $\xi \preccurlyeq E$ and $\xi \xrightarrow{(s,t)}_{\mathscr{A}_1} \xi'$, then $\xi' \preccurlyeq E'$ where E' is the set of all η' such that there is some $\eta \in E$ and a compatible set of transitions $T \subseteq \mathscr{T}_2$ such that:
  - $\mathrm{dom}(\eta) \xrightarrow{T}_{\mathscr{A}_2} \mathrm{dom}(\eta')$;
  - $\forall (s', t') \in T, \eta(s') \subseteq s$ and $\forall (x, q) \in t', (x, \eta'(q)) \in t$;
  - $\forall p \in \mathrm{dom}(\eta), (\forall (s', t') \in T, p \notin s') \Rightarrow \eta(p) = \eta'(p)$.
- if $\xi \preccurlyeq E$ and $\xi \in \mathscr{F}_1$, then there must be some $\eta \in E$ such that $\mathrm{dom}(\eta) \in \mathscr{F}_2$. *

We will now prove that the language of $\mathscr{A}_1$ is contained in the language of $\mathscr{A}_2$ if and only if there exists such a simulation. We first introduce the following notion of embedding.

Definition 20 (Embedding)
Let $\xi = \langle (\xi_k)_{0 \leqslant k \leqslant n}, (\tau_k)_{0 \leqslant i < n} \rangle$ be a run in $\mathscr{A}_1$, and $\Xi = \langle (\Xi_k)_{0 \leqslant k \leqslant n}, (T_i)_{0 \leqslant i < n} \rangle$ a parallel run in $\mathscr{A}_2$. An *embedding* of Ξ into ξ is a sequence $(\eta_i)_{0 \leqslant i \leqslant n}$ of maps such that for any $i < n$, we have:

- η_i is a map from Ξ_i to ξ_i;
- the image of T_i by η_i is included in τ_i, meaning that for any $(s, t) \in T_i$, for any $p \in s$ and $(x, q) \in t$, $\eta_i(p)$ is contained in the input of τ_i and $(x, \eta_{i+1}(q))$ is in the output of τ_i;
- the image of the tokens in Ξ_i that do not appear in the input of T_i are preserved ($\eta_i(p) = \eta_{i+1}(p)$) and their image is not in the input of τ_i.



*

Figure 8 illustrates the embedding of some parallel run, with trace $((b \cdot c \cdot a \cdot b) \wedge (b \cdot b \cdot c \cdot a)) \cdot d$, into the run presented in Figure 6. Notice that is it necessary to have a parallel run instead of a simple one: to find something that matches transition $\boxed{1}$, we need to fire two transitions in parallel.

There is a close relationship between simulations and embeddings:

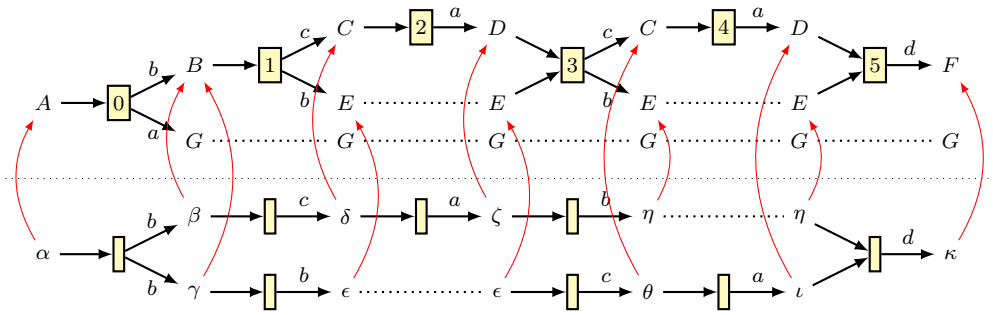


Figure 8: Embedding of a parallel run into the run from Figure 6.

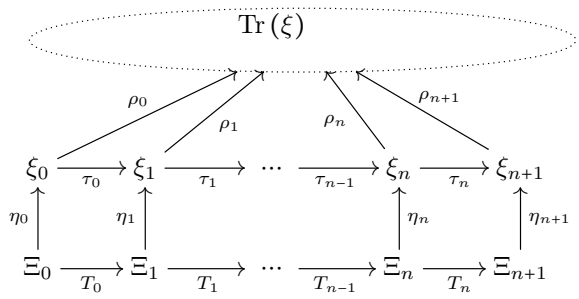
Lemma 21. *Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be two Petri automata, the following are equivalent:*

1. *there exists a simulation $\preccurlyeq$ between $\mathscr{A}_1$ and $\mathscr{A}_2$;*
2. *for any accepting run ξ in $\mathscr{A}_1$, there is an accepting parallel run Ξ in $\mathscr{A}_2$ that can be embedded into ξ.*

Proof. If we have a simulation $\preccurlyeq$, let $\xi = \langle (\xi_k)_{0\leqslant k\leqslant n}, (\tau_k)_{0\leqslant k<n}\rangle$ be an accepting run in $\mathscr{A}_1$. By the definition of simulation, we can find a sequence of sets of maps $(E_k)_{0\leqslant k\leqslant n}$ such that $E_0 = \{[\iota_2 \mapsto \iota_1]\}$ and $\forall k, \xi_k \preccurlyeq E_k$. Furthermore, we can extract from this a sequence of maps $(\eta_k)_{0\leqslant k\leqslant n}$ and a sequence of parallel transitions $(T_k)_{0\leqslant k<n}$ such that (η_k) is an embedding of $\langle (\mathrm{dom}\,(\eta_k))_{0\leqslant k\leqslant n}, (T_k)_{0\leqslant k<n}\rangle$ (which is accepting) into ξ. This follows directly from the definitions of embedding and simulation.

On the other hand, if we have property *2.*, then we can define a relation $\preccurlyeq$ by saying that $\xi \preccurlyeq E$ if there is an accepting run $\xi' = \langle (\xi'_k)_{0\leqslant k\leqslant n}, (\tau_k)_{0\leqslant k<n}\rangle$ in $\mathscr{A}_1$ such that there is an index k_0: $\xi = \xi'_{k_0}$; and the following holds: $\eta \in E$ if there is an accepting parallel run $\Xi = \langle (\Xi_k)_{0\leqslant k\leqslant n}, (T_k)_{0\leqslant k<n}\rangle$ and $(\eta'_k)_{0\leqslant k\leqslant n}$ an embedding of Ξ into ξ such that $\eta = \eta'_{k_0}$. It is then immediate to check that $\preccurlyeq$ is indeed a simulation. □

If η is an embedding of Ξ into ξ, we can easily check that $(\rho_i \circ \eta_i)_{0\leqslant i\leqslant n}$ is a parallel reading of $\mathrm{Tr}\,(\xi)$ along Ξ in $\mathscr{A}_2$, as illustrated by this diagram:



Thus, it is clear that once we have such a run Ξ with the sequence of maps η, we have that $\mathrm{Tr}\,(\xi)$ is indeed in the language of $\mathscr{A}_2$. The more difficult question is the completeness of this approach: if $\mathrm{Tr}\,(\xi)$ is recognised by $\mathscr{A}_2$, is it always the case that we can find a run Ξ that may be embedded into ξ? The answer is affirmative, thanks to Lemma 23 below. If $(\rho_j)_{0\leqslant j\leqslant n}$ is a reading of G along $\xi = \langle (\xi_k)_{0\leqslant k\leqslant n}, (s_k, t_k)_{0\leqslant k<n}\rangle$, we write $\mathit{active}(j)$ for the only position in $\rho_j(s_j)$[1].

[1]Recall that if $(\rho_j)_{0\leqslant j\leqslant n}$ is a reading along ξ then $\forall p, q \in s_j, \rho_j(p) = \rho_j(q)$.

Definition 22 (Consistent ordering)
$\leqslant$ is a *consistent ordering* on $G = \langle V, E, \iota, o\rangle$ if $\langle V, \leqslant\rangle$ is a linear order and $(p, x, q) \in E$ entails $p \leqslant q$. ∗

Lemma 23. *Let $G \in \mathscr{L}(\mathscr{A}_2)$ and $\leqslant$ be any consistent ordering on G. Then there exists a run ξ and a reading $(\rho_j)_{0\leqslant j\leqslant n}$ of G along ξ such that $\forall k, active(k) \leqslant active(k+1)$.*

Proof. The proof of this result is achieved by taking any run ξ accepting G, and then exchanging transitions in ξ according to $\leqslant$, while preserving the existence of a reading. The details of this proof being a bit technical, we omit them here. □

Notice that if G contains cycles, this lemma cannot apply because of the lack of consistent ordering. This enables us to build an embedding from any reading of $\mathrm{Tr}(\xi)$ in $\mathscr{A}_2$.

Lemma 24. *Let ξ a accepting run of $\mathscr{A}_1$. Then $Tr(\xi)$ is in $\mathscr{L}(\mathscr{A}_2)$ if and only if there is an accepting parallel run in $\mathscr{A}_2$ that can be embedded into ξ.*

Proof. We do not include the details of this proof here for length reasons.

For the if direction, we build a parallel reading from the embedding, as explained above. For the other direction, we consider a reading of $\mathrm{Tr}(\xi)$ in $\mathscr{A}_2$ along some run ξ'. Notice that the natural ordering on $\mathbb{N}$ is consistent for $\mathrm{Tr}(\xi)$; we may thus change the order of the transitions in ξ' (using Lemma 23) and group them adequately to obtain a parallel reading Ξ that embeds in ξ. □

So we know that the existence of embeddings is equivalent to the inclusion of languages, and we previously established that it is also equivalent to the existence of a simulation relation. Hence, the following characterisation holds:

Theorem 25. *Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be two Petri automata. $\mathscr{L}(\mathscr{A}_1) \subseteq \mathscr{L}(\mathscr{A}_2)$ if and only if there exists a simulation relation $\preccurlyeq$ between $\mathscr{A}_1$ and $\mathscr{A}_2$.*

Proof. By Lemmas 14, 21 and 24. □

As Petri automata are finite, there are finitely many relations in $\mathcal{P}(\mathcal{P}(P_1) \times \mathcal{P}(P_2 \dashrightarrow P_1))$. The existence of a simulation thus is decidable, allowing us to prove the main result:

Theorem 26. *Given two expressions $e, f \in Reg^{\wedge -}_X$, testing whether $Rel \vDash e = f$ is decidable.*

Proof. By Theorems 6, 17 and 25, and reasoning by double inclusion. □

In practice, we may build the simulation on-the-fly, starting from the pair $(\{\iota_1\}, \{[\iota_2 \mapsto \iota_1]\})$ and progressing from there. We have implemented this algorithm in OCaml: even though its theoretical worst case time complexity is huge[2], we get a result almost instantaneously on simple one-line examples.

Conclusions and directions for future work

By introducing Petri automata, we proved the decidability of the equivalence of identity-free regular expressions with intersection, with respect to their relational interpretations. Actually, this also holds for their language interpretations, because Andréka et al. showed in [1] that the classes of identity-free relational Kleene lattices and identity-free language Kleene lattices coincide. They differ however when we include the identity constant, or the converse operation.

[2] A quick analysis gives a $\mathcal{O}\left(2^{n+n^m}\right)$ complexity bound, where n and m are the numbers of places of the automata.

The construction and algorithm presented here were implemented in OCaml as an exercise. The resulting program is available online as an interactive applet [5].

By adding ϵ-transitions to Petri automata, we could partly cope with the identity, in the sense that we can build automata to recognise the graph languages of expressions over the signature $\langle \vee, \wedge, \cdot, ^\star, \mathbb{1}, \mathbb{0} \rangle$. However, we did not find a way of comparing ϵ-Petri automata: the notion of simulation we described here is not equivalent to the inclusion of languages for these automata.

Similarly, we could define a variant of Petri automata for recognising the graph languages associated to expressions with converse: it suffice to consider a duplicated alphabet. However, we did not find a proper way of extending our notion of simulation to capture language inclusion of such automata.

References

[1] H. Andréka, S. Mikulás, and I. Németi. The equational theory of Kleene lattices. *Theoretical Computer Science*, 412(52):7099–7108, 2011.

[2] H. Andréka and D. Bredikhin. The equational theory of union-free algebras of relations. *Algebra Universalis*, 33(4):516–532, 1995.

[3] S. L. Bloom, Z. Ésik, and G. Stefanescu. Notes on equational theories of relations. *Algebra Universalis*, 33(1):98–126, 1995.

[4] P. Brunet and D. Pous. Kleene algebra with converse. In *Proc. RAMiCS*, volume 8428 of *Lecture Notes in Computer Science*, pages 101–118. Springer Verlag, 2014.

[5] P. Brunet and D. Pous. Web appendix to this abstract, 2014. `http://perso.ens-lyon.fr/paul.brunet/rklm.html`.

[6] J. H. Conway. *Regular algebra and finite machines*. Chapman and Hall Mathematics Series, 1971.

[7] Z. Ésik and L. Bernátsky. Equational properties of Kleene algebras of relations with conversion. *Theoretical Computer Science*, 137(2):237–251, 1995.

[8] P. J. Freyd and A. Scedrov. *Categories, Allegories*. North Holland, 1990.

[9] S. C. Kleene. *Representation of Events in Nerve Nets and Finite Automata*. Memorandum. Rand Corporation, 1951.

[10] D. Kozen. On Kleene Algebras and closed semirings. In *Proc. MFCS*, volume 452 of *Lecture Notes in Computer Science*, pages 26–47. Springer Verlag, 1990.

[11] D. Kozen. A completeness theorem for Kleene Algebras and the algebra of regular events. In *Proc. LICS*, pages 214–225. IEEE Computer Society, 1991.

[12] D. Krob. A Complete System of B-Rational Identities. In *Proc. ICALP*, volume 443 of *Lecture Notes in Computer Science*, pages 60–73. Springer Verlag, 1990.

[13] T. Murata. Petri nets: Properties, analysis and applications. *Proc. of the IEEE*, 77(4):541–580, Apr 1989.

[14] C. A. Petri. Fundamentals of a theory of asynchronous information flow. In *IFIP Congress*, pages 386–390, 1962.

[15] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Darmstadt Univ. of Tech., 1962.

[16] K. Thompson. Regular expression search algorithm. *C. of the ACM*, 11:419–422, 1968.

Kleene Algebra with Converse

Paul Brunet, Damien Pous

Kleene Algebra with Converse

Paul Brunet and Damien Pous *

LIP, CNRS, ENS Lyon, INRIA, Université de Lyon, UMR 5668

Abstract The equational theory generated by all algebras of binary relations with operations of union, composition, converse and reflexive transitive closure was studied by Bernátsky, Bloom, Ésik, and Stefanescu in 1995. We reformulate some of their proofs in syntactic and elementary terms, and we provide a new algorithm to decide the corresponding theory. This algorithm is both simpler and more efficient; it relies on an alternative automata construction, that allows us to prove that the considered equational theory lies in the complexity class PSpace.
Specific regular languages appear at various places in the proofs. Those proofs were made tractable by considering appropriate automata recognising those languages, and exploiting symmetries in those automata.

Introduction

In many contexts in computer science and mathematics operations of union, sequence or product and iteration appear naturally. *Kleene Algebra*, introduced by John H. Conway under the name *regular algebra* [Con71], provides an algebraic framework allowing to express properties of these operators, by studying the equivalence of expressions built with these connectives. It is well known that the corresponding equational theory is decidable [Kle51], and that it is complete for language and relation models.

As expressive as it may be, one may wish to integrate other usual operations in such a setting. Theories obtained this way, by addition of a finite set of equations to the axioms of Kleene Algebra, are called *Extensions of Kleene Algebra*. We shall focus here on one of these extensions, where an operation of *converse* is added to Kleene Algebra. The converse of a word is its mirror image (the word obtained by reversing the order of the letters), and the converse $R^\vee$ of a relation R is its reciprocal ($xR^\vee y \triangleq yRx$). This natural operation can be expressed simply as a set of equations that we add to Kleene Algebra's axioms.

The question that arises once this theory is built is its decidability: given two formal expressions built with the connectives product, sum, iteration and converse, can one decide automatically if they are equivalent, meaning that their equality can be proven using the axioms of the theory? Bloom, Ésik,

* Work partially funded by the french projects PiCoq (ANR-09-BLAN-0169-01) and PACE (ANR-12IS02001).

Stefanescu and Bernátsky gave an affirmative answer to that question in two articles, [BÉS95] and [ÉB95], in 1995.

However, although the algorithm they define proves the decidability result, it is too complicated to be used in actual applications. In this paper, beside some simplifications of the proofs given in [BÉS95], we give a new and more efficient algorithm to decide this problem, which we place in the complexity class PSpace.

The equational theory of Kleene algebra cannot be finitely axiomatised [Red64]. Krob presented the first purely axiomatic (but infinite) presentation [Kro90]. Several finite quasi-equational characterisations have been proposed [Sal66,Bof90, Kro90,Koz91,Bof95]; here we follow the one from Kozen [Koz91].

A Kleene Algebra is an algebraic structure $\langle K, +, \cdot, ^\star, \mathbb{0}, \mathbb{1}\rangle$ such that $\langle K, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$ is an idempotent semi-ring, and the operation * satisfies the following properties

$$\mathbb{1} + aa^\star \leqslant a^\star \tag{1a}$$
$$\mathbb{1} + a^\star a \leqslant a^\star \tag{1b}$$
$$b + ax \leqslant x \Rightarrow a^\star b \leqslant x \tag{1c}$$
$$b + xa \leqslant x \Rightarrow ba^\star \leqslant x \tag{1d}$$

(Here $a \leqslant b$ is a shorthand for $a + b = b$.)

The quasi-variety KA consists in the axioms of an idempotent semi-ring together with axioms and inference rules (1a) to (1d). Kleene Algebras are thus *models* of KA. We shall call *regular expressions over* X, written Reg_X, the expressions built from letters of X, the binary connectives $+$ and $\cdot$, the unary connective * and the two constants $\mathbb{0}$ and $\mathbb{1}$.

Two families of such algebras are of particular interest: languages (sets of finite words over a finite alphabet, with union as sum and concatenation as product) and relations (binary relations over an arbitrary set with union and composition). KA is complete for both these models [Kro90, Koz91], meaning that for any $e, f \in \mathrm{Reg}_X$, $\mathrm{KA} \vdash e = f$ if and only if e and f coincide under any language (resp. relational) interpretation. This last property will be written $e \equiv_{\mathrm{Lang}} f$ (resp. $e \equiv_{\mathrm{Rel}} f$).

More remarkably, if we denote by $[\![e]\!]$ the language denoted by an expression e, we have that for any $e, f \in \mathrm{Reg}_X$, $\mathrm{KA} \vdash e = f$ if and only if $[\![e]\!] = [\![f]\!]$. By Kleene's theorem (see [Kle51]) the equality of two regular languages can be reduced to the equivalence of two finite automata, which is easy to compute. Hence, the theory KA is decidable.

Now let us add a unary operation of converse to regular expressions. We shall denote by $\mathrm{Reg}^\vee_X$ the set of regular expressions with converse over a finite alphabet X. While doing so, several questions arise:

1. Can the converse on languages and on relations be encoded in the same theory?
2. What axioms do we need to add to KA to model these operations?

3. Are the resulting theories complete for languages and relations?
4. Are these theories decidable?

There is a simple answer to the first question: no. Indeed the equation $a \leqslant a \cdot a^\vee \cdot a$ is valid for any relation a (because if $(x,y) \in a$, then $(x,y) \in a$, $(y,x) \in a^\vee$, and $(x,y) \in a$, so that $(x,y) \in a \circ a^\vee \circ a$). But this equation is not satisfied for all languages a (for instance, with the language $a = \{x\}$, $a \cdot a^\vee \cdot a = \{xxx\}$ and $x \notin \{xxx\}$). This means that there are two distinct theories corresponding to these two families of models. Let us begin by considering the case of languages.

Theorem 1 (Completeness of KAC$^-$ [BÉS95])**.** *A complete axiomatisation of the variety Lang$^\vee$ of languages generated by concatenation, union, star, and converse consists of the axioms of KA together with axioms (2a) to (2d).*

$$(a+b)^\vee = a^\vee + b^\vee \tag{2a}$$
$$(a \cdot b)^\vee = b^\vee \cdot a^\vee \tag{2b}$$
$$(a^\star)^\vee = (a^\vee)^\star \tag{2c}$$
$$a^{\vee\vee} = a. \tag{2d}$$

We call this theory KAC$^-$; it is decidable.

As for relations, we write $e \equiv_{\mathrm{Lang}^\vee} f$ if e and f have the same language interpretations (for a formal definition, see the "Notation" subsection below). To prove this result, one first associates to any expression $e \in \mathrm{Reg}_X^\vee$ an expression $\mathbf{e} \in \mathrm{Reg}_{\mathbf{X}}$, where $\mathbf{X}$ is an alphabet obtained by adding to X a disjoint copy of itself. Then, one proves that the following implications hold.

$$e \equiv_{\mathrm{Lang}^\vee} f \quad \Rightarrow \quad [\![\mathbf{e}]\!] = [\![\mathbf{f}]\!] \tag{3}$$
$$[\![\mathbf{e}]\!] = [\![\mathbf{f}]\!] \quad \Rightarrow \quad \mathrm{KAC}^- \vdash e = f \tag{4}$$

(That KAC$^- \vdash e = f$ entails $e \equiv_{\mathrm{Lang}^\vee} f$ is obvious; decidability comes from that of regular languages equivalence.) We reformulate Bloom et al.'s proofs of these implications in elementary terms in Section 1.1.

As stated before, the equation $a \leqslant a \cdot a^\vee \cdot a$ provides a difference between languages with converse and relations with converse. It turns out that it is the only difference, in the sense that the following theorem holds:

Theorem 2 (Completeness of KAC [BÉS95, ÉB95])**.** *A complete axiomatisation of the variety Rel$^\vee$ of relations generated by composition, union, star, and converse consists of the axioms of KAC$^-$ together with the axiom (5).*

$$a \leqslant a \cdot a^\vee \cdot a. \tag{5}$$

We call this theory KAC; it is decidable.

The proof of this result also relies on a translation into regular languages. Ésik et al. define a notion of *closure*, written $\mathcal{cl}()$, for languages over $\mathbf{X}$, and they prove the following implications:

$$e \equiv_{\mathrm{Rel}^\vee} f \quad \Rightarrow \quad \mathcal{cl}(\llbracket \mathbf{e} \rrbracket) = \mathcal{cl}(\llbracket \mathbf{f} \rrbracket) \tag{6}$$

$$\mathcal{cl}(\llbracket \mathbf{e} \rrbracket) = \mathcal{cl}(\llbracket \mathbf{f} \rrbracket) \quad \Rightarrow \quad \mathrm{KAC} \vdash e = f \tag{7}$$

(Again, that $\mathrm{KAC} \vdash e = f$ entails $e \equiv_{\mathrm{Rel}^\vee} f$ is obvious.) The first implication (6) was proven in [BÉS95]; we give a new formulation of this proof in Section 1.2. The second one (7) was proven in [ÉB95].

The last consideration is the decidability of KAC. To this end, Bloom et al. propose a construction to obtain an automaton recognising $\mathcal{cl}(L)$, when given an automaton recognising L. Decidability follows: to decide whether $\mathrm{KAC} \vdash e = f$ one can build two automata recognising $\mathcal{cl}(\llbracket \mathbf{e} \rrbracket)$ and $\mathcal{cl}(\llbracket \mathbf{f} \rrbracket)$ and check if they are equivalent. Unfortunately, their construction tends to produce huge automata, which makes it useless for practical application. We propose a new and simpler one in Section 2; by analysing this construction, we show in Section 3 how it leads to a proof that the problem of equivalence in KAC is PSpace.

Notation

For any word w, $|w|$ is the size of w, meaning its number of letters; for any $1 \leqslant i \leqslant |w|$, we'll write $w(i)$ for the i^{th} letter of w and $w|_i \triangleq w(1)w(2)\cdots w(i)$ for its prefix of size i. Also, $\mathrm{suffixes}(w) \triangleq \{v \mid \exists u : uv = w\}$ is the set of all suffixes of w. A deterministic automaton is a tuple $\langle Q, \Sigma, q_0, T, \delta\rangle$; with Q a set of states, Σ an alphabet, $q_0 \in Q$ an initial state, $T \subseteq Q$ a set of final states and $\delta : Q \times \Sigma \to Q$ a transition function. A non-deterministic automaton is a tuple $\langle Q, \Sigma, I, T, \Delta\rangle$; with Q, Σ and T same as before, $I \subseteq Q$ a set of initial states and $\Delta \subseteq Q \times \Sigma \times Q$ a set of transitions. We write $L(\mathscr{A})$ for the *language recognised by the automaton* $\mathscr{A}$. For any $a \in \Sigma$, we write $\Delta(a)$ for $\{(p,q) \mid (p,a,q) \in \Delta\}$. We also use the compact notation $p \xrightarrow{w}_{\mathscr{A}} q$ to denote that there is in the automaton $\mathscr{A}$ a path labelled by w from the state p to the state q. For a set $E \subseteq Q$ and a relation R over Q, we write $E \cdot R$ for the set $\{y \mid \exists x \in E : xRy\}$.

Given a map σ from a set X to the languages on an alphabet Σ (resp. the relations on a set S), there is a unique extension of σ into a homomorphism from Reg_X to Lang_Σ (resp. Rel_S), which we denote by $\widehat{\sigma}$. The same thing can be done with regular expressions with converse, and we will use the same notation for it. We finally denote by $\equiv_V$ the equality in a variety V (Lang, Rel, $\mathrm{Lang}^\vee$ or $\mathrm{Rel}^\vee$): $e \equiv_V f \triangleq \forall K, \forall \sigma : X \to V_K, \widehat{\sigma}(e) = \widehat{\sigma}(f)$.

1 Preliminary material

1.1 Languages with converse: theory $\mathbf{KAC}^-$

We consider regular expressions with converse over a finite alphabet X. The alphabet $\mathbf{X}$ is defined as $X \cup X'$, where $X' \triangleq \{x' \mid x \in X\}$ is a disjoint copy

of X. As a shorthand, we use $'$ as an internal operation on $\mathbf{X}$ going from X to X' and from X' to X such that if $x \in X$, $x' \triangleq x' \in X'$ and $(x')' \triangleq x \in X$. An important operation in the following is the translation of an expression $e \in \mathrm{Reg}_X^\vee$ to an expression $\mathbf{e} \in \mathrm{Reg}_{\mathbf{X}}$. We proceed to its definition in two steps.

Let $\tau(e)$ denote the normal form of an expression $e \in \mathrm{Reg}_X^\vee$ in the following convergent term rewriting system:

$$\begin{array}{lll} (a+b)^\vee \to a^\vee + b^\vee & \mathbb{0}^\vee \to \mathbb{0} & (a^\star)^\vee \to (a^\vee)^\star \\ (a \cdot b)^\vee \to b^\vee \cdot a^\vee & \mathbb{1}^\vee \to \mathbb{1} & a^{\vee\vee} \to a \end{array}$$

The corresponding equations being derivable in KAC^-, one easily obtain that

$$\forall e \in \mathrm{Reg}_X^\vee,\ \mathrm{KAC}^- \vdash \tau(e) = e \tag{8}$$

We finally denote by $\mathbf{e}$ the expression obtained by further applying the substitution $\nu \triangleq [x^\vee \mapsto x', (\forall x \in \mathbf{X})]$, i.e., $\mathbf{e} \triangleq \nu(\tau(e))$. (Note that $\mathbf{e} \in \mathrm{Reg}_{\mathbf{X}}$: it is regular, all occurrences of the converse operation have been eliminated.) As explained in the introduction, Bloom et al.'s proof [BÉS95] amounts to proving the implications (3) and (4). We include a syntactic and elementary presentation of this proof, for the sake of completeness.

Lemma 3. *For all $e, f \in Reg_X^\vee$, $e \equiv_{Lang^\vee} f$ entails $[\![\mathbf{e}]\!] = [\![\mathbf{f}]\!]$.*

Proof. For any $e \in \mathrm{Reg}_X^\vee$, we have $\tau(e) \equiv_{\mathrm{Lang}^\vee} e$ (†) as an immediate consequence of (8). Let us write $X_\bullet \triangleq X \uplus \{\bullet\}$ and consider the following interpretations (which appear in [BÉS95, proof of Proposition 4.3]):

$$\begin{array}{rl} \mu : X \longrightarrow & \mathcal{P}\left(X_\bullet^\star\right) \\ x \longmapsto & \{x \cdot \bullet\} \end{array} \qquad \begin{array}{rl} \eta : \mathbf{X} \longrightarrow & \mathcal{P}\left(X_\bullet^\star\right) \\ x \in X \longmapsto & \{x \cdot \bullet\} \\ x' \in X' \longmapsto & \{\bullet \cdot x\} \end{array}$$

One can check (see Appendix A.1)that $\widehat{\eta}$ is injective modulo equality of denoted languages, in the sense that for any expression $e \in \mathrm{Reg}_{\mathbf{X}}$, we have

$$\widehat{\eta}(e) = \widehat{\eta}(f) \text{ implies that } [\![e]\!] = [\![f]\!]\ . \tag{9}$$

By a simple induction on e, we get $\widehat{\mu}(\tau(e)) = \widehat{\eta}(\nu(\tau(e))) = \widehat{\eta}(\mathbf{e})$. Combined with (†), we deduce that $\widehat{\mu}(e) = \widehat{\eta}(\mathbf{e})$. All in all, we obtain: $e \equiv_{\mathrm{Lang}^\vee} f \Rightarrow \widehat{\mu}(e) = \widehat{\mu}(f) \Rightarrow \widehat{\eta}(\mathbf{e}) = \widehat{\eta}(\mathbf{f}) \Rightarrow [\![\mathbf{e}]\!] = [\![\mathbf{f}]\!]$. □

The second implication is even more immediate, using KA completeness.

Lemma 4. *For all $e, f \in Reg_X^\vee$, if $[\![\mathbf{e}]\!] = [\![\mathbf{f}]\!]$ then $KAC^- \vdash e = f$.*

Proof. By completeness of KA [Kro90, Koz91], if $[\![\mathbf{e}]\!] = [\![\mathbf{f}]\!]$, then we know that there is a proof $\pi_1 : \mathrm{KA} \vdash \mathbf{e} = \mathbf{f}$. As KA is contained in KAC^-, the same proof can be seen as $\pi_1 : \mathrm{KAC}^- \vdash \mathbf{e} = \mathbf{f}$. By substituting x' by $(x^\vee)$ everywhere in this proof, we get a new proof $\pi_2 : \mathrm{KAC}^- \vdash \tau(e) = \tau(f)$. By (8) and transitivity we thus get $\mathrm{KAC}^- \vdash e = f$. □

We finally deduce that $e \equiv_{\mathrm{Lang}^\vee} f \Leftrightarrow [\![\mathbf{e}]\!] = [\![\mathbf{f}]\!] \Leftrightarrow \mathrm{KAC}^- \vdash e = f$. Since the regular expressions $\mathbf{e}$ and $\mathbf{f}$ can be easily computed from e and f, the problem of equivalence in KAC^- thus reduces to an equality of regular languages, which makes it decidable.

1.2 Relations with converse: theory KAC

We now move to the equational theory generated by relational models. It turns out that this theory will be characterised using "closed" languages on the extended alphabet $\mathbf{X}$. To define this closure operation, we first define a mirror operation $\overline{w}$ on words over $\mathbf{X}$, such that $\overline{\epsilon} \triangleq \epsilon$ and for any $x, w \in \mathbf{X} \times \mathbf{X}^\star$, $\overline{wx} = x'\overline{w}$. Accordingly with the axiom (5) of KAC we define a reduction relation $\rightsquigarrow$ on words over $\mathbf{X}$, using the following word rewriting rule.

$$w\overline{w}w \rightsquigarrow w \quad .$$

We call $w\overline{w}w$ a *pattern* of *root* w. The last two thirds of the pattern are $\overline{w}w$. Following [BÉS95, ÉB95], we extend this relation into a closure operation on languages.

Definition 5. The *closure* of a language $L \subseteq \mathbf{X}^\star$ is the smallest language containing L that is downward-closed with respect to $\rightsquigarrow$:

$$\mathcal{cl}(L) \triangleq \{v \mid \exists u \in L : u \rightsquigarrow^\star v\} \quad .$$

Example 6. If $X = \{a, b, c, d\}$, then $\mathbf{X} = \{a, b, c, d, a', b', c', d'\}$, and $\overline{ab'} = ba'$. We have the reduction $cab'ba'ab'd' \rightsquigarrow cab'd'$, by triggering a pattern of root ab'. For $L = \{aa'a, b, cab'ba'ab'd'\}$, we have $\mathcal{cl}(L) = L \cup \{a, cab'd'\}$.

Now we define a family of languages which play a prominent role in the sequel.

Definition 7. For any word $w \in \mathbf{X}^\star$, we define a regular language $\Gamma(w)$ by:

$$\Gamma(\epsilon) \triangleq \{\epsilon\}$$
$$\forall x \in \mathbf{X}, \forall w \in \mathbf{X}^\star, \quad \Gamma(wx) \triangleq (x'\Gamma(w)x)^\star \quad .$$

An equivalent operator called G is used in [BÉS95]: we actually have $\Gamma(w) = G(\overline{w})$, and our recursive definition directly corresponds to [BÉS95, Proposition 5.11.(2)]. By using such a simple recursive definition, we avoid the need for the notion of *admissible maps*, which is extensively used in [BÉS95].

Instead, we just have the following property to establish, which illustrates why these languages are of interest: words in $\Gamma(w)$ reduce into the last two thirds of a pattern compatible with w. Therefore, in the context of recognition by an automaton, $\Gamma(w)$ contains all the words that could potentially be skipped after reading w, in a closure automaton.

Proposition 8. *For all words u and v, $u \in \Gamma(v) \Leftrightarrow \exists t \in \mathit{suffixes}(v) : u \rightsquigarrow^\star \overline{t}t$.*

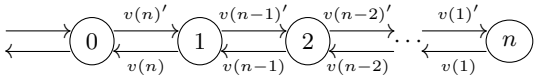


Fig. 1: Automaton $\mathscr{G}(v)$ recognising $\Gamma(v)$, with $|v| = n$.

Proof. The proof of the implication from left to right is routine but a bit lengthy,so that we put it in Appendix A.2.

For the converse implication, we first define the following language: $\Gamma'(v) \triangleq \{\bar{t}t \mid t \in \text{suffixes}(v)\}$. We thus have to show that the upward closure of $\Gamma'(v)$ is contained in $\Gamma(v)$. We first check that this language satisfies $\Gamma'(\epsilon) = \epsilon$ and $\Gamma'(vx) = \epsilon + x'\Gamma'(v)x$, which allows us to deduce that $\Gamma'(v) \subseteq \Gamma(v)$ by a straightforward induction.

It thus suffices to show that $\Gamma(v)$ is upward-closed with respect to $\rightsquigarrow$. For this, we introduce the family of automata $\mathscr{G}(v)$ depicted in Figure 1. One can check that $\mathscr{G}(v)$ recognises $\Gamma(v)$ by a simple induction on v. One can moreover notice that in this automaton, if $p \xrightarrow{x}_{\mathscr{G}(v)} q$, then $q \xrightarrow{x'}_{\mathscr{G}(v)} p$. More generally, for any word u, if $p \xrightarrow{u}_{\mathscr{G}(v)} q$, then $q \xrightarrow{\overline{u}}_{\mathscr{G}(v)} p$. So if $u_1 w u_2 \in \Gamma(v)$, then by definition of the automaton we have $0 \xrightarrow{u_1}_{\mathscr{G}(v)} q_1 \xrightarrow{w}_{\mathscr{G}(v)} q_2 \xrightarrow{u_2}_{\mathscr{G}(v)} 0$, and thus, by the previous remark:

$$0 \xrightarrow[\mathscr{G}(v)]{u_1} q_1 \xrightarrow[\mathscr{G}(v)]{w} q_2 \xrightarrow[\mathscr{G}(v)]{\overline{w}} q_1 \xrightarrow[\mathscr{G}(v)]{w} q_2 \xrightarrow[\mathscr{G}(v)]{u_2} 0 \quad ,$$

i.e., $u_1 w \overline{w} w u_2 \in \Gamma(v)$. In other words, for any words v and w and any $u \in \Gamma(v)$, if $w \rightsquigarrow u$ then w is also in $\Gamma(v)$, meaning exactly that $\Gamma(v)$ is upward-closed with respect to $\rightsquigarrow$.

Since $\Gamma'(v) \subseteq \Gamma(v)$, we deduce that $\Gamma(v)$ contains the upward closure of $\Gamma'(v)$, as expected. □

We now have enough material to embark in the proof of the implication (6) from the introduction, stating that if two expressions $e, f \in \text{Reg}_X^{\vee}$ are equal for all interpretations in all relational models, then $\mathcal{cl}(\mathbf{e}) = \mathcal{cl}(\mathbf{f})$.

Proof. Bloom et al. [BÉS95] consider specific relational interpretations: for any word $u \in \mathbf{X}^\star$ and for any letter $x \in \mathbf{X}$, they define

$$\phi_u(x) \triangleq \{(i-1, i) \mid u(i) = x\} \cup \{(i, i-1) \mid u(i) = x'\} \subseteq \{0, \dots, n\}^2 \quad ,$$

where $n \triangleq |u|$. The key property of those interpretations is the following:

$$(0, n) \in \widehat{\phi_u}(v) \Leftrightarrow v \rightsquigarrow^\star u \ . \tag{10}$$

We give a new proof of this property, by using the automaton $\Phi(u)$ depicted in Figure 2. By definition of $\Phi(u)$ and ϕ_u, we have that

$$(i, j) \in \phi_u(x) \Leftrightarrow i \xrightarrow{x}_{\Phi(u)} j \ .$$

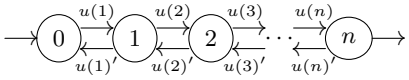


Fig. 2: Automaton $\Phi(u)$, with $|u| = n$.

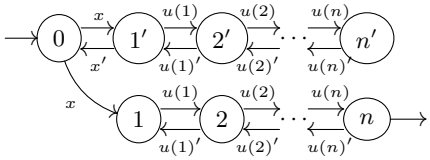


Fig. 3: Automaton $\Phi'(xu)$, with $|u| = n$, language equivalent to $\Phi(xu)$.

Therefore, proving (10) amounts to proving

$$v \in L(\Phi(u)) \Leftrightarrow v \rightsquigarrow^\star u \ . \tag{11}$$

First notice that $i \xrightarrow{x}_{\Phi(u)} j \Leftrightarrow j \xrightarrow{x'}_{\Phi(u)} i$. We can extend this to paths (as in the proof of Proposition 8) and then prove that if $s \rightsquigarrow t$ and $i \xrightarrow{t}_{\Phi(u)} j$ then $i \xrightarrow{s}_{\Phi(u)} j$. As u is clearly in $L(\Phi(u))$, any v such that $v \rightsquigarrow^\star u$ is also in $L(\Phi(u))$.

We proceed by induction on u for the other implication. The case $u = \epsilon$ being trivial, we consider $v \in L(\Phi(xu))$. We introduce a second automaton $\Phi'(xu)$ given in Figure 3, that recognises the same language as $\Phi(xu)$. The upper part of this automaton is actually the automaton $\mathscr{G}(\overline{xu})$ (as given in Figure 1), recognising the language $\Gamma(\overline{xu})$. Moreover, the lower part starting from state 1 is the automaton $\Phi(u)$. This allows us to obtain that $L(\Phi(xu)) = \Gamma(\overline{xu})xL(\Phi(u))$. Hence, for any $v \in L(\Phi(xu))$, there are $v_1 \in \Gamma(\overline{xu})$ and $v_2 \in L(\Phi(u))$ such that $v = v_1 x v_2$. By induction, we get $v_2 \rightsquigarrow^\star u$, and by Proposition 8 we know that $v_1 \rightsquigarrow^\star \overline{w}w$, with $w \in \text{suffixes}(\overline{xu})$. That means that $\overline{xu} = tw$, for some word t, so $xu = \overline{tw} = \overline{w}\ \overline{t}$. If we put everything back together:

$$v = v_1 x v_2 \rightsquigarrow^\star v_1 xu \rightsquigarrow^\star \overline{w}wxu = \overline{w}w\overline{w}\ \overline{t} \rightsquigarrow \overline{w}\ \overline{t} = xu \ .$$

This concludes the proof of (11), and thus (10).

We follow Bloom et al.'s proof [BÉS95] to deduce that the implication (6) from the introduction holds: we first prove that for all $e \in \text{Reg}_{\mathbf{X}}$, we have

$$\begin{aligned} u \in \mathcal{cl}(\llbracket e \rrbracket) &\Leftrightarrow \exists v \in \llbracket e \rrbracket, v \rightsquigarrow^\star u && \text{(by definition)} \\ &\Leftrightarrow \exists v \in \llbracket e \rrbracket, (0,n) \in \widehat{\phi_u}(v) && \text{(by (10))} \\ &\Leftrightarrow (0,n) \in \widehat{\phi_u}(e) \ . \end{aligned}$$

(For the last line, we use the fact that for any relational interpretation ϕ, we have $\widehat{\phi}(e) = \bigcup_{w \in [\![e]\!]} \widehat{\phi}(w)$.)

Furthermore, as $\phi_u(x') = \phi_u(x)^\vee$, we can prove that $\widehat{\phi_u}(e) = \widehat{\phi_u}(\mathbf{e})$ (see Appendix A.3). Therefore, for all expressions $e, f \in \mathrm{Reg}_X^\vee$ such that $e \equiv_{\mathrm{Rel}^\vee} f$, we have $\widehat{\phi_u}(\mathbf{e}) = \widehat{\phi_u}(e) = \widehat{\phi_u}(f) = \widehat{\phi_u}(\mathbf{f})$, and we deduce that $\mathcal{cl}([\![\mathbf{e}]\!]) = \mathcal{cl}([\![\mathbf{f}]\!])$ thanks to the above characterisation. □

2 Closure of an automaton

The problem here is the following: given two regular expressions $e, f \in \mathrm{Reg}_X^\vee$, how to decide $\mathcal{cl}([\![\mathbf{e}]\!]) = \mathcal{cl}([\![\mathbf{f}]\!])$? We follow the approach proposed by Bloom et al.: given an automaton recognising a language L, we show how to construct an automaton recognising $\mathcal{cl}(L)$. To solve the initial problem, it then suffices to build two automata recognising $[\![\mathbf{e}]\!]$ and $[\![\mathbf{f}]\!]$, to apply a construction to obtain two automata for $\mathcal{cl}([\![\mathbf{e}]\!])$ and $\mathcal{cl}([\![\mathbf{f}]\!])$, and to check those for language equivalence.

As a starting point, we first recall the construction proposed in [BÉS95].

2.1 The original construction

This construction uses the transition monoid of the input automaton:

Definition 9 (Transition monoid)**.** Let $\mathscr{A} = \langle Q, \Sigma, q_0, T, \delta \rangle$ be a deterministic automaton. Each word $u \in \Sigma^\star$ induces a function $u_\mathscr{A} : Q \to Q$ which associates to a state p the state q obtained by following the unique path from p labelled by u. The *transition monoid* of $\mathscr{A}$, written $M_\mathscr{A}$, is the set of functions $Q \to Q$ induced by words of $\Sigma^\star$, equipped with the composition of functions and the identity function.

This monoid is finite, and its subsets form a Kleene Algebra. Bloom et al. then proceed to define the closure automaton in the following way:

Theorem 10 (Closure automaton of [BÉS95])**.** *Let $L \subseteq \mathbf{X}^\star$ be a regular language, recognised by the deterministic automaton $\mathscr{A} = \langle Q, \mathbf{X}, q_0, Q_f, \delta \rangle$. Let $M_\mathscr{A}$ be the transition monoid of $\mathscr{A}$. Then the following deterministic automaton recognises $\mathcal{cl}(L)$:*

$$\mathscr{B} \triangleq \langle \mathcal{P}(M_\mathscr{A}) \times \mathcal{P}(M_\mathscr{A}), \mathbf{X}, (\{\epsilon_\mathscr{A}\}, \{\epsilon_\mathscr{A}\}), T, \delta_1 \rangle$$
$$\text{with } T \triangleq \{(F, G) \mid \exists u_\mathscr{A} \in F : u_\mathscr{A}(q_0) \in Q_f\},$$
$$\text{and } \delta_1((F, G), x) \triangleq \left(F \cdot \{x_\mathscr{A}\} \cdot \left((\{x'_\mathscr{A}\} \cdot G \cdot \{x_\mathscr{A}\})^\star\right),\ (\{x'_\mathscr{A}\} \cdot G \cdot \{x_\mathscr{A}\})^\star\right).$$

An important idea in this construction, that inspired our own, is the transition rule for the second component above. Let us write $\delta_2(G, x)$ for the expression $(\{x'_\mathscr{A}\} \cdot G \cdot \{x_\mathscr{A}\})^\star$, so that the definition of δ_1 can be reformulated as

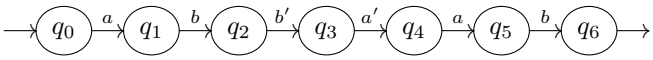
$$\delta_1((F, G), x) = (F \cdot \{x_\mathscr{A}\} \cdot \delta_2(G, x),\ \delta_2(G, x)).$$

With that in mind, one can see the second component as some kind of *history*, that runs on its own, and is used at each step to enrich the first component. At this point, it might be interesting to notice that the formula for $\delta_2(G, x)$ closely resembles the one for $\Gamma(wx) = (x'\Gamma(w)x)^\star$, which we defined in Section 1.2.

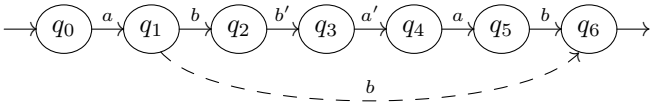
2.2 Intuitions

Let us forget the above construction, and try to build a closure automaton. One way would be to simply add transitions to the initial automaton. This idea comes naturally when one realises that if $u \rightsquigarrow^\star v$, then v is obtained by erasing some subwords from u: at each reduction step $u_1 w \overline{w} w u_2 \rightsquigarrow u_1 w u_2$ we just erase $\overline{w}w$. To "erase" such subwords using an automaton, it suffices to allow one to jump along certain paths.

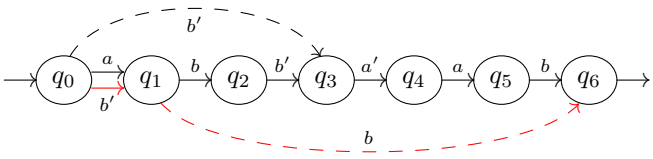
Suppose for instance that we start from the following automaton:



We can detect the pattern $ab\overline{ab}ab$, and allow one to "jump" over it when reading the last letter of the root of the pattern, in this case the b in second position. Our automaton thus becomes:



However, this approach is too naive, and it quickly leads to errors. If for instance we slightly modify the above example by adding a transition labelled by b' between q_0 and q_1, the same method leads to the following automaton, by detecting the patterns $b'bb'$ between q_0 and q_3 and $abb'a'ab$ between q_0 and q_6.



The problem is that the word $b'b$ is now wrongly recognised in the produced automaton. What happens here is that we can use the jump from q_1 to q_6, even though we didn't read the prerequisite for doing so, in this case the a constituting the beginning of the root ab of pattern $ab\overline{ab}ab$. (Note that the dual idea, consisting in enabling a jump when reading the first letter of the root of the pattern, would lead to similar problems.)

A way to prevent that, which was implicitly introduced in the original construction, consists in using a notion of *history*. The states of the closure automaton will be pairs of a state in the initial automaton and a history. That will allow us to distinguish between the state q_1 after reading a and the state q_1 after reading b', and to specify which jumps are possible considering what has been

previously read. In the construction given in [BÉS95], the history is given by an element of $\mathcal{P}(M_{\mathscr{A}})$, in the second component of the states (the "G" part). We will define a history as a set of words allowing for the same jumps, using $\Gamma(w)$.

2.3 Our construction

We have shown in Section 1.2 that $\forall u \in \Gamma(w), \exists v \in \text{suffixes}(w) : u \rightsquigarrow^\star \overline{v}v$, so we do have a characterisation of the words "allowing jumps" after having read some word w. The problem is that we want a finite number of possible histories, and there are infinitely many $\Gamma(w)$ (for instance, all the $\Gamma(a^n)$ are different). To get that, we will project $\Gamma(w)$ on the automaton. Let us consider a non-deterministic automaton $\mathscr{A} = \langle Q, \mathbf{X}, I, T, \Delta \rangle$ recognising a language L.

Definition 11. For any word $w \in \mathbf{X}^\star$ we define the relation $\gamma(w)$ between states of $\mathscr{A}$ by $\gamma(\epsilon) \triangleq \mathrm{Id}_Q$ and $\gamma(wx) = (\Delta(x') \circ \gamma(w) \circ \Delta(x))^\star$.

One can notice right away the strong relationship between γ and Γ:

Proposition 12. $\forall w, q_1, q_2,\ \ (q_1, q_2) \in \gamma(w) \ \Leftrightarrow\ \exists u \in \Gamma(w) : q_1 \xrightarrow{u}_{\mathscr{A}} q_2$.

This result is straightforward once one realises that $\gamma(w) = \widehat{\sigma}(\Gamma(w))$ with $\sigma(x) = \Delta(x)$. By composing Propositions 8 and 12 we eventually obtain that $((q_1, q_2) \in \gamma(w))$ iff $\exists u : q_1 \xrightarrow{u}_{\mathscr{A}} q_2$ and $u \rightsquigarrow^\star \overline{v}v$, with v a suffix of w.

The set Q being finite, γ has a finite index and one can define a finite set of histories as follows:

Definition 13. Let $\sim_\gamma$ be the kernel of γ: $u \sim_\gamma v$ iff $\gamma(u) = \gamma(v)$. We define the set G as the quotient of $\mathbf{X}^\star$ by $\sim_\gamma$. We denote by $[w]$ the elements of G, in such a way that $[u] = [v] \Leftrightarrow u \sim_\gamma v \Leftrightarrow \gamma(u) = \gamma(v)$.

We now have all the tools required for our construction of the closure of $\mathscr{A}$:

Theorem 14 (Closure Automaton)**.** *The closure of the language L is recognised by the automaton $\mathscr{A}' \triangleq \langle Q \times G, \mathbf{X}, I \times \{[\epsilon]\}, T \times G, \Delta' \rangle$ with:*

$$\Delta' = \{((q_1, [w]), x, (q_2, [wx])) \mid (q_1, q_2) \in \Delta(x) \circ \gamma(wx)\}.$$

We shall write L' for the language recognised by $\mathscr{A}'$. One can read the set of transitions as "from a state q_1 with an history w, perform a step x in the automaton $\mathscr{A}$, and then a jump compatible with wx, which becomes the new history". One can see, from the definition of Δ' and Proposition 12 that :

$$(q_1, [u]) \xrightarrow{x}_{\mathscr{A}'} (q_2, [ux]) \ \Leftrightarrow\ \exists (q_3, v) \in Q \times \Gamma(ux) : q_1 \xrightarrow{x}_{\mathscr{A}} q_3 \xrightarrow{v}_{\mathscr{A}} q_2. \quad (12)$$

Now we prove the correctness of this construction. First recall the notion of *simulation* [Mil89]:

Definition 15 (Simulation)**.** A relation R between the states of two automata $\mathscr{A}$ and $\mathscr{B}$ is a *simulation* if for all $(p, q) \in R$ we have (a) if $p \xrightarrow{x}_{\mathscr{A}} p'$, then there exists q' such that $q \xrightarrow{x}_{\mathscr{B}} q'$ and $(p', q') \in R$, and (b) if $p \in T_{\mathscr{A}}$ then $q \in T_{\mathscr{B}}$.

We say that $\mathscr{A}$ *is simulated by* $\mathscr{B}$ if there is a simulation R such that for any $p_0 \in I_{\mathscr{A}}$, there is $q_0 \in I_{\mathscr{B}}$ such that $p_0\ R\ q_0$.

The following property of γ is proved by exhibiting such a simulation:

Proposition 16. *For all words $u, v \in \mathbf{X}^\star$ such that $u \rightsquigarrow v$, we have $\gamma(u) \subseteq \gamma(v)$.*

Proof. First, notice that $\Gamma(u) \subseteq \Gamma(v) \Rightarrow \gamma(u) \subseteq \gamma(v)$, using Proposition 12. It thus suffices to prove $u \rightsquigarrow v \Rightarrow \Gamma(u) \subseteq \Gamma(v)$, which can be rewritten as $\Gamma(u_1 w \overline{w} w u_2) \subseteq \Gamma(u_1 w u_2)$. We can drop u_2 (it is clear that $\Gamma(w_1) \subseteq \Gamma(w_2) \Rightarrow \forall x \in \mathbf{X}, \Gamma(w_1 x) \subseteq \Gamma(w_2 x)$, from the definition of Γ): we now have to prove that $\Gamma(u_1 w \overline{w} w) \subseteq \Gamma(u_1 w)$. The proof of this inclusion relies on the fact that the automaton $\mathscr{G}(u_1 w \overline{w} w)$ is simulated by the automaton $\mathscr{G}(u_1 w)$, see Appendix A.4 for a formal definition of this simulation. □

We define an order relation $\preccurlyeq$ on the states of the produced automaton $(Q \times G)$, by $(p, [u]) \preccurlyeq (q, [v]) \triangleq p = q \wedge \gamma(u) \subseteq \gamma(v)$.

Proposition 17. *The relation $\preccurlyeq$ is a simulation for the automaton $\mathscr{A}'$.*

Proof. Suppose that $(p, [u]) \preccurlyeq (q, [v])$ and $(p, [u]) \xrightarrow{x}_{\mathscr{A}'} (p', [ux])$, i.e., $(p, p') \in \Delta_x \circ \gamma(ux)$. We have $p = q$ and $\gamma(u) \subseteq \gamma(v)$, hence $\gamma(ux) \subseteq \gamma(vx)$, and thus $(p, p') \in \Delta_x \circ \gamma(vx)$ meaning that $(p, [v]) \xrightarrow{x}_{\mathscr{A}'} (p', [vx])$. It remains to check that $(p', [ux]) \preccurlyeq (p', [vx])$, i.e., $\gamma(ux) \subseteq \gamma(vx)$, which we just proved. □

We may now prove that $L' = \mathcal{cl}(L)$.

Lemma 18. $L' \subseteq \mathcal{cl}(L)$

Proof. We prove by induction on u that for all q_0, q such that $(q_0, [\epsilon]) \xrightarrow{u}_{\mathscr{A}'} (q, [u])$, there exists v such that $v \rightsquigarrow^\star u$ and $q_0 \xrightarrow{v}_{\mathscr{A}} q$. The case $u = \epsilon$ is trivial.

If $(q_0, [\epsilon]) \xrightarrow{u}_{\mathscr{A}'} (q_1, [u]) \xrightarrow{x}_{\mathscr{A}'} (q, [ux])$, by induction one can find v_1 such that $q_0 \xrightarrow{v_1}_{\mathscr{A}} q_1$ and $v_1 \rightsquigarrow^\star u$. We also know (by (12) and Proposition 8) that there are some q_2, v_2 and $v_3 \in \mathrm{suffixes}(ux)$ such that $q_1 \xrightarrow{x}_{\mathscr{A}} q_2$, $v_2 \rightsquigarrow^\star \overline{v}_3 v_3$ and $q_2 \xrightarrow{v_2}_{\mathscr{A}} q$. We thus get

$$q_0 \xrightarrow{v_1}_{\mathscr{A}} q_1 \xrightarrow{x}_{\mathscr{A}} q_2 \xrightarrow{v_2}_{\mathscr{A}} q \text{ and } v_1 x v_2 \rightsquigarrow^\star u x v_2 \rightsquigarrow^\star u x \overline{v}_3 v_3 \rightsquigarrow u x.$$

By choosing $q \in T$, we obtain the desired result. □

Lemma 19. $L \subseteq L'$

Proof. This is actually very simple. First notice that for all u, $\gamma(u)$ is a reflexive relation, hence $q_1 \xrightarrow{x}_{\mathscr{A}} q_2$ entails $\forall u, (q_1, [u]) \xrightarrow{x}_{\mathscr{A}'} (q_2, [ux])$. This means that the relation R defined by $p \; R \; (q, [w]) \Leftrightarrow p = q$ is a simulation between $\mathscr{A}$ and $\mathscr{A}'$, and thus $L = L(\mathscr{A}) \subseteq L(\mathscr{A}') = L'$. □

Lemma 20. *L' is downward-closed for $\rightsquigarrow$.*

A technical lemma is required to establish this closure property:

Lemma 21. *If $(q_1, [uw]) \xrightarrow{x}_{\mathscr{A}'} (q_2, [uwx]) \xrightarrow{\overline{wx}\; wx}_{\mathscr{A}'} (q_3, [uwx\ \overline{wx}\ wx])$, then $(q_1, [uw]) \xrightarrow{x}_{\mathscr{A}'} (q_3, [uwx])$.*

Proof sketch. The proof being quite verbose and dry, we shall only give a sketch of it here, referring to Appendix A.5 for a detailed one. If $|w| = n$ and $|u| = m$, the premise can be equivalently stated:

$$(q_1, [(uw)|_{m+n-1}]) \xrightarrow{w(n)}_{\mathscr{A}'} (q_2, [uw]) \xrightarrow{\overline{w}w}_{\mathscr{A}'} (q_3, [uw\overline{w}w]).$$

(Recall that $u|_i$ denotes the prefix of length i of a word u.) Let us write $\Gamma_i = \Gamma((uw\overline{w}w)|_{m+n+i}) = \Gamma(uw(\overline{w}w)|_i)$ and $x_i = (uw\overline{w}w)(n+m+i)$ for $0 \leqslant i \leqslant 2n$. By Proposition 12 and the definition of $\mathscr{A}'$, we can show that there are $v_i \in \Gamma_i$ such that the execution above can be lifted into an execution in $\mathscr{A}$:

$$q_1 \xrightarrow{x_0 v_0 x_1 v_1 \cdots x_i v_i \cdots x_{2n} v_{2n}}_{\mathscr{A}} q_3.$$

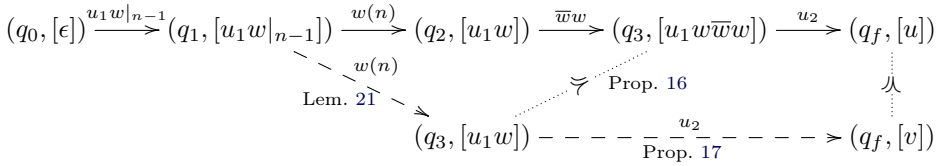
Then one can prove by recurrence on i and using Proposition 8 that:

$$\forall i, \exists t_i \in \Gamma(uw) : (\overline{w}w)|_i v_i \rightsquigarrow^\star t_i(\overline{w}w)|_i. \tag{13}$$

We deduce that $v_0 x_1 v_1 \cdots x_i v_i \cdots x_{2n} v_{2n} \rightsquigarrow^\star t_0 t_1 \cdots t_{2n} \overline{w}w \in \Gamma(uw)^{2n+2} \subseteq \Gamma(uw)$. By Proposition 8, this means that $v_0 x_1 v_1 \cdots x_i v_i \cdots x_{2n} v_{2n}$ is in $\Gamma(uw)$, so that $(q_1, q_3) \in \Delta(w(n)) \circ \gamma(uw)$, and $(q_1, [uw|_{n-1}]) \xrightarrow{w(n)}_{\mathscr{A}'} (q_2, [uw])$. □

With this intermediate lemma, one can obtain a succinct proof of Lemma 20:

Proof. The statement of the lemma is equivalent to saying that if $u \rightsquigarrow v$ with $u \in L'$ then v is also in L'. Consider $u = u_1 w \cdot \overline{w} \cdot w u_2$ and $v = u_1 w u_2$ with $|w| = n \geqslant 1$ (the case where $w = \epsilon$ doesn't hold any interest since it implies that $u = v$). By combining Lemma 21 and Proposition 17 we can build the following diagram:



□

Lemmas 19 and 20 tell us that L' is closed and contains L, so by definition of the closure of a language, we get $\mathcal{cl}(L) \subseteq L'$. Lemma 18 gives us the other inclusion, thus proving Theorem 14.

3 Analysis and consequences

3.1 Relationship with [BÉS95]'s construction

As suggested by an anonymous referee, one can also formally relate our construction to the one from [BÉS95]: we give below an explicit and rather natural

bisimulation relation between the automata produced by both these methods. This results in an alternative correctness proof of our construction, by reducing it to the correctness of the one from [BÉS95].

We first make the two constructions comparable: the original construction, because it considers the transition monoid, takes as input a deterministic automaton. It returns a deterministic automaton. Instead, our construction does not require determinism in its input, but produces a non-deterministic automaton. We thus have to ask of both methods to accept as their input a *non-deterministic* automaton, and to return a *deterministic* automaton.

For our construction, the straightforward thing to do would be to determinise the automaton afterwards. We can actually do better, by noticing that from a state $(p, [u])$, reading some x, there may be a lot of accessible states, but all of their histories (second components) will be equal to $[ux]$. So in order to get a deterministic automaton, one only has to perform the power-set construction on the first component of the automaton. This way, we get an automaton $\mathscr{A}_1$ with states in $\mathcal{P}(Q) \times G$ and a transition function

$$\delta_1((P, [u]), x) = (P \cdot (\Delta(x) \circ \gamma(ux)), [ux]) .$$

The original construction can also be adjusted very easily: first build a deterministic automaton $\mathcal{D}$ with the usual powerset construction, then apply the construction as described in Theorem 10 to get an automaton which we call $\mathscr{A}_2$. An important thing here is to understand the shape of the resulting transition monoid $M_{\mathcal{D}}$: its elements are functions over sets of states (because of the power-set construction) induced by words; more precisely, they are sup-semilattice homomorphisms, and they are in bijection with binary relations on states.

Define the following KA-homomorphism from $\mathcal{P}(M_{\mathcal{D}})$ to $\mathcal{P}(Q^2)$:

$$i(F) = \{(p, q) \mid \exists u_{\mathcal{D}} \in F : q \in u_{\mathcal{D}}(\{p\})\} .$$

(That i is a KA-homomorphism comes from the fact that the elements of $M_{\mathcal{D}}$ are themselves sup-semilattice homomorphisms on $\mathcal{P}(Q)$.) We can check that for all $x \in \mathbf{X}$, we have

$$\begin{aligned} i(\{x_{\mathcal{D}}\}) &= \{(p, q) \mid q \in x_{\mathcal{D}}(\{p\})\} = \{(p, q) \mid q \in \delta(\{p\}, x)\} \\ &= \left\{(p, q) \;\middle|\; p \xrightarrow{x}_{\mathscr{A}} q\right\} = \Delta(x) , \end{aligned}$$

It follows that the following relation is a bisimulation between $\mathscr{A}_1$ and $\mathscr{A}_2$.

$$\{((Q, [u]),\ (F, G)) \mid Q = I \cdot i(F) \text{ and } \gamma(u) = i(G)\}$$

In Appendix A.6 we give a detailed proof of this.

3.2 Complexity

Because we are speaking about algorithms rather than actual programs, it is a bit difficult to give accurate complexity bounds, considering the many possible

data structures appearing during the computation. However, one may think that a relevant complexity measure of the final algorithm (for deciding equality in KAC) could be the size of the produced automata. In the following the *size* of an automaton is its number of states. In order to give a fair comparison, we will consider the generic algorithms given in the previous subsection, taking as their input a *non-deterministic* automaton, and returning a *deterministic* automaton.

Let us begin by evaluating the size of the automaton produced by the method in [BÉS95], given a non-deterministic automaton of size n. As explained above, the states of the constructed transition monoid ($M_{\mathcal{D}}$) are in bijection with the binary relations on Q. There are thus at most 2^{n^2} elements in this monoid. We deduce that the final automaton, whose states are pairs of subsets of $M_{\mathcal{D}}$ has at most $2^{2^{n^2}} \times 2^{2^{n^2}} = 2^{2^{n^2+1}}$ states.

Now with the deterministic version of our construction, the states are in the set $\mathcal{P}(Q) \times G$. Since G is the set of equivalence classes of $\sim_\gamma$ and γ has values in the reflexive binary relations over Q, we know that $\sim_\gamma$ has less than $2^{n\times(n-1)}$ elements. Hence we can see that $|\mathcal{P}(Q) \times G| \leqslant 2^n \times 2^{n\times(n-1)} = 2^{n^2}$, which is significantly smaller than the $2^{2^{n^2+1}}$ states we get with the other construction.

3.3 A polynomial-space algorithm

The above upper-bound on the number of states of the automata produced by our construction allows us to show that the problem of equivalence in KAC is in PSpace (the problem was already known to be PSpace-hard since KAC is conservative over KA, which is PSpace-complete [MS73]).

Recall that the equivalence of two deterministic automata $\mathscr{A}$ and $\mathscr{B}$ is in LogSpace. The algorithm to show that relies on the fact that $\mathscr{A}$ and $\mathscr{B}$ are different if and only if there is a word w in the difference of $L(\mathscr{A})$ and $L(\mathscr{B})$ such that $|w| \leqslant |\mathscr{A}| \times |\mathscr{B}|$. With that in mind, we can give a non-deterministic algorithm, by simulating a computation in both automata with a letter chosen non-deterministically at each step, with a counter to stop us at size $|\mathscr{A}| \times |\mathscr{B}|$. The resulting algorithm will only have to store the counter of size $\log(|\mathscr{A}| \times |\mathscr{B}|)$ and the two current states.

For our problem, the first step is to compute $\mathbf{e}$ and $\mathbf{f}$ from the regular expressions with converse e and f. It is obvious that such a transformation can be done in linear time and space, by a single sweep of both e and f. Then we have to build automata for $\mathbf{e}$ and $\mathbf{f}$. Once again this is a very light operation: if one considers for instance the position automaton (also called Glushkov's construction [Glu61]), we obtain automata of respective sizes $n = |\mathbf{e}| + 1 = |e| + 1$ and $m = |\mathbf{f}| + 1 = |f| + 1$, where $|\cdot|$ denotes the number of variable leaves of a regular expression (possibly with converse).

Our construction then produces closed automata of size at most 2^{n^2} and 2^{m^2}, so that the non-deterministic algorithm to check their equivalence needs to scan all words of size smaller than by $2^{n^2} \times 2^{m^2} = 2^{n^2+m^2}$. The counter used to bound the recursion depth can thus be stored in polynomial space (n^2+m^2). It is worth

input : Two regular expressions with converse $e, f \in \mathrm{Reg}^{\vee}_X$
output: A Boolean, saying whether or not $\mathrm{KAC} \vdash e = f$.

$\mathscr{A}_1 = \langle Q_1, \mathbf{X}, I_1, T_1, \Delta_1 \rangle \leftarrow$ Glushkov' automaton recognising $[\![\mathbf{e}]\!]$;
$\mathscr{A}_2 = \langle Q_2, \mathbf{X}, I_2, T_2, \Delta_2 \rangle \leftarrow$ Glushkov' automaton recognising $[\![\mathbf{f}]\!]$;
$N \leftarrow (2^{(|e|+1)^2} \times 2^{(|f|+1)^2})$; /* N gets a value $\geqslant |\mathcal{L}(\mathscr{A}_1)| \cdot |\mathcal{L}(\mathscr{A}_2)|$ */
$((P_1, R_1), (P_2, R_2)) \leftarrow ((I_1, \mathrm{Id}_{Q_1}), (I_2, \mathrm{Id}_{Q_1}))$;
while $N > 0$ **do**
  $N \leftarrow N - 1$; /* N bounds the recursion depth */
  $f_1 \leftarrow$ is_empty$(P_1 \cap T_1)$;
  $f_2 \leftarrow$ is_empty$(P_2 \cap T_2)$;
  if $f_1 = f_2$ **then**
    $x \leftarrow$random(**X**); /* Non-deterministic choice */
    $(R_1, R_2) \leftarrow ((\Delta_1(x') \circ R_1 \circ \Delta_1(x))^\star, (\Delta_2(x') \circ R_2 \circ \Delta_2(x))^\star)$;
    $(P_1, P_2) \leftarrow (P_1 \cdot (\Delta_1(x) \circ R_1), P_2 \cdot (\Delta_2(x) \circ R_2))$;
  else
    return false; /* A difference appeared for some word, $e \neq f$ */
  end
end
return true; /* There was no difference, KAC $\vdash e = f$ */

Algorithm 1: A PSpace algorithm for KAC

mentionning here that with the automata constructed in [BÉS95], the counter would have size $2^{n^2+1} + 2^{m^2+1}$ which is not a polynomial.

Now the last two important things to worry about are the representation of the states of the closure automata, in particular their "history" component, and the way to compute their transition function. Let us focus on the automaton for e and let Q be the set of states of the Glushkov automaton built out of it.

- For the state representation, one needs to represent an equivalence class $[u] \in G$ by its image under γ: while the smallest word $w \in [u]$ may be quite long, $\gamma(u)$ is just a binary relation on Q. We shall thus represent the states in the determinised closure automaton as pairs of a set of states in Q and a binary relation (set of pairs) over Q. Such a pair can be stored in polynomial space (recall that $|Q| = n = |e| + 1$).
- For computing the transition function, the image of a pair $(\{q_1, \cdots, q_k\}, R)$ (with $R \subseteq Q^2$) by a letter $x \in \mathbf{X}$ is done in two steps: first the relation becomes $R' = (\Delta(x') \circ R \circ \Delta(x))^\star$, then the set of states becomes $\{q \mid \exists i, 1 \leqslant i \leqslant k : (q_i, q) \in \Delta(x) \circ R'\}$. Those computations take place in PSpace. (The composition of two relations in Q^2 can be performed in space $\mathcal{O}\left(|Q|^2\right)$, and the same holds for the reflexive and transitive closure of a relation R by building the powers $(R + \mathrm{Id}_Q)^{2^k}$ and keeping a copy of the previous iteration to stop when the fixed-point is reached.)

Summing up, we obtain Algorithm 1, which is PSpace.

Conclusion

Starting from the works of Bernátsky, Bloom, Ésik and Stefanescu, we gave a new and more efficient algorithm to decide the theory KAC. This algorithm relies on a new construction for the closure of an automaton, which allowed us to show that the problem was in fact in the complexity class PSpace.

To prove the correctness of our construction, we used the family of regular languages $\Gamma(w)$ ($G(w^\vee)$ in [BÉS95]), and we establish its main properties using a proper finite automata characterisation. Moreover, this function allowed us to reformulate the proof of the completeness of the reduction from equality in $\mathrm{Rel}^\vee$ to equivalence of closed automata (implication (6) from the introduction).

As an exercise, we have implemented and tested the various constructions and algorithms in an OCaml program which is available online[1].

To continue this work, we would like to implement our algorithm in the proof assistant Coq, as a tactic to automatically prove the equalities in KAC—as it has already been done for the theories KA and KAT. The simplifications we propose in this paper give us hope that such a task is feasible. The main difficulty certainly lies in the formalisation of the completeness proof of KAC (implication (7) from the introduction): the proof given in [ÉB95] uses yet another automaton construction for the closure, which is much more complicated than the one used in [BÉS95], and which seems quite difficult to formalise in Coq. We hope to find an alternative completeness proof, by exploiting the simplicity of the presented construction.

Acknowledgements. We are grateful to the anonymous referees who suggested us the alternative proof of correctness which we provide in Section 3.1, and who helped us to improve this paper.

References

[BÉS95] Bloom, S. L., Ésik, Z., Stefanescu, G.: Notes on equational theories of relations. Algebra Universalis 33, 98–126 (1995)

[Bof90] Boffa, M.: Une remarque sur les systèmes complets d'identités rationnelles. Informatique Théorique et Applications 24, 419–428 (1990)

[Bof95] Boffa, M.: Une condition impliquant toutes les identités rationnelles. Informatique Théorique et Applications 29, 515–518 (1995)

[Con71] Conway, J. H.: Regular algebra and finite machines. Chapman and Hall Mathematics Series (1971)

[ÉB95] Ésik, Z., Bernátsky, L.: Equational properties of Kleene algebras of relations with conversion. Theoretical Computer Science 137, 237–251 (1995)

[Glu61] Glushkov, V. M.: The abstract theory of automata. Russian Mathematical Surveys 16, 1 (1961)

[Kle51] Kleene, S. C.: Representation of Events in Nerve Nets and Finite Automata. Memorandum. Rand Corporation (1951)

[1] http://perso.ens-lyon.fr/paul.brunet/cka.html

[Koz91] Kozen, D.: A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. In: LICS, pp. 214–225. IEEE Computer Society (1991)
[Kro90] Krob, D.: A Complete System of B-Rational Identities. In: ICALP, Lecture Notes in Computer Science, vol. 443, pp. 60–73. Springer (1990)
[MS73] Meyer, A., Stockmeyer., L. J.: Word problems requiring exponential time. In: Proc. ACM symposium on Theory of computing, pp. 1–9. ACM (1973)
[Mil89] Milner, R.: Communication and Concurrency. Prentice Hall (1989)
[Red64] Redko, V. N.: On defining relations for the algebra of regular events. Ukrainskii Matematicheskii Zhurnal pp. 120–126 (1964)
[Sal66] Salomaa, A.: Two Complete Axiom Systems for the Algebra of Regular Events. J. ACM 13, 158–169 (1966)

A Omitted proofs

A.1 Proof of Equation (9)

We will show here that $\widehat{\eta}(e) = \widehat{\eta}(f)$ implies that $[\![e]\!] = [\![f]\!]$, for e and f regular expressions over $\mathbf{X}$.

It is well known that for any expression $e \in \mathrm{Reg}_{\mathbf{X}}$, for any $\sigma : \mathbf{X} \longrightarrow \mathcal{P}(\Sigma^\star)$,

$$\widehat{\sigma}(e) = \bigcup_{w \in [\![e]\!]} \widehat{\sigma}(w).$$

Consider the following partial function: $i : \begin{array}{rcl} X_\bullet^\star & \longrightarrow & \mathbf{X}^\star \\ \epsilon & \longmapsto & \epsilon \\ x \bullet w & \longmapsto & x \cdot i(w) \\ \bullet x w & \longmapsto & x' \cdot i(w). \end{array}$

We will write $\widehat{i}$ the function $[W \mapsto \{i(w) \mid w \in W\}]$. We will show by induction on $w \in \mathbf{X}^\star$ that $\widehat{i} \circ \widehat{\eta}(w) = \{w\}$:

- $\widehat{i} \circ \widehat{\eta}(\epsilon) = \widehat{i}(\{\epsilon\}) = \{\epsilon\}$;
- if $x \in X$, then

$$\begin{aligned} \widehat{i} \circ \widehat{\eta}(xw) &= \widehat{i}(\eta(x) \cdot \widehat{\eta}(w)) && (\widehat{\eta} \text{ is a morphism}) \\ &= \widehat{i}(\{x\bullet\} \cdot \widehat{\eta}(w)) && (\text{definition of } \eta) \\ &= \{x\} \cdot (\widehat{i} \circ \widehat{\eta}(w)) && (\text{definition of } i) \\ &= \{xw\}; && (\text{induction hypothesis}) \end{aligned}$$

- and similarly if $x' \in X'$, then $\widehat{i} \circ \widehat{\eta}(x'w) = \widehat{i}(\{\bullet x\} \cdot \widehat{\eta}(w)) = \{x'\} \cdot (\widehat{i} \circ \widehat{\eta}(w)) = \{x'w\}$.

Thus, we get that:

$$[\![e]\!] = \bigcup_{w \in [\![e]\!]} \{w\} = \bigcup_{w \in [\![e]\!]} \widehat{i} \circ \widehat{\eta}(w) = \widehat{i}\left(\bigcup_{w \in [\![e]\!]} \widehat{\eta}(w) \right) = \widehat{i}(\widehat{\eta}(e)).$$

Thus we get $[\![e]\!] = \widehat{i}(\widehat{\eta}(e)) = \widehat{i}(\widehat{\eta}(f)) = [\![f]\!]$.

A.2 Proof of Proposition 8

Let us prove the first implication of Proposition 8:

$$\forall \mathbf{w} \in \mathbf{X}^\star, \forall u \in \Gamma(\mathbf{w}), \exists v \in \mathrm{suffixes}(\mathbf{w}) : u \rightsquigarrow^\star \overline{v} v.$$

We will proceed by induction on $\mathbf{w}$:

1. If $\mathbf{w} = \epsilon$, then $u \in \Gamma(\epsilon) = \{\epsilon\}$. So $u = \epsilon \rightsquigarrow^0 \overline{\epsilon}\epsilon$ and obviously $\epsilon \in \mathrm{suffixes}(\epsilon)$.

2. Otherwise $\mathbf{w} = wx$, and $u \in \Gamma(wx) = (x'\Gamma(w)x)^\star$. Thus we know that for some $n \in \mathbb{N}$, $u \in (x'\Gamma(w)x)^n$. We now will prove by recurrence on n that $u \in (x'\Gamma(w)x)^n \Rightarrow \exists v \in \text{suffixes}(wx) : u \rightsquigarrow^\star \overline{v}v$:
   (a) If $n = 0$ then $u = \epsilon \rightsquigarrow^0 \overline{\epsilon}\epsilon$ and $\epsilon \in \text{suffixes}(wx)$.
   (b) If $n = m + 1$ then we can introduce $u_1 \in \Gamma(w)$ and $u_2 \in (x'\Gamma(w)x)^m$ such that $u = x'u_1xu_2$.
      i. By induction hypothesis, $\exists v_1 \in \text{suffixes}(w)$ such that $u_1 \rightsquigarrow^\star \overline{v_1}v_1$.
      ii. By reccurence hypothesis, $\exists v_2 \in \text{suffixes}(wx)$ such that $u_2 \rightsquigarrow^\star \overline{v_2}v_2$.

   Thus we know that $u = x'u_1xu_2 \rightsquigarrow^\star x'\overline{v_1}v_1x\overline{v_2}v_2$. We will now do a case analysis on the length of v_2.
      i. If $|v_2| = 0$, then $v_2 = \epsilon$ so $u \rightsquigarrow^\star x'\overline{v_1}v_1x = \overline{v_1x}v_1x$.
      ii. If $|v_2| > 0$, as $v_2 \in \text{suffixes}(wx)$, we can write $v_2 = v_3x$ with $v_3 \in \text{suffixes}(w)$. We will now compare the sizes of v_1 and v_3, both being suffixes of w.
         A. If $|v_1| \leqslant |v_3|$, then $v_3 = v_4v_1$. Thus we have:

$$\begin{aligned} u \rightsquigarrow^\star x'\overline{v_1}v_1x\overline{v_3x}v_3x &= x'\overline{v_1}v_1xx'\overline{v_1}\ \overline{v_4}v_4v_1x \\ &= \overline{v_1x}v_1x\overline{v_1x}\ \overline{v_4}v_4v_1x \\ &\rightsquigarrow \overline{v_1x}\ \overline{v_4}v_4v_1x = \overline{v_2}v_2 \end{aligned}$$

         B. Otherwise we can write $v_1 = v_5v_3$ and thus:

$$\begin{aligned} u &\rightsquigarrow^\star x'\overline{v_5v_3}v_5v_3x\overline{v_3x}v_3x \\ &\rightsquigarrow \overline{v_3v_5x}v_5v_3x = \overline{v_1x}v_1x \end{aligned}$$

   So we have shown that either $u \rightsquigarrow^\star \overline{v_1x}v_1x$ or $u \rightsquigarrow^\star \overline{v_2}v_2$, and as we know that both v_1x and v_2 are suffixes of wx, we have finished.

A.3 Proof of $\widehat{\phi}_u(e) = \widehat{\phi}_u(\mathbf{e})$

We first give an alternative definition of $\mathbf{e}$: let χ and ξ be the following mutually recursive functions:

$\chi(\mathbb{0}) \triangleq \mathbb{0}$	$\xi(\mathbb{0}) \triangleq \mathbb{0}$
$\chi(\mathbb{1}) \triangleq \mathbb{1}$	$\xi(\mathbb{1}) \triangleq \mathbb{1}$
$\chi(x) \triangleq x$	$\xi(x) \triangleq x'$
$\chi(e + f) \triangleq \chi(e) + \chi(f)$	$\xi(e + f) \triangleq \xi(e) + \xi(f)$
$\chi(e \cdot f) \triangleq \chi(e) \cdot \chi(f)$	$\xi(e \cdot f) \triangleq \xi(f) \cdot \xi(e)$
$\chi(e^\star) \triangleq (\chi(e))^\star$	$\xi(e^\star) \triangleq (\xi(e))^\star$
$\chi(e^\vee) \triangleq \xi(e)$	$\xi(e^\vee) \triangleq \chi(e)$

χ and ξ are both functions mapping an expression in $\text{Reg}_X^\vee$ to an expression in $\text{Reg}_{\mathbf{X}}$. It is quite immediate that $\mathbf{e} = \nu(\tau(e)) = \chi(e)$.

Hence, what we want is to prove that $\widehat{\phi}_u(e) = \widehat{\phi}_u(\chi(e))$. Because of the mutually reccursive definition we gave, we will prove inductively on $e \in \text{Reg}_X^\vee$ the following:

$$\widehat{\phi}_u(\chi(e)) = \widehat{\phi}_u(e) \quad \wedge \quad \widehat{\phi}_u(\xi(e)) = \widehat{\phi}_u(e)^\vee$$

- $\chi(\mathbb{0}) = \mathbb{0}$ and $\widehat{\phi}_u(\xi(\mathbb{0})) = \widehat{\phi}_u(\mathbb{0}) = \mathbb{0} = \mathbb{0}^\vee = \widehat{\phi}_u(\mathbb{0})^\vee$, so this case and the case $\mathbb{1}$ don't hold any difficulty.
- $\widehat{\phi}_u(\chi(x)) = \widehat{\phi}_u(x)$, so no problem there, but $\widehat{\phi}_u(\xi(x)) = \widehat{\phi}_u(x') = \phi_u(x')$. By the definition of ϕ_u we get:

$$\begin{aligned}\phi_u(x') &= \{(i-1,i) \mid u(i) = x'\} \cup \{(i,i-1) \mid u(i) = x\} \\ &= \{(i,i-1) \mid u(i) = x'\}^\vee \cup \{(i-1,i) \mid u(i) = x\}^\vee \\ &= (\{(i,i-1) \mid u(i) = x'\} \cup \{(i-1,i) \mid u(i) = x\})^\vee \\ &= \phi_u(x)^\vee\end{aligned}$$

Every other case is then quite simple:

- $e + f$: $\begin{aligned}\widehat{\phi}_u(\chi(e+f)) &= \widehat{\phi}_u(\chi(e) + \chi(f)) = \widehat{\phi}_u(\chi(e)) \cup \widehat{\phi}_u(\chi(f)) \\ &= \widehat{\phi}_u(e) \cup \widehat{\phi}_u(f) = \widehat{\phi}_u(e+f)\end{aligned}$
  $\begin{aligned}\widehat{\phi}_u(\xi(e+f)) &= \widehat{\phi}_u(\xi(e) + \xi(f)) = \widehat{\phi}_u(\xi(e)) \cup \widehat{\phi}_u(\xi(f)) \\ &= \widehat{\phi}_u(e)^\vee \cup \widehat{\phi}_u(f)^\vee = \left(\widehat{\phi}_u(e) \cup \widehat{\phi}_u(f)\right)^\vee \\ &= \left(\widehat{\phi}_u(e+f)\right)^\vee\end{aligned}$
- $e \cdot f$: $\begin{aligned}\widehat{\phi}_u(\chi(e \cdot f)) &= \widehat{\phi}_u(\chi(e) \cdot \chi(f)) = \widehat{\phi}_u(\chi(e)) \circ \widehat{\phi}_u(\chi(f)) \\ &= \widehat{\phi}_u(e) \circ \widehat{\phi}_u(f) = \widehat{\phi}_u(e \cdot f)\end{aligned}$
  $\begin{aligned}\widehat{\phi}_u(\xi(e \cdot f)) &= \widehat{\phi}_u(\xi(f) \cdot \xi(e)) = \widehat{\phi}_u(\xi(f)) \circ \widehat{\phi}_u(\xi(e)) \\ &= \widehat{\phi}_u(f)^\vee \circ \widehat{\phi}_u(e)^\vee = \left(\widehat{\phi}_u(e) \circ \widehat{\phi}_u(f)\right)^\vee \\ &= \left(\widehat{\phi}_u(e \cdot f)\right)^\vee\end{aligned}$
- $e^\star$: $\widehat{\phi}_u(\chi(e^\star)) = \widehat{\phi}_u(\chi(e)^\star) = \left(\widehat{\phi}_u(\chi(e))\right)^\star = \left(\widehat{\phi}_u(e)\right)^\star = \widehat{\phi}_u(e^\star)$
  $\begin{aligned}\widehat{\phi}_u(\xi(e^\star)) &= \widehat{\phi}_u(\xi(e)^\star) = \left(\widehat{\phi}_u(\xi(e))\right)^\star = \left(\widehat{\phi}_u(e)^\vee\right)^\star \\ &= \left(\widehat{\phi}_u(e)^\star\right)^\vee = \widehat{\phi}_u(e^\star)^\vee\end{aligned}$
- $e^\vee$: $\widehat{\phi}_u(\chi(e^\vee)) = \widehat{\phi}_u(\xi(e)) = \widehat{\phi}_u(e)^\vee = \widehat{\phi}_u(e^\vee)$
  $\widehat{\phi}_u(\xi(e^\vee)) = \widehat{\phi}_u(\chi(e)) = \widehat{\phi}_u(e) = \widehat{\phi}_u(e)^{\vee\vee} = \widehat{\phi}_u(e^\vee)^\vee$

A.4 Proof of $\Gamma(uw\overline{w}w) \subseteq \Gamma(uw)$

We will prove in this section that for any $u, w \in \mathbf{X}^*$, $\Gamma(uw\overline{w}w) \subseteq \Gamma(uw)$. First recall that for any word w, the language $\Gamma(w)$ is recognised by the automaton given in Figure 1). With that in mind, we give in Figure 4 an abstract view of the automata recognising $\Gamma(uw\overline{w}w)$ and $\Gamma(uw)$ defined as before. With the notations of this figure, now define a relation $\leqslant$ as follows (this relation is also represented in dashed lines in Figure 4):

$$\begin{aligned} a_i &\leqslant b_i && \text{for all } i \leq n+m \ , \\ a_{n+m+i} &\leqslant b_{n+m-i} && \text{for all } i \leq n \ , \\ a_{2n+m+i} &\leqslant b_{m+i} && \text{for all } i \leq n \ ; \end{aligned}$$

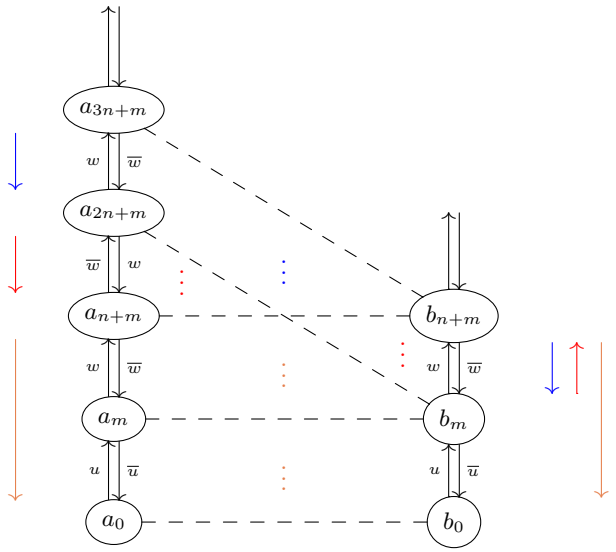


Fig. 4: Automata $\mathscr{G}(uw\overline{w}w)$ and $\mathscr{G}(uw)$, with $|u| = m$ and $|w| = n$

One easily checks that this relation is a simulation, thus establishing in particular that the language recognised by the left-hand side automaton (for $\Gamma(uw\overline{w}w)$) is contained in that of the right-hand side (for $\Gamma(uw)$).

A.5 Proof of result (13)

Recall that $n = |w|$ and $m = |u|$, and that for any $0 \leqslant i \leqslant 2n$, we have:

- $\Gamma_i = \Gamma\left((uw\overline{w}w)|_{m+n+i}\right) = \Gamma\left(uw(\overline{w}w)|_i\right)$
- $v_i \in \Gamma_i$.

We will give here a proof that

$$\forall 0 \leqslant i \leqslant 2n, \exists t_i \in \Gamma(uw) : (\overline{w}w)|_i v_i \rightsquigarrow^\star t_i(\overline{w}w)|_i.$$

As v_i is in $\Gamma\left(uw(\overline{w}w)|_i\right)$, we know that there is some suffix t of $uw(\overline{w}w)|_i$ such that $v_i \rightsquigarrow^\star \overline{t}t$. We will do a case analysis on the size of t:

- if $n + i \leqslant |t|$, then there is a suffix s of u such that $t = sw(\overline{w}w)|_i$, so $(\overline{w}w)|_i v_i \rightsquigarrow^\star (\overline{w}w)|_i\overline{(\overline{w}w)|_i}\overline{w}\ \overline{s}sw(\overline{w}w)|_i$.
  - If $i < n$ then there is a word p such that $\overline{w} = (\overline{w}w)|_i p$ so

$$\begin{aligned}(\overline{w}w)|_i v_i &\rightsquigarrow^\star (\overline{w}w)|_i\overline{(\overline{w}w)|_i}(\overline{w}w)|_i p\overline{s}sw(\overline{w}w)|_i \\ &\rightsquigarrow (\overline{w}w)|_i p\overline{s}sw(\overline{w}w)|_i = \overline{sw}sw(\overline{w}w)|_i.\end{aligned}$$

- Otherwise we can write $(\overline{w}w)|_i = \overline{w}w_1$ and $w = w_1w_2$, so

$$\begin{aligned}(\overline{w}w)|_i v_i &\rightsquigarrow^\star \overline{w}w_1\overline{\overline{w}w_1}\overline{w}\ \bar{s}sw(\overline{w}w)|_i\\ &= \overline{w_2}\ \overline{w}_1w_1\overline{w}_1w\overline{w}\ \bar{s}sw(\overline{w}w)|_i\\ &\rightsquigarrow \overline{w_2}\ \overline{w}_1w\overline{w}\ \bar{s}sw(\overline{w}w)|_i\\ &= \overline{w}w\overline{w}\ \bar{s}sw(\overline{w}w)|_i\\ &\rightsquigarrow \overline{sw}sw(\overline{w}w)|_i.\end{aligned}$$

  As $s \in \text{suffixes}(u)$ we know that $sw \in \text{suffixes}(uw)$, hence $\overline{sw}sw \in \Gamma(uw)$.

– If $i \leqslant |t| < n + i$ then $w = w_1w_2$ and $t = w_2(\overline{w}w)|_i$ so

$$(\overline{w}w)|_i v_i \rightsquigarrow^\star (\overline{w}w)|_i\overline{(\overline{w}w)|_i}\overline{w_2}w_2(\overline{w}w)|_i$$

  - If $i < n$ then there is a word p such that $\overline{w} = (\overline{w}w)|_i p$. As $\overline{w} = \overline{w_2}\ \overline{w_1}$, we can also compare $(\overline{w}w)|_i$ with $\overline{w_2}$:
    * If $(\overline{w}w)|_i = \overline{w_2}w_3$ then

$$\begin{aligned}(\overline{w}w)|_i v_i &\rightsquigarrow^\star \overline{w_2}w_3\overline{w_3}w_2\overline{w_2}w_2(\overline{w}w)|_i\\ &\rightsquigarrow \overline{w_2}w_3\overline{w_3}w_2(\overline{w}w)|_i\\ &= (\overline{w}w)|_i\overline{(\overline{w}w)|_i}(\overline{w}w)|_i\\ &= \overline{\overline{(\overline{w}w)|_i}}\ \overline{(\overline{w}w)|_i}(\overline{w}w)|_i\end{aligned}$$

      And as $\overline{w} = (\overline{w}w)|_i p$, $w = \overline{p}\overline{(\overline{w}w)|_i}$ so $\overline{(\overline{w}w)|_i} \in \text{suffixes}(w) \subseteq \text{suffixes}(uw)$, hence $\overline{\overline{(\overline{w}w)|_i}}\ \overline{(\overline{w}w)|_i} \in \Gamma(uw)$.
    * If on the other hand $\overline{w_2} = (\overline{w}w)|_i w_3$, we have

$$\begin{aligned}(\overline{w}w)|_i v_i &\rightsquigarrow^\star (\overline{w}w)|_i\overline{(\overline{w}w)|_i}(\overline{w}w)|_i w_3\overline{w_3}\overline{(\overline{w}w)|_i}(\overline{w}w)|_i\\ &\rightsquigarrow (\overline{w}w)|_i w_3\overline{w_3}\overline{(\overline{w}w)|_i}(\overline{w}w)|_i\\ &= \overline{w_2}w_2(\overline{w}w)|_i\end{aligned}$$

      $w_2 \in \text{suffixes}(w) \subseteq \text{suffixes}(uw)$ so $\overline{w}_2w_2 \in \Gamma(uw)$.
  - Otherwise we can write $(\overline{w}w)|_i = \overline{w}w_3$ and $w = w_3w_4$, so

$$\begin{aligned}(\overline{w}w)|_i v_i &\rightsquigarrow^\star \overline{w}w_3\overline{w_3}w_3w_4\overline{w_2}w_2(\overline{w}w)|_i\\ &\rightsquigarrow \overline{w}w_3w_4\overline{w_2}w_2(\overline{w}w)|_i\\ &= \overline{w}w\overline{w_2}w_2(\overline{w}w)|_i\\ &= \overline{w}w_1w_2\overline{w_2}w_2(\overline{w}w)|_i\\ &\rightsquigarrow \overline{w}w_1w_2(\overline{w}w)|_i\\ &= \overline{w}w(\overline{w}w)|_i\end{aligned}$$

      And obviously $\overline{w}w \in \Gamma(uw)$.

– If $|t| < i$ then $(\overline{w}w)|_i = st$. In this case we have $(\overline{w}w)|_i v_i \rightsquigarrow^\star st\bar{t}t \rightsquigarrow st = \bar{\epsilon}\epsilon(\overline{w}w)|_i$, and $\epsilon \in \text{suffixes}(uw)$ so $\bar{\epsilon}\epsilon \in \Gamma(uw)$.

In all cases, we have shown that $(\overline{w}w)_i v_i \rightsquigarrow^\star t_i(\overline{w}w)|_i$ with $t_i \in \Gamma(uw)$.

A.6 Proof of the bisimulation between the two closure constructions

Let us be more precise : starting from a non-deterministic automaton $\mathscr{A} = \langle Q, \mathbf{X}, I, Q_f, \Delta\rangle$, its determinised is $\mathcal{D} = \langle \mathcal{P}(Q), \mathbf{X}, I, T, \delta\rangle$ with

$$T = \{P : P \cap Q_f \neq \emptyset\} \text{ and } \delta(P, x) = P \cdot \Delta(x).$$

We can build two automata recognising its closure. The first one, derived from our construction, is

$$\mathscr{A}_1 = \langle \mathcal{P}(Q) \times G, \mathbf{X}, (I, [\epsilon]), T_1, \delta_1\rangle$$

where G is the set of equivalence relations of $\sim_\gamma$, $T_1 \triangleq \{(P, [u]) \mid P \cap Q_f \neq \emptyset\}$ and

$$\delta_1((P, [u]), x) = (P \cdot (\Delta(x) \circ \gamma(ux)), [ux]).$$

The second one, given by the original construction, is

$$\mathscr{A}_2 = \langle \mathcal{P}(M_\mathcal{D}) \times \mathcal{P}(M_\mathcal{D}), \mathbf{X}, (\underline{\epsilon}, \underline{\epsilon}), T_1, \delta_2\rangle$$

where $M_\mathcal{D}$ is the transition monoid of $\mathcal{D}$, a set of endomorphisms of $\mathcal{P}(Q)$ induced by words, $\underline{w} \triangleq \{w_\mathcal{D}\}$ is a singleton containing the interpretation of a word w in $M_\mathcal{D}$, $T_2 \triangleq \{(F, G) \mid \exists q_f \in Q_f, \exists f \in F : q_f \in f(I)\}$, and the transition function is

$$\delta_2((F, G), x) = (F \odot \underline{x} \odot (\underline{x}' \odot G \odot \underline{x})^\star, (\underline{x}' \odot G \odot \underline{x})^\star).$$

($A \odot B \triangleq \{g \circ f \mid f \in A \wedge g \in B\}$.) The fact that the elements of $M_\mathcal{D}$ are semilattice-homomorphisms can be easily checked, as $u_\mathcal{D}(P)$ is the only state of $\mathcal{D}$ (*i.e.* a set of states of $\mathscr{A}$) such that $P \xrightarrow{u}_\mathcal{D} u_\mathcal{D}(P)$. Then is is straightforward that :

$$\begin{aligned} u_\mathcal{D}(P_1 \cup P_2) &= \{q \in Q \mid \exists p \in P_1 \cup P_2 : p \xrightarrow{u}_\mathscr{A} q\} \\ &= \{q \in Q \mid \exists p \in P_1 : p \xrightarrow{u}_\mathscr{A} q\} \cup \{q \in Q \mid \exists p \in P_2 : p \xrightarrow{u}_\mathscr{A} q\} \\ &= u_\mathcal{D}(P_1) \cup u_\mathcal{D}(P_2). \end{aligned}$$

Now, to give the bisimulation we need the following morphism i from $\mathcal{P}(M_\mathcal{D})$ to $\mathcal{P}(Q^2)$ defined by

$$i(F) \triangleq \{(p, q) \mid \exists f \in F : q \in f(\{p\})\}.$$

Note that i is a KA-homomorphism because the elements of the transition monoid of the determinised automaton are semilattice-homomorphisms from

$\mathcal{P}(Q)$ to $\mathcal{P}(Q)$. Let's check that :

$$\begin{aligned}
\epsilon_{\mathcal{D}} &= \mathrm{Id}_{\mathcal{P}(Q)}, \text{ meaning that } i(\underline{\epsilon}) = \mathrm{Id}_Q;\\
i(F_1 \cup F_2) &= \{(p,q) \mid \exists f \in F_1 \cup F_2 : q \in f(\{p\})\}\\
&= \{(p,q) \mid \exists f \in F_1 : q \in f(\{p\})\} \cup \{(p,q) \mid \exists f \in F_2 : q \in f(\{p\})\}\\
&= i(F_1) \cup i(F_2);\\
i(F_1 \odot F_2) &= \{(p,q) \mid \exists f \in F_1 \odot F_2 : q \in f(\{p\})\}\\
&= \{(p,q) \mid \exists f, g \in F_1 \times F_2 : q \in g \circ f(\{p\})\}\\
&= \{(p,q) \mid \exists f \in F_1 : \exists p' \in f(\{p\}) : \exists g \in F_2 : q \in g(\{p'\})\}\\
&\qquad\qquad (g \text{ is a semilattice homomorphism})\\
&= \{(p,q) \mid \exists p' : (p,p') \in i(F_1) \wedge (p',q) \in i(F_2)\}\\
&= i(F_1) \circ i(F_2)
\end{aligned}$$

For the * operation, recall that

$$\forall F \in \mathcal{P}(M_{\mathcal{D}}), \exists n_1(F) \in \mathbb{N} : \forall m \leqslant n_1(F), F^\star = (F \cup \underline{\epsilon})^m;$$

and that

$$\forall R \in \mathcal{P}(Q)^2, \exists n_2(R) \in \mathbb{N} : \forall m \leqslant n_2(R), R^\star = (R \cup \mathrm{Id}_Q)^m.$$

Then, if we write $m = \max(n_1(F), n_2(u_{\mathcal{D}}(F)))$,

$$\begin{aligned}
i(F^\star) &= i((F \cup \underline{\epsilon})^m)\\
&= (i(F) \cup i(\underline{\epsilon}))^m\\
&= (i(F))^\star
\end{aligned}$$

We can also check that, for any $x \in \mathbf{X}$:

$$\begin{aligned}
i(\underline{x}) &= \{(p,q) \mid q \in x_{\mathcal{D}}(\{p\})\}\\
&= \{(p,q) \mid q \in \delta(\{p\}, x)\}\\
&= \{(p,q) \mid p \xrightarrow{x}_{\mathscr{A}} q\}\\
&= \Delta(x).
\end{aligned}$$

The bisimulation $\sim$ can thus be expressed :

$$\sim \triangleq \{((Q,[u]),\ (F,G)) \mid Q = I \cdot i(F) \text{ and } \gamma(u) = i(G)\}$$

where $(Q,[u])$ is a state of $\mathscr{A}_1$ and (F,G) is a state of $\mathscr{A}_2$. We will now show prove that it is indeed a bisimulation.

1. We need the inital states to be related. This is obvious as $\epsilon_{\mathcal{D}} = \mathrm{Id}_{\mathcal{P}(Q)}$, meaning that $i(\underline{\epsilon}) = \mathrm{Id}_Q$. Furthermore, $\gamma(\epsilon) = \mathrm{Id}_Q$ and $I = I \cdot \mathrm{Id}_Q$. That means $(I,[\epsilon]) \sim (\underline{\epsilon}, \underline{\epsilon})$.

2. For the final states, it isn't much more complicated :

$$\begin{aligned}(F,G) \in T_2 &\Leftrightarrow \exists q_f \in Q_f : \exists f \in F : q_f \in f(I) \\ &\Leftrightarrow \exists q_f \in Q_f : q_f \in I \cdot i(F) \\ &\Leftrightarrow I \cdot i(F) \cap Q_f \neq \emptyset \\ &\Leftrightarrow (I \cdot i(F), i(G)) \in T_1.\end{aligned}$$

3. What remains to be shown is that this relation is stable under transitions from both sides. Suppose that $(Q,[u]) \sim (F,G)$, and consider $x \in \mathbf{X}$. After reading x we get in $\mathscr{A}_2$ $(F \odot \underline{x} \odot G', G')$, with $G' = (\underline{x'} \odot G \odot \underline{x})^\star$, and in $\mathscr{A}_1$ $(Q \cdot (\Delta(x) \circ \gamma(ux)), [ux])$. We will prove that they are still related in two steps, first by looking at the second component, and then dealing with the first one.
   (a) We know that $\gamma(u) = i(G)$, and that $i(\underline{x}) = \Delta(x)$.

$$\begin{aligned}\gamma(ux) &= (\Delta(x') \circ \gamma(u) \circ \Delta(x))^\star \\ &= (i(\underline{x'}) \circ i(G) \circ i(\underline{x}))^\star \\ &= i(G') \qquad\qquad (i \text{ is a morphism})\end{aligned}$$

   (b) Now the first component comes quite easily :

$$\begin{aligned}Q \cdot (\Delta(x) \circ \gamma(ux)) &= (I \cdot i(F)) \cdot (i(\underline{x}) \circ i(G')) \\ &= I \cdot (i(F) \circ i(\underline{x}) \circ i(G')) \\ &= I \cdot i(F \odot \underline{x} \odot G').\end{aligned}$$

Symbolic Algorithms for Language Equivalence and Kleene Algebra with Tests

Damien Pous

Symbolic Algorithms for Language Equivalence and Kleene Algebra with Tests

Damien Pous *
Plume team – CNRS, ENS de Lyon, Université de Lyon, INRIA, UMR 5668, France
Damien.Pous@ens-lyon.fr

Abstract

We propose algorithms for checking language equivalence of finite automata over a large alphabet. We use symbolic automata, where the transition function is compactly represented using (multi-terminal) binary decision diagrams (BDD). The key idea consists in computing a bisimulation by exploring reachable pairs symbolically, so as to avoid redundancies. This idea can be combined with already existing optimisations, and we show in particular a nice integration with the disjoint sets forest data-structure from Hopcroft and Karp's standard algorithm.

Then we consider Kleene algebra with tests (KAT), an algebraic theory that can be used for verification in various domains ranging from compiler optimisation to network programming analysis. This theory is decidable by reduction to language equivalence of automata on guarded strings, a particular kind of automata that have exponentially large alphabets. We propose several methods allowing to construct symbolic automata out of KAT expressions, based either on Brzozowski's derivatives or on standard automata constructions.

All in all, this results in efficient algorithms for deciding equivalence of KAT expressions.

Categories and Subject Descriptors F.4.3 [*Mathematical Logic*]: Decision Problems; F.1.1 [*Models of computation*]: Automata; D.2.4 [*Program Verification*]: Model Checking

Keywords Binary decision diagrams (BDD), symbolic automata, Disjoint set forests, union-find, language equivalence, Kleene algebra with tests (KAT), guarded string automata, Brzozowski's derivatives, Antimirov' partial derivatives.

* We acknowledge support from the following ANR projects: 2010-BLAN-0305 PiCoq and 12IS02001 PACE.

To appear in Proc. POPL 15, January 1517 2015, Mumbai, India.
http://dx.doi.org/10.1145/2676726.2677007

1. Introduction

A wide range of algorithms in computer science build on the ability to check language equivalence or inclusion of finite automata. In model-checking for instance, one can build an automaton for a formula and an automaton for a model, and then check that the latter is included in the former. More advanced constructions need to build a sequence of automata by applying a transducer, and to stop whenever two subsequent automata recognise the same language [7]. Another field of application is that of various extensions of Kleene algebra, whose equational theories are reducible to language equivalence of various kinds of automata: regular expressions and finite automata for plain Kleene algebra [26], "closed" automata for Kleene algebra with converse [5, 15], or guarded string automata for Kleene algebra with tests (KAT) [28].

The theory of KAT has been developed by Kozen et al. [12, 27, 28], it has received much attention for its applications in various verification tasks ranging from compiler optimisation [29] to program schematology [3], and very recently for network programming analysis [2, 17]. Like for Kleene algebra, the equational theory of KAT is PSPACE-complete, making it a challenging task to provide algorithms that are computationally practical on as many inputs as possible.

A difficulty with KAT is that the underlying automata work on an input alphabet which is exponentially large in the number of variables of the starting expressions. As such, it renders standard algorithms for language equivalence intractable, even for reasonably small inputs. This difficulty is shared with other fields where various people proposed to work with *symbolic automata* to cope with large, potentially infinite, alphabets [10, 41]. By symbolic automata, we mean finite automata whose transition function is represented using a compact data-structure, typically binary decision diagrams (BDDs) [9, 10], allowing one to explore the automata in a symbolic way.

D'Antoni and Veanes recently proposed a minimisation algorithm for symbolic automata [13], which is much more efficient than the adaptations of the traditional algorithms [22, 31, 32]. However, to our knowledge, the simpler problem of language equivalence for symbolic automata has not been covered yet. We say 'simpler' because language equivalence can be reduced trivially to minimisation—it suffices to minimise the disjoint union of the automata and to check whether the corresponding initial states are equated—but minimisation has complexity $n\ln n$ while Hopcroft and Karp's algorithm for language equivalence [23] is almost linear [40]. (This latter algorithm for checking language equivalence of finite automata can be seen as an instance of Huet's first-order unification algorithm without occur-check [24, Section 5.8]: one tries to unify the two automata recursively, keeping track of the generated equivalence classes of states using an efficient union-find data-structure.)

Our main contributions are the following:

- We propose a simple coinductive algorithm for checking language equivalence of symbolic automata (Section 3). This algorithm is generic enough to support various improvements that have been proposed in the literature for plain automata [1, 6, 14, 42].
- We show how to combine binary decisions diagrams (BDD) and *disjoint set forests*, the efficient data-structure used by Hopcroft and Karp to define their almost linear algorithm [23, 40] for deterministic automata. This results in a new version of their algorithm, for symbolic automata (Section 3.3).
- We study several constructions for building efficiently a symbolic automaton out of a KAT expression (Section 4): we consider symbolic versions of the extensions of Brzozowski's derivatives [11] and Antimirov' partial derivatives [4] to KAT, as well as a generalisation of Ilie and Yu's inductive construction [25]. The latter construction also requires us to generalise the standard procedure consisting of eliminating epsilon transitions.

Notation

We denote sets by capital letters $X, Y, S, T \ldots$ and functions by lower case letters $f, g, \ldots$ Given sets X and Y, $X \times Y$ is their Cartesian product, $X \uplus Y$ is their disjoint union and X^Y is the set of functions $f\colon Y \to X$. The collection of subsets of X is denoted by $\mathcal{P}(X)$. For a set of letters A, $A^\star$ denotes the set of all finite words over A; ϵ the empty word; and uv the concatenation of words $u, v \in A^\star$. We use 2 for the set $\{0, 1\}$.

2. Preliminary material

We first recall some standard definitions about finite automata and binary decision diagrams.

For finite automata, the only slight difference with the setting described in [6] is that we work with Moore machines [31] rather than automata: the accepting status of a state is not necessarily a Boolean, but a value in a fixed yet arbitrary set. Since this generalisation is harmless, we stick to the standard automata terminology.

2.1 Finite automata

A deterministic finite automaton (DFA) over the input alphabet A and with outputs in B is a triple $\langle S, t, o\rangle$, where S is a finite set of states, $o\colon S \to B$ is the output function, and $t\colon S \to S^A$ is the (total) transition function which returns, for each state x and for each input letter $a \in A$, the next state $t_a(x)$. For $a \in A$, we write $x \xrightarrow{a} x'$ for $t_a(x) = x'$. For $w \in A^\star$, we denote by $x \xrightarrow{w} x'$ the least relation such that (1) $x \xrightarrow{\epsilon} x$ and (2) $x \xrightarrow{au} x'$ if $x \xrightarrow{a} x''$ and $x'' \xrightarrow{u} x'$ for some x''.

The *language* accepted by a state $x \in S$ of a DFA is the function $[\![x]\!]\colon A^\star \to B$ defined as follows:

$$[\![x]\!](\epsilon) = o(x) \ , \qquad [\![x]\!](aw) = [\![t_a(x)]\!](w) \ .$$

(When the output set is 2, these functions are indeed characteristic functions of formal languages). Two states $x, y \in S$ are said to be *language equivalent* (written $x \sim y$) when they accept the same language.

2.2 Coinduction

Checking whether two states of two distinct automata recognise the same language reduces to checking whether two states of a single automaton recognise the same language: one can always build the disjoint union of the two automata. We thus fix a single DFA, and we define bisimulations. We make explicit the underlying notion of progression which we need in the sequel.

```
1  type (s,β) dfa = {t: s → A → s; o: s → β}
2
3  let equiv (M: (s,β) dfa) (x y: s) =
4    let r = Set.empty () in
5    let todo = Queue.singleton (x,y) in
6    while ¬Queue.is_empty todo do
7      (* invariant: r ⇀ r ∪ todo *)
8      let (x,y) = Queue.pop todo in
9      if Set.mem r (x,y) then continue
10     if M.o x ≠ M.o y then return false
11     iter_A (fun a → Queue.push todo (M.t x a, M.t y a))
12     Set.add r (x,y)
13   done
14   return true
```

Figure 1. Simple algorithm for checking language equivalence.

Definition 1 (Progression, Bisimulation)**.** *Given two relations $R, R' \subseteq S \times S$ on the states of an automaton, R* progresses to *R', denoted $R \rightarrowtail R'$, if whenever $x\ R\ y$ then*

1. *$o(x) = o(y)$ and*
2. *for all $a \in A$, $t_a(x)\ R'\ t_a(y)$.*

A bisimulation *is a relation R such that $R \rightarrowtail R$.*

Bisimulations provide a sound and complete proof technique for checking language equivalence of DFA:

Proposition 1 (Coinduction)**.** *Two states of an automaton are language equivalent iff there exists a bisimulation that relates them.*

Accordingly, we obtain the simple algorithm described in Figure 1, for checking language equivalence of two states of the given automaton.

This algorithm works as follows: the variable `r` contains a relation which is a bisimulation candidate and the variable `todo` contains a queue of pairs that remain to be processed. To process a pair (x, y), one first checks whether it already belongs to the bisimulation candidate: in that case, the pair can be skipped since it was already processed. Otherwise, one checks that the outputs of the two states are the same ($o(x) = o(y)$), and one pushes all derivatives of the pair to the `todo` queue: all pairs $(t_a(x), t_a(y))$ for $a \in A$. (This requires the type A of letters to be iterable, and thus finite, an assumption which is no longer required with the symbolic algorithm to be presented in Section 3.) The pair (x, y) is finally added to the bisimulation candidate, and we proceed with the remainder of the queue.

The main invariant of the loop (line 7: $\texttt{r} \rightarrowtail \texttt{r} \cup \texttt{todo}$) ensures that when `todo` becomes empty, then `r` contains a bisimulation, and the starting states were indeed bisimilar. Another invariant of the loop is that for any pair (x', y') in `todo`, there exists a word w such that $x \xrightarrow{w} x'$ and $y \xrightarrow{w} y'$. Therefore, if we reach a pair of states whose outputs are distinct—line 10, then the word w associated to that pair witnesses the fact that the two initial states are not equivalent.

Remark 1. *Note that such an algorithm can be modified to check for language inclusion in a straightforward manner: assuming an arbitrary preorder $\leq$ on the output set B, and letting language inclusion mean $x \leq y$ if for all $w \in A^\star$, $[\![x]\!](w) \leq [\![y]\!](w)$, it suffices to replace line 10 in Figure 1 by*

```
if ¬(M.o x ≤ M.o y) then return false.
```

2.3 Up-to techniques

The previous algorithm can be enhanced by exploiting *up-to techniques* [36, 39]: an up-to technique is a function f on binary rela-

tions such that any relation R satisfying $R \rightarrowtail f(R)$ is contained in a bisimulation. Intuitively, such relations, that are not necessarily bisimulations, are constrained enough to contain only language equivalent pairs.

We have recently shown with Bonchi [6] that the standard algorithm by Hopcroft and Karp [23] actually exploits such an up-to technique: on line 9, rather than checking whether the processed pair is already in the candidate relation `r`, Hopcroft and Karp check whether it belongs to the equivalence closure of `r`. Indeed the function e mapping a relation to its equivalence closure is a valid up-to technique, and this optimisation allows the algorithm to stop earlier. Hopcroft and Karp moreover use an efficient data-structure to perform this check in almost constant time [40]: *disjoint sets forests*. We recall this data-structure in Section 3.3.

Other examples of valid up-to techniques include context-closure, as used in antichain based algorithms [1, 14, 42], or congruence closure [6], which combines both context-closure and equivalence closure. These techniques require working with automata whose states carry a semi-lattice structure, as is typically the case for a DFA obtained from a non-deterministic automaton through the powerset construction.

2.4 Binary decision diagrams

Assume an ordered set $(A, <)$ and an arbitrary set B. Binary decision diagrams are directed acyclic graphs that can be used to represent functions of type $2^A \to B$. When $B = 2$ is the two elements set, BDDs thus intuitively represent Boolean formulas with variables in A.

Formally, a *(multi-terminal, ordered) binary decision diagram* (BDD) is a pair (N, c) where N is a finite set of nodes and c is a function of type $N \to B \uplus (A \times N \times N)$ such that if $c(n) = (a, l, r)$ and either $c(l) = (a', \_, \_)$ or $c(r) = (a', \_, \_)$, then $a < a'$.

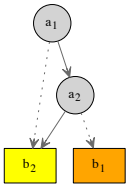
The condition on c ensures that the underlying graph is acyclic, which makes it possible to associate a function $\lceil n \rceil : 2^A \to B$ to each node n of a BDD:

$$\lceil n \rceil(\alpha) = \begin{cases} b & \text{if } c(n) = b \in B \\ \lceil l \rceil(\alpha) & \text{if } c(n) = (a, l, r) \text{ and } \alpha(a) = 0 \\ \lceil r \rceil(\alpha) & \text{if } c(n) = (a, l, r) \text{ and } \alpha(a) = 1 \end{cases}$$

Let us now recall the standard graphical representation of BDDs:

- A node n such that $c(n) = b \in B$ is represented by a square box labelled by b.
- A node n such that $c(n) = (a, l, r) \in A \times N \times N$ is a decision node, which we picture by a circle labelled by a, with a dotted arrow towards the *left child* (l) and a plain arrow towards the *right child* (r).

For instance, the following drawing represents a BDD with four nodes; its top-most node denotes the function given on the right-hand side.



$$\alpha \mapsto \begin{cases} b_1 & \text{if } \alpha(a_1) = 1 \text{ and } \alpha(a_2) = 0 \\ b_2 & \text{otherwise} \end{cases}$$

A BDD is *reduced* if c is injective, and $c(n) = (a, l, r)$ entails $l \neq r$. (The above example BDD is reduced.) Any BDD can be transformed into a reduced one. When A is finite, reduced (ordered) BDD nodes are in one-to-one correspondence with functions from 2^A to B [9, 10]. The main interest in this data-structure is that it is often extremely compact.

```
1  type β node = β descr hash_consed
2  and  β descr = V of β | N of A × β node × β node
3
4  val hashcons: β descr → β node
5  val c: β node → β descr
6  val memo_rec: ((α'→β'→γ)→α'→β'→γ)→α'→β'→γ
7   (* with α' = α hash_consed, β' = β hash_consed *)
8
9  let constant v = hashcons (V v)
10 let node a l r = if l==r then l else hashcons (N(a,l,r))
11
12 let apply (f: α → β → γ): α node → β node → γ node =
13  memo_rec (fun app x y →
14  match c(x), c(y) with
15    | V v, V w → constant (f v w)
16    | N(a,l,r), V _ → node a (app l y) (app r y)
17    | V _, N(a,l,r) → node a (app x l) (app x r)
18    | N(a,l,r), N(a',l',r') →
19      if a=a' then node a (app l l') (app r r')
20      if a<a' then node a (app l y ) (app r y )
21      if a>a' then node a' (app x l') (app x r'))
```

Figure 2. An implementation of BDDs.

In the sequel, we only work with reduced ordered BDDs, which we simply call BDDs. We denote by $\mathrm{BDD}_A[B]$ the set of nodes of a BDD with values in B, which is large enough to represent all considered functions. We let $\lfloor f \rfloor$ denote the unique BDD node representing a given function $f : 2^A \to B$. This notation is useful to give abstract specifications to BDD operations: in the sequel, all usages of this notation actually underpin efficient BDD operations.

Implementation. To better explain parts of the proposed algorithms, we give a simple implementation of BDDs in Figure 2.

The type for BDD nodes is given first: we use Filliâtre's hash-consing library [16] to enforce unique representation of each node, whence the two type declarations and the two conversion functions `hashcons` and `c` between those types. The third utility function `memo_rec` is just a convenient operator for defining recursive memoised functions on pairs of hash-consed values.

The function `constant` creates a constant node, making sure it was not already created. The function `node` creates a new decision node, unless that node is useless and can be replaced by one of its two children. The generic function `apply` is central to BDDs [9, 10]: many operations are just instances of this function. Its specification is the following:

$$\texttt{apply}\ f\ x\ y = \lfloor \alpha \mapsto f(\lceil x \rceil(\alpha))(\lceil y \rceil(\alpha)) \rfloor$$

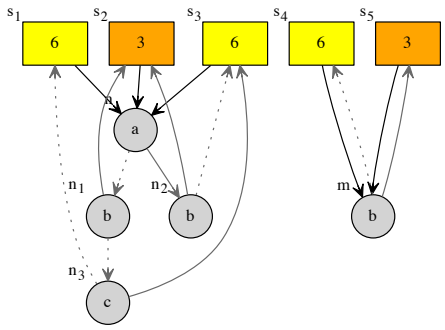
This function is obtained by "zipping" the two BDDs together until a constant is reached. Memoisation is used to exploit sharing and to avoid performing the same computations again and again.

Suppose now that we want to define logical disjunction on Boolean BDD nodes. Its specification is the following:

$$x \vee y = \lfloor \alpha \mapsto \lceil x \rceil(\alpha) \vee \lceil y \rceil(\alpha) \rfloor.$$

We can thus simply use the `apply` function, applied to the Boolean disjunction function:

```
1 let dsj: bool node → bool node → bool node = apply (||)
```



	s_1, s_2, s_3								s_4, s_5							
a	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
b	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
c	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
t	s_1	s_3	s_2	s_2	s_3	s_3	s_2	s_2	s_4	s_4	s_5	s_5	s_4	s_4	s_5	s_5

Figure 3. A symbolic DFA with five states.

Note that this definition could actually be slightly optimised by inlining `apply`'s code, and noticing that the result is already known whenever one of the two arguments is a constant:

```
let dsj: bool node → bool node → bool node =
 memo_rec (fun dsj x y →
 match c(x), c(y) with
   | V true, _ | _, V false → x
   | _, V true | V false, _ → y
   | N(a,l,r), N(a',l',r') →
     if a=a' then node a (dsj l l') (dsj r r')
     if a<a' then node a (dsj l y ) (dsj r y )
     if a>a' then node a' (dsj x l') (dsj x r'))
```

We ignore such optimisations in the sequel, for the sake of clarity.

3. Symbolic automata

A standard technique [10, 13, 20, 41] for working with automata over a large input alphabet consists in using BDDs to represent the transition function: a *symbolic DFA* with output set B and input alphabet $A' = 2^A$ for some set A is a triple $\langle S, t, o\rangle$ where S is the set of states, $t\colon S \to \mathrm{BDD}_A[S]$ maps states into nodes of a BDD over A with values in S, and $o\colon S \to B$ is the output function.

Such a symbolic DFA is depicted in Figure 3. It has five states, input alphabet $2^{\{a,b,c\}}$, and natural numbers as output set. We represent the BDD graphically; rather than giving the functions t and o separately, we label the square box corresponding to a state x with its output value $o(x)$ and we link this box to the node $t(x)$ defining the transitions of x using a solid arrow. The explicit transition table is given below the drawing.

The simple algorithm described in Figure 1 is not optimal when working with such symbolic DFAs: at each non-trivial iteration of the main loop, one goes through all letters of $A' = 2^A$ to push all the derivatives of the current pair of states to the queue `todo` (line 11), resulting in a lot of redundancies.

Suppose for instance that we run the algorithm on the DFA of Figure 3, starting from states s_1 and s_4. After the first iteration, `r` contains the pair (s_1, s_4), and the queue `todo` contains eight pairs:

$$(s_1, s_4), (s_3, s_4), (s_2, s_5), (s_2, s_5), (s_3, s_4), (s_3, s_4), (s_2, s_5), (s_2, s_5)$$

```
let iter2 (f: α × β → unit): α node → β node → unit =
 memo_rec (fun iter2 x y →
 match c(x), c(y) with
   | V v, V w → f (v,w)
   | V _, N(_,l,r) → iter2 x l; iter2 x r
   | N(_,l,r), V _ → iter2 l y; iter2 r y
   | N(a,l,r), N(a',l',r') →
     if a=a' then iter2 l l'; iter2 r r'
     if a<a' then iter2 l y ; iter2 r y
     if a>a' then iter2 x l'; iter2 x r')
```

Figure 4. Iterating over the set of pairs reachable from two nodes.

```
type (s,β) sdfa = {t: s → s bdd; o: s → β}

let symb_equiv (M: (s,β) sdfa) (x y: s) =
  let r = Set.empty() in
  let todo = Queue.singleton (x,y) in
  let push_pairs = iter2 (Queue.push todo) in
  while ¬Queue.is_empty todo do
    let (x,y) = Queue.pop todo in
    if Set.mem (x,y) r then continue
    if M.o x ≠ M.o y then return false
    push_pairs (M.t x) (M.t y)
    Set.add r (x,y)
  done;
  return true
```

Figure 5. Symbolic algorithm for checking language equivalence.

Assume that elements of this queue are popped from left to right. The first element is removed during the following iteration, since (s_1, s_4) already is in `r`. Then (s_3, s_4) is processed: it is added to `r`, and the above eight pairs are appended again to the queue, which now has fourteen elements. The following pair is processed similarly, resulting in a queue with twenty one $(14 - 1 + 8)$ pairs. Since all pairs of this queue are already in `r`, it is finally emptied through twenty one iterations, and the algorithm returns true.

Note that it would be even worse if the input alphabet was actually declared to be $2^{\{a,b,c,d\}}$: even though the bit d of all letters is irrelevant for the considered DFA, each non-trivial iteration of the algorithm would push even more copies of each pair to the `todo` queue.

What we propose here is to exploit the symbolic representation, so that a given pair is pushed only once. Intuitively, we want to recognise that starting from the pair of nodes (n, m), the letters 010, 011, 110 and 111 are equivalent[1], since they lead to the same pair, (s_2, s_5). Similarly, the letters 001, 100, and 101 are equivalent: they lead to the pair (s_3, s_4).

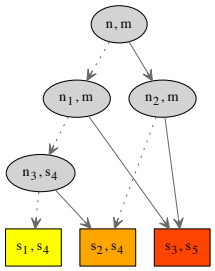
This idea is easy to implement using BDDs: like for the `apply` function (Figure 2), it suffices to zip the two BDDs together, and to push pairs when we reach two leaves. We use for that the procedure `iter2` from Figure 3, which successively applies a given function to all pairs reachable from two nodes. Its code is almost identical to `apply`, except that nothing is constructed (and memoisation is just used to remember those pairs that have already been visited).

We finally modify the simple algorithm from Section 2.1 by using this procedure on line 11; we obtain the code given in Figure 5. We apply `iter2` to its first argument once and for all (line 6), so that we maximise memoisation: a pair of nodes that has been vis-

[1] Letters being elements of $2^{\{a,b,c\}}$ here, we represent them with bit-vectors of length three

ited in the past will never be visited again, since all pairs of states reachable from that pair of nodes are already guaranteed to be processed. (As an invariant, we have that all pairs reachable from a pair of nodes memoised in `push_pairs` appear in `r` ∪ `todo`.)

Let us illustrate this algorithm by running it on the DFA from Figure 3, starting from states s_1 and s_4 as previously. During the first iteration, the pair (s_1, s_4) is added to `r`, and `push_pairs` is called on the pair of nodes (n, m). This call virtually results in building the following BDD, where leaves consist of calls to `Queue.push todo`.



The following three pairs are thus pushed to `todo`.

$$(s_1, s_4), (s_3, s_4), (s_2, s_5)$$

The first pair is removed by a trivial iteration: (s_1, s_4) already belongs to `r`. The two other pairs are processed by adding them to `r`, but without pushing any new pair to `todo`: thanks to memoisation, the two expected calls to `push_pairs n m` are skipped.

All in all, each reachable pair is pushed only once to the `todo` queue. More importantly, the derivatives of a given pair are explored symbolically. In particular, the algorithm would execute exactly in the same way, even if the alphabet was actually declared to be much larger (for instance because the considered states were part of a bigger automaton with more letters). In fact, the main loop is executed at most n^2 times, where n is the total number of BDD nodes (both leaves and decision nodes) reachable from the starting states.

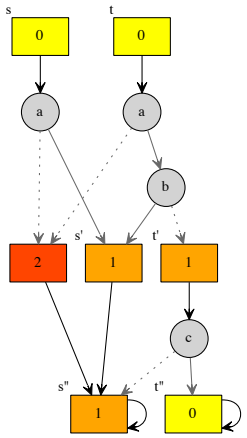
Finally note that in the code from Figure 5, the candidate relation `r` is redundant, as the pairs it contains are also stored implicitly in the memoisation table of `iter2` (except for the initial pair). The corresponding lines (4, 9, and 12) can thus be removed.

3.1 Displaying symbolic counter-examples.

Bisimulation-based algorithms for language equivalence can be instrumented to produce counter-examples in case of failure, i.e., a word which is accepted by one state and not by the other.

An advantage of the previous algorithm is that those counter-examples can be displayed symbolically; thus enhancing readability. This is particularly important in the context of formal assisted proofs (e.g., when working with KAT in Coq [34]), where a plain guarded string is often too big to be useful to the user, while a 'symbolic' guarded string—where only the bits that are relevant for the counter-example are displayed—can be really helpful to understand which hypotheses have to be used to solve the current goal.

Consider for instance the following automaton.



Intuitively, the topmost states s and t are not equivalent because t can take two transitions to reach t'', with output 0, while with two transitions, s can only reach s'', with output 1.

More formally, the word 100 001 over $2^{\{a,b,c\}}$ is a counter-example: we have

$$[\![s]\!](100\ 001) = [\![s']\!](001) = o(s'') = 1 \ ,$$
$$[\![t]\!](100\ 001) = [\![t']\!](001) = o(t'') = 0 \ .$$

But there are plenty of other counter-examples of length two: it suffices that: a be assigned true and b be assigned false in the first letter, and that c be assigned true in the second letter. The values of the bit c in the first letter, and of the bits a and b in the second letter do not change the above computation. As a consequence, this counter-example is best described as the pseudo-word 10- --1, or alternatively the word $(a \wedge \neg b)\, c$ whose letters are conjunctions of literals indicating the least requirements to get a counter example.

The algorithm from Figure 5 makes it possible to give this information back to the user:

- modify the queue *todo* to store triples (w, x, y) where (x, y) is a pair of states to process, and w is the associated potential counter-example;
- modify the function `iter2` (Figure 3), so that it uses an additional argument to record the encountered node labels, with negative polarity when going through the recursive call for the left child, and positive polarity for the right child;
- modify line 10 of the main algorithm to return the symbolic word associated with the current pair when the output test fails.

3.2 Non-deterministic automata

Standard coinductive algorithms for DFA can be applied to non-deterministic automata (NFA) by using the *powerset construction*. This construction transforms a non-deterministic automaton into a deterministic one; we extend it to symbolic automata in the obvious way.

A *symbolic NFA* is a tuple $\langle S, t, o \rangle$ where S is the set of states, $o\colon S \to B$ is the output function, and $t\colon S \to \mathrm{BDD}_A[\mathcal{P}(S)]$ maps a state and a letter of the alphabet $A' = 2^A$ to a set of possible successor states, using a symbolic representation. The set B of output values must be equipped with a semi-lattice structure $\langle B, \wedge, \bot \rangle$. Assuming such an NFA, one defines a symbolic DFA

$\langle \mathcal{P}(S), t^\sharp, o^\sharp \rangle$ as follows:

$$t^\sharp(\{x_1, \ldots, x_n\}) \triangleq t(x_1) \sqcup \ldots \sqcup t(x_n) \ ,$$
$$o^\sharp(\{x_1, \ldots, x_n\}) \triangleq o(x_1) \vee \cdots \vee o(x_n) \ .$$

(Where $\sqcup$ denotes the pointwise union of two BDDs over sets: $n \sqcup m = \lfloor \phi \mapsto \lceil n \rceil(\phi) \cup \lceil m \rceil(\phi) \rfloor$.)

This DFA has exponentially many states. However, when applying bisimulation-based algorithms to such automata, one explores them on the fly, and only those subsets that are reachable from the initial states need to be visited. This number of reachable subsets is usually much smaller than the exponential worst-case bound; in fact it is quite often of the same order as the number of states of the starting DFA (see, e.g., the experiments in Section 5).

3.3 Hopcroft and Karp: disjoint sets forests

The previous algorithm can be freely enhanced by using up-to techniques, as described in Section 2.3: it suffices to modify line 9 to skip pairs more or less aggressively, according to the chosen up-to technique. For an up-to technique f, line 9 thus becomes

```
if Set.mem (x,y) (f r) then continue .
```

The up-to-equivalence technique used in Hopcroft and Karp's algorithm can however be integrated in a deeper way, by exploiting the fact that we work with BDDs. This leads to a second algorithm, which we describe in this section.

Let us first recall *disjoint sets forests*, the data structure used by Hopcroft and Karp to represent equivalence classes. This standard data-structure makes it possible to check whether two elements belong to the same class and to merge two equivalence classes, both in almost constant amortised time [40].

The idea consists in storing a partial map from elements to elements and whose underlying graph is acyclic. An element for which the map is not defined is the *representative* of its equivalence class, and the representative of an element pointing in the map to some y is the representative of y. Two elements are equivalent if and only if they lead to the same representative; to merge two equivalence classes, it suffices to add a link from the representative of one class to the representative of the other class. Two optimisations are required to obtain the announced theoretical complexity:

- when following the path leading from an element to its representative, one should compress it in some way, by modifying the map so that the elements in this path become closer to their representative. There are various ways of compressing paths, in the sequel, we use the method called *halving* [40];
- when merging two classes, one should make the smallest one point to the biggest one, to avoid generating too many long paths. Again, there are several possible heuristics, but we elude this point in the sequel.

As explained above, the simplest thing to do would be to replace the bisimulation candidate r from Figure 5 by a disjoint sets forest over the states of the considered automaton.

The new idea consists in relating the BDD nodes of the symbolic automaton rather than just its states (i.e., just the BDD leaves). By doing so, one avoids visiting pairs of nodes that have already been visited up to equivalence.

Concerning the implementation, we first introduce a BDD unification algorithm (Figure 3.3), i.e., a variant of the function `iter2` which uses disjoint sets forest rather than plain memoisation. This function first creates an empty forest (we use Filliâtre's module `Hmap` of maps over hash-consed values to represent the corresponding partial maps). The function `link` adds a link between two representatives; the recursive terminal function `repr` looks for the representative of a node and implements halving. The inner function

```
1  let unify (f: β × β → unit): β node → β node → unit =
2   (* the disjoint sets forest *)
3   let m = Hmap.empty() in
4   let link x y = Hmap.add m x y in
5   (* representative of a node *)
6   let rec repr x =
7    match Hmap.get m x with
8     | None → x
9     | Some y → match Hmap.get m y with
10      | None → y
11      | Some z → link x z; repr z
12  in
13  let rec unify x y =
14   let x = repr x in
15   let y = repr y in
16   if x ≠ y then
17   match c(x), c(y) with
18    | V v, V w → link x y; f (v,w)
19    | V _, N(_,l,r) → link y x; unify x l; unify x r
20    | N(_,l,r), V _ → link x y; unify l y; unify r y
21    | N(a,l,r), N(a',l',r') →
22      if a=a' then link x y; unify l l'; unify r r'
23      if a<a' then link x y; unify l y ; unify r y
24      if a>a' then link y x; unify x l'; unify x r')
25  in unify
```

Figure 6. Unifying two nodes of a BDD, using disjoint set forests.

```
1 let dsf_equiv (M: (s,β) sdfa) (x y: s) =
2  let todo = Queue.singleton (x,y) in
3  let push_pairs = unify (Queue.push todo) in
4  while ¬Queue.is_empty todo do
5   let (x,y) = Queue.pop todo in
6   if M.o x ≠ M.o y then return false
7   push_pairs (M.t x) (M.t y)
8  done;
9  return true
```

Figure 7. Symbolic algorithm optimised with disjoint set forests.

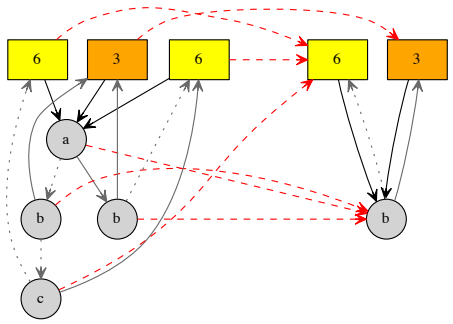
`unify` is defined similarly as `iter2`, except that it first takes the representative of the two given nodes, and that it adds a link from one to the other before recursing.

Those links can be put in any direction on lines 18 and 22, and we should actually use an appropriate heuristic to take this decision, as explained above. In the four other cases, we put a link either from the node to the leaf, or from the node with the smallest label to the node with the biggest label. By proceeding this way, we somehow optimise the BDD, by leaving as few decision nodes as possible.

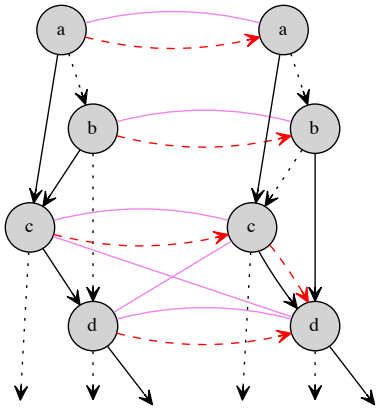
It is important to notice that there is actually no choice left in those four cases: we work implicitly with the optimised BDD obtained by mapping all nodes to their representatives, so that we have to maintain the invariant that this optimised BDD is ordered and acyclic. (Notice that this optimised BDD need not be reduced anymore: the children of given a node might be silently equated, and a node might have several representations since its children might be silently equated with the children of another node with the same label)

We finally obtain the algorithm given in Figure 7. It is similar to the previous one (Figure 5), except that we use the new function `unify` to push pairs into the `todo` queue, and that we no longer store the bisimulation candidate `r`: this relation is subsumed by the restriction of the disjoint set forests to BDD leaves.

If we execute this algorithm on the symbolic DFA from Figure 3, between states s_1 and s_4, we obtain the disjoint set forest depicted below using dashed red arrows. This actually corresponds to the pairs which would be visited by the first symbolic algorithm (Figure 5).



If instead we start from the top-most nodes in the following partly described automaton, we would get the disjoint set forest depicted similarly in red, while the first algorithm would go through all violet lines, one of which is superfluous.



The corresponding optimised BDD consists of the three nodes labelled with a, b, and d on the right-hand side. This BDD is not reduced, as explained above: the node labelled with b should be removed since it points twice to the node labelled with d, and removing this node makes the node labelled with a useless, in turn.

Complexity. Concerning complexity, while the algorithm from Figure 5 is quadratic in the number n of BDD nodes (and leaves) that are reachable from the starting symbolic DFA, the optimised algorithm from Figure 7 performs at most n iterations: two equivalence classes of nodes are merged each time a link is added, and we start with the discrete partition of nodes.

Unfortunately, we cannot immediately deduce that the algorithm is almost linear, as did Tarjan for Hopcroft and Karp's algorithm [40]. The problem is that we cannot always freely choose how to link two representatives (i.e., on lines 19, 20, 23, and 24 in Figure 3.3), so that we cannot guarantee that the amortised complexity of maintaining those equivalence classes is almost constant. We conjecture that such a result holds, however, as the choice we enforce in those cases virtually suppresses binary decision nodes, thus reducing the complexity of subsequent BDD unifications.

Unification with row types. As mentioned in the Introduction, Hopcroft and Karp's algorithm can be seen as an instance of Huet's first-order unification algorithm for recursive terms (i.e., without occur-check). The algorithm presented in Figure 7, and more specifically the BDD unification sub-algorithm (Figure 3.3) is reminiscent of Rémy's extension of this unification algorithm for dealing with *row types*—to obtain an ML-like type inference algorithm in presence of extensible records [33, 37, 38].

More precisely, row types are almost-constant functions from a given set of labels to types, typically represented as association lists with a default value. Unification of such row types is performed pointwise, and is implemented by zipping the two association lists together, as we do here with BDDs (which generalise from almost constant functions to functions with finitely many output values).

It would thus be interesting to understand whether our generalisation of this unification sub-algorithm, from association lists to BDDs, could be useful in the context of unification: either by exploiting the richer structure of functions represented by BDDs, or just for the sake of efficiency, when the set of labels is large (e.g., for type inference on object-oriented programs, where labels correspond to method names).

4. Kleene algebra with tests

Now we consider Kleene algebra with tests (KAT), for which we provide several automata constructions that allow us to use the previous symbolic algorithms.

A *Kleene algebra with tests* is a tuple $\langle X, B, \cdot, +, \cdot^\star, \neg, 1, 0\rangle$ such that:

(i) $\langle X, \cdot, +, \cdot^\star, 1, 0\rangle$ is a Kleene algebra [26], i.e., an idempotent semiring with a unary operation, called "Kleene star", satisfying the following axioms:

$$1 + x \cdot x^\star \leq x^\star$$
$$y \cdot x \leq x \;\Rightarrow\; y^\star \cdot x \leq x$$
$$x \cdot y \leq x \;\Rightarrow\; x \cdot y^\star \leq x$$

(the preorder $(\leq)$ being defined by $x \leq y \;\triangleq\; x + y = y$);

(ii) $B \subseteq X$;

(iii) $\langle B, \cdot, +, \neg, 1, 0\rangle$ is a Boolean algebra.

The elements of the set B are called "tests"; we denote them by ϕ, ψ. The elements of X, called "Kleene elements", are denoted by x, y, z. We sometimes omit the operator "$\cdot$" from expressions, writing xy for $x \cdot y$. The following (in)equations illustrate the kind of laws that hold in all Kleene algebra with tests:

$$\phi + \neg\phi = 1 \qquad \phi \cdot (\neg\phi + \psi) = \phi \cdot \psi = \neg(\neg\phi + \neg\psi)$$

$$x^\star x^\star = x^\star \qquad (x+y)^\star = x^\star(yx^\star)^\star \qquad (x+xxy)^\star \leq (x+xy)^\star$$

$$\phi \cdot (\neg\phi \cdot x)^\star = \phi \qquad \phi \cdot (\phi \cdot x \cdot \neg\phi + \neg\phi \cdot y \cdot \phi)^\star \cdot \phi \leq (x \cdot y)^\star$$

The laws from the first line come from the Boolean algebra structure, while the ones from the second line come from the Kleene algebra structure. The two laws from the last line require both Boolean algebra and Kleene algebra reasoning.

Binary relations. Binary relations form a Kleene algebra with tests; this is the main model we are interested in, in practice. The Kleene elements are the binary relations over a given set S, the tests are the predicates over this set, encoded as sub-identity relations, and the star of a relation is its reflexive transitive closure.

This relational model is typically used to interpret imperative programs: such programs are state transformers, i.e., binary relations between states, and the conditions used to define the control-flow of these programs are just predicates on states. Typically, a program "`while` ϕ `do` p" is interpreted through the KAT expression $(\phi \cdot p)^\star \cdot \neg\phi$.

KAT expressions. We denote by $\mathcal{R}eg(V)$ the set of *regular expressions* over a set V:

$$x, y ::= v \in V \mid x + y \mid x \cdot y \mid x^\star \ .$$

Assuming a set A of elementary tests, we denote by $B(A)$ the set of *Boolean expressions* over A:

$$\phi, \psi ::= a \in A \mid 1 \mid 0 \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \ .$$

Further assuming a set Σ of letters (or atomic Kleene elements), a *KAT expression* is a regular expression over the disjoint union $\Sigma \uplus B(A)$. We let p, q range over elements of Σ. Note that the constants 0 and 1 from the signature of KAT, and usually found in the syntax of regular expressions, are represented here by injecting the corresponding tests.

Guarded string languages. Guarded string languages are the natural generalisation of string languages for Kleene algebra with tests. We briefly define them.

An *atom* is a valuation from elementary tests to Booleans; it indicates which of these tests are satisfied. We let α, β range over atoms, the set of which is denoted by At: $At = 2^A$. A Boolean formula ϕ is *valid* under an atom α, denoted by $\alpha \vDash \phi$, if ϕ evaluates to true under the valuation α.

A *guarded string* is an alternating sequence of atoms and letters, both starting and ending with an atom:

$$\alpha_1, p_1, \alpha_2, \ldots, \alpha_n, p_n, \alpha_{n+1} \ .$$

The concatenation $u * v$ of two guarded strings u, v is a partial operation: it is defined only if the last atom of u is equal to the first atom of v; it consists in concatenating the two sequences and removing one copy of the shared atom in the middle.

To any KAT expression, one associates a *guarded string language*, i.e., a set of guarded strings, as follows.

$$\begin{aligned} G(\phi) &= \{\alpha \in At \mid \alpha \vDash \phi\} && (\phi \in B(A)) \\ G(p) &= \{\alpha p \beta \mid \alpha, \beta \in At\} && (p \in \Sigma) \\ G(x + y) &= G(x) \cup G(y) \\ G(xy) &= \{u * v \mid u \in G(x), v \in G(y)\} \\ G(x^\star) &= \{u_1 * \cdots * u_n \mid \exists u_1 \ldots u_n, \forall i \leq n, u_i \in G(x)\} \end{aligned}$$

KAT Completeness. Kozen and Smith proved that the equational theory of Kleene algebra with tests is complete over the relational model [30]: any equation that holds universally in this model can be proved from the axioms of KAT. Moreover, two expressions are provably equal if and only if they denote the same language of guarded strings. By a reduction to automata theory this gives algorithms to decide the equational theory of KAT. Now we study several such algorithms, and we show each time how to exploit symbolic representations to make them efficient.

4.1 Brzozowski's derivatives

Derivatives were introduced by Brzozowski [11] for (plain) regular expressions; they make it possible to define a deterministic automaton where the states of the automaton are the regular expressions themselves.

Derivatives can be extended to KAT expressions in a very natural way [28]. We recall this extension in Figure 8: one first defines a Boolean function ϵ_α, that indicates whether an expression accepts

$$\begin{aligned} \epsilon_\alpha(x+y) &= \epsilon_\alpha(x) + \epsilon_\alpha(y) & \delta_{\alpha p}(x+y) &= \delta_{\alpha p}(x) + \delta_{\alpha p}(y) \\ \epsilon_\alpha(x \cdot y) &= \epsilon_\alpha(x) \cdot \epsilon_\alpha(y) & \delta_{\alpha p}(x \cdot y) &= \begin{cases} \delta_{\alpha p}(x) \cdot y & \text{if } \epsilon_\alpha(x) = 0 \\ \delta_{\alpha p}(x) \cdot y + \delta_{\alpha p}(y) & \text{oth.} \end{cases} \\ \epsilon_\alpha(x^\star) &= 1 & \delta_{\alpha p}(x^\star) &= \delta_{\alpha p}(x) \cdot x^\star \\ \epsilon_\alpha(q) &= 0 & \delta_{\alpha p}(q) &= \begin{cases} 1 & \text{if } p = q \\ 0 & \text{oth.} \end{cases} \\ \epsilon_\alpha(\phi) &= \begin{cases} 1 & \text{if } \alpha \vDash \phi \\ 0 & \text{oth.} \end{cases} & \delta_{\alpha p}(\phi) &= 0 \end{aligned}$$

Figure 8. Explicit derivatives for KAT expressions

$$\begin{aligned} \epsilon^{\mathrm{s}}(x+y) &= \epsilon^{\mathrm{s}}(x) \vee \epsilon^{\mathrm{s}}(y) & \delta^{\mathrm{s}}(x+y) &= \delta^{\mathrm{s}}(x) \oplus \delta^{\mathrm{s}}(y) \\ \epsilon^{\mathrm{s}}(x \cdot y) &= \epsilon^{\mathrm{s}}(x) \wedge \epsilon^{\mathrm{s}}(y) & \delta^{\mathrm{s}}(x \cdot y) &= (\delta^{\mathrm{s}}(x) \odot y) \oplus (\epsilon^{\mathrm{s}}(x) \otimes \delta^{\mathrm{s}}(y)) \\ \epsilon^{\mathrm{s}}(x^\star) &= 1 & \delta^{\mathrm{s}}(x^\star) &= \delta^{\mathrm{s}}(x) \odot x^\star \\ \epsilon^{\mathrm{s}}(p) &= 0 & \delta^{\mathrm{s}}(p) &= \lceil p \mapsto 1, \_ \mapsto 0 \rceil \\ \epsilon^{\mathrm{s}}(\phi) &= \phi & \delta^{\mathrm{s}}(\phi) &= 0 \end{aligned}$$

Figure 9. Symbolic derivatives for KAT expressions

the single atom α; this function is then used to define the derivation function $\delta_{\alpha p}$, that intuitively returns what remains of the given expression after reading the atom α and the letter p. These two functions make it possible to give a coalgebraic characterisation of the characteristic function of G. We have:

$$G(x)(\alpha) = \epsilon_\alpha(x) \ , \qquad G(x)(\alpha\, p\, u) = G(\delta_{\alpha p}(x))(u) \ .$$

The tuple $\langle \mathcal{R}eg(\Sigma \uplus B(A)), \delta, \epsilon \rangle$ can be seen as a deterministic automaton with input alphabet $At \times \Sigma$, and output set 2^{At}. Thanks to the above characterisation, a state x in this automaton accepts precisely the guarded string language $G(x)$—modulo the isomorphism $(At \times \Sigma)^\star \to 2^{At} \approx \mathcal{P}((At \times \Sigma)^\star \times At)$.

However, we cannot directly apply the explicit algorithm from Section 2.1, because this automaton is not finite. First, there are infinitely many KAT expressions, so that we have to restrict to those that are accessible from the expressions we want to check for equality. This is however not sufficient: we also have to quotient regular expressions w.r.t. a few simple laws [28]. This quotient is simple to implement by normalising expressions; we thus assume that expressions are normalised in the remainder of this section.

Symbolic derivatives. The input alphabet of the above automaton is exponentially large w.r.t. the number of primitive tests: $At \times \Sigma = 2^A \times \Sigma$. Therefore, the simple algorithm from Section 2.1 is not tractable in practice. Instead, we would like to use its symbolic version (Figure 5).

The output values, in $(2^{At} = 2^A \to 2)$, are also exponentially large, and are best represented symbolically, using Boolean BDDs. In fact, any test appearing in a KAT expression can be pre-compiled into a Boolean BDD: rather than working with regular expressions over $\Sigma \uplus B(A)$ we thus move to regular expressions over $\Sigma \uplus \mathrm{BDD}_A[2]$, which we call *symbolic KAT expressions*. We denote the set of such expressions by SyKAT, and we let $(\!|e|\!)$ denote the symbolic version of a KAT expression e.

Note that there is a slight discrepancy here w.r.t. Section 3: the input alphabet is $2^A \times \Sigma$ rather than just $2^{A'}$ for some A'. For the sake of simplicity, we just assume that Σ is actually of the shape $2^{\Sigma'}$; alternatively, we could work with automata whose transition functions are represented partly symbolically (for At), and partly explicitly (for Σ)—this is what we do in the implementation.

We define the symbolic derivation operations in Figure 9.

The output function, ϵ^{s}, has type $\mathrm{SyKAT} \to \mathrm{BDD}_A[2]$, it maps symbolic KAT expressions to Boolean BDD nodes. The operations used on the right-hand side of this definition are those on Boolean BDDs. The function ϵ^{s} is much more efficient than its explicit counterpart (ϵ, in Figure 8): the set of all accepted atoms is computed at once, symbolically.

The function δ^{s} has type $\mathrm{SyKAT} \to \mathrm{BDD}_{A \uplus \Sigma'}[\mathrm{SyKAT}]$. It maps symbolic KAT expressions to BDDs whose leaves are themselves symbolic KAT expressions. Again, in contrast to its explicit counterpart, δ^{s} computes all the transitions of a given expression once and for all. The operations used on the right-hand side of the definition are the following ones:

- $n \oplus m$ is defined by applying pointwise the syntactic sum operation from KAT expressions to the two BDDs n and m: $n \oplus m = \lfloor \phi \mapsto \lceil n \rceil(\phi) + \lceil m \rceil(\phi) \rfloor$;
- $n \odot x$ syntactically multiplies all leaves of the BDD n by the expression x, from the right: $n \odot x = \lfloor \phi \mapsto \lfloor n \rfloor(\phi) \cdot x \rfloor$;
- $f \otimes n$ "multiplies" the Boolean BDD f with the BDD n: $f \otimes n = \lfloor \phi \mapsto \lfloor n \rfloor(\phi)$ if $\lfloor f \rfloor(\phi) = 1, 0$ otherwise$\rfloor$.
- $\lfloor q \mapsto 1, \_ \mapsto 0 \rfloor$ is the BDD mapping q to 1 and everything else to 0 ($q \in \Sigma = 2^{\Sigma'}$ being cast into an element of $2^{A \uplus \Sigma'}$).

By two simple inductions, one proves that for every expression $x \in \mathrm{SyKAT}$, atom $\alpha \in At$, and letter $p \in \Sigma$, we have:

$$\lceil \epsilon^{\mathrm{s}}(\!|x|\!) \rceil(\alpha) = \epsilon_\alpha(x)$$
$$\lceil \delta^{\mathrm{s}}(\!|x|\!) \rceil(\alpha p) = (\!|\delta_{\alpha p}(x)|\!)$$

(Again, we abuse notation by letting the pair αp denote an element of $2^{A \uplus \Sigma'}$.) This ensures that the symbolic deterministic automaton $\langle \mathrm{SyKAT}, \delta^{\mathrm{s}}, \epsilon^{\mathrm{s}} \rangle$ faithfully represents the previous explicit automaton, and that we can use the symbolic algorithms from Section 3.

4.2 Partial derivatives

An alternative to Brzozowski's derivatives consists in using Antimirov' *partial derivatives* [4], which generalise to KAT in a straightforward way [34]. The difference with Brzozowski's derivative is that they produce a non-deterministic automaton: states are still expressions, but the derivation function produces a set of expressions. An advantage is that we do not need to normalise expressions on the fly: the set of partial derivatives reachable from an expression is always finite.

We give directly the symbolic definition, which is very similar to the previous one.

$$\begin{aligned}
\delta'^{\mathrm{s}}(x+y) &= \delta'^{\mathrm{s}}(x) \sqcup \delta'^{\mathrm{s}}(y) \\
\delta'^{\mathrm{s}}(x \cdot y) &= (\delta'^{\mathrm{s}}(x) \boxdot y) \sqcup (\epsilon^{\mathrm{s}}(x) \boxtimes \delta'^{\mathrm{s}}(y)) \\
\delta'^{\mathrm{s}}(x^\star) &= \delta'^{\mathrm{s}}(x) \boxdot x^\star \\
\delta'^{\mathrm{s}}(p) &= \lfloor p \mapsto \{1\}, _ \mapsto \emptyset \rfloor \\
\delta'^{\mathrm{s}}(\phi) &= \emptyset
\end{aligned}$$

The differences lie in the BDD operations, whose leaves are now sets of expressions:

- $n \sqcup m = \lfloor \phi \mapsto \lceil n \rceil(\phi) \cup \lceil m \rceil(\phi) \rfloor$;
- $n \boxdot x = \lfloor \phi \mapsto \{x' \cdot x \mid x' \in \lceil n \rceil(\phi)\} \rfloor$;
- $f \boxtimes n = \lfloor \phi \mapsto \lceil n \rceil(\phi)$ if $\lceil f \rceil(\phi) = 1, \emptyset$ otherwise$\rfloor$.

One can finally relate partial derivatives to Brzozowski's one:

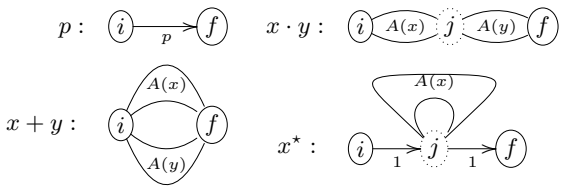
$$\mathrm{KA} \vdash \sum \lceil \delta'^{\mathrm{s}}(x) \rceil(\alpha p) = (\!|\delta_{\alpha p}(x)|\!) \ .$$

(The above Σ denotes the iterated sum of the set of partial derivatives—we do not have a syntactic equality because partial derivatives inherently exploit the fact that multiplication distributes over sums.) Using symbolic determinisation as described in Section 3.2, one can thus use the algorithm from Section 3 with Antimirov' partial derivatives.

4.3 Ilie & Yu's construction

Other automata constructions from the literature can be generalised to KAT expressions. We can for instance consider Ilie and Yu's construction [25], which produces non-deterministic automata with epsilon transitions with exactly one initial state, and one accepting state.

We consider a slightly simplified version here, where we omit a few optimisations and just proceed by induction on the expression. The four cases are depicted below: i and f are the initial and accepting states, respectively; in the concatenation and star cases, a new state j is introduced.



To adapt this construction to KAT expressions, it suffices to generalise epsilon transitions to transitions labelled by tests. In the base case for a test ϕ, we just add a transition labelled by ϕ between i and f; the two epsilon transitions needed for the star case just become transitions labelled by the constant test 1.

As expected, when starting from a symbolic KAT expression, those counterparts to epsilon transitions are labelled by Boolean BDD nodes rather than by explicit Boolean expressions.

Epsilon cycles. The most important optimisation we miss with this simplified presentation of Ilie and Yu's construction is that we should merge states that belong to cycles of epsilon transitions. An alternative to this optimisation consists in normalising first the expressions so that for all subexpressions of the shape $e^\star$, e does not contain 1, i.e., $\epsilon^{\mathrm{s}}(e) \neq 1$. Such a normalisation procedure has been proposed for plain regular expressions by Brüggemann-Klein [8]. When working with such normal forms, the automata produced by the above simplified construction (on plain regular expressions) have acyclic epsilon transitions, so that the aforementioned optimisation is unnecessary.

This normalisation procedure generalises easily to (symbolic) KAT expressions. For instance, here are typical normalisations:

$$(\phi + p)^\star \mapsto p^\star \tag{1}$$
$$(p^\star + q)^\star \mapsto (p + q)^\star \tag{2}$$
$$((1 + p)(1 + q))^\star \mapsto (p + q)^\star \tag{3}$$

We say that Symbolic KAT expressions satisfying the above property are in *strict star form*. The normalisation procedure is linear in the size of the expressions; it always produces a smaller expression. As a consequence, when deciding whether a KAT equation holds or not, it is always beneficial to put the expressions in strict star form first, independently from the considered automata construction. (See the experiments in Section 5).

According to the example (1), it might be tempting to strengthen example (3) into $((\phi + p)(\psi + q))^\star \mapsto (p + q)^\star$. Such a step is invalid, unfortunately. (The second expression accepts the guarded string $\alpha p \beta$ for all α, β, while the starting expression needs $\beta \vDash \psi$.) This example seems to show that one cannot ensure that all starred

	iterations		time		NFA states	DFA states
	`symb_equiv`	`dsf_equiv`	`symb_equiv`	`dsf_equiv`		
Antimirov ∘ ssf	6 715	3 980	0.53s	0.47s	2 704	4 142
Antimirov	7 141	4 256	0.84s	0.74s	3 039	4 442
Ilie & Yu ∘ ssf	6 985	4 209	1.77s	1.73s	4 716	4 441
Ilie & Yu	7 328	4 445	3.89s	3.83s	5 730	4 647
Brzozowski ∘ ssf	11 952	6 525	6.88s	4.67s	-	6 684
Brzozowski	19 781	10 080	43.00s	30.00s	-	10 265

Table 1. Checking random saturated pairs of expressions.

subexpressions are mapped to 0 by ϵ^{s}. As a consequence we cannot assume that test-labelled transitions generated by Ilie and Yu's construction form an acyclic graph in general.

4.4 Test-labelled transitions removal

The above construction produces symbolic NFA with test-labelled transitions, which have to be eliminated in order to apply the algorithms from Section 3. Other constructions from the literature produce automata with epsilon transitions and can be adapted to KAT using test-labelled transitions. A generic procedure for eliminating such transitions is thus desirable.

The usual technique with plain automata consists in computing the reflexive transitive closure of epsilon transitions, to precompose the other transitions with the resulting relation, and declare a state as accepting in the new automaton whenever it can reach an accepting state through this reflexive-transitive closure.

More formally, let us recall Kozen's matricial representation of non-deterministic automaton with epsilon transitions [26], as tuples $\langle n, u, J, N, v\rangle$, where u is a $(1, n)$ 01-matrix denoting the initial states, J is a (n, n) 01-valued matrix denoting the epsilon transitions, N is a (n, n) matrix representing the other transitions (with entries sets of letters in Σ), and v is a $(n, 1)$ 01-matrix encoding the accepting states.

The language accepted by such an automaton can be represented by the following matricial product, using Kleene star on matrices:

$$u \cdot (J + N)^\star \cdot v \ .$$

Thanks to the algebraic law $(a+b)^\star = a^\star \cdot (b \cdot a^\star)^\star$, which is valid in any Kleene algebra, we get

$$\mathrm{KA} \vdash u \cdot (J + N)^\star \cdot v = u \cdot (J^\star N)^\star \cdot (J^\star v) \ .$$

We finally check that $\langle n, u, 0, J^\star N, J^\star v\rangle$ represents a non-deterministic automaton without epsilon transitions. This is how Kozen validates epsilon elimination for plain automata, algebraically [26].

The same can be done here for KAT by noticing that tests (or Boolean BDD nodes) form a Kleene algebra with a degenerate star operation: the constant-to-1 function. One can thus generalise the above reasoning to the case where J is a tests-valued matrix rather than a 01-matrix.

The iteration $J^\star$ of such a matrix can be computed using standard shortest-path algorithms [21], on top of the efficient semiring of Boolean BDD nodes. The resulting automaton has the expected type:

- there is a transition labelled by αp between i and j if there exists a k such that $\alpha \vDash (J^\star)_{i,k}$ and $p \in N_{k,j}$. (The corresponding non-deterministic symbolic transition function can be computed efficiently using appropriate BDD functions.)
- The output value of a state i is the Boolean BDD node obtained by taking the disjunction of all the $(J^\star)_{i,j}$ such that j is an accepting state (i.e., just $(J^\star)_{(i,f)}$ when using Ilie and Yu's construction).

5. Experiments

We implemented all presented algorithms in OCaml; the corresponding library is available online, together with an applet allowing to trace them on user-provided examples [35].

Symbolic KAT expressions are hash-consed, which allows us to represent sets of expressions using Patricia trees (e.g., for Antimirov' partial derivates). Expressions are also normalised using smart constructors: sums associated to the left, sorted, and without duplicates; products are associated to the left; consecutive tests are merged; units are cancelled as much as possible. For Antimirov' construction and for Ilie and Yu's construction, the produced symbolic NFA are memoised once and for all, and reindexed so that their states are just natural numbers. This allows us to use bit-vectors to represent sets produced during determinisation. The queue `todo` used for storing the pairs to process is a FIFO queue, so that the automata are explored in a breadth-first manner.

We performed a few experiments to compare the presented algorithms and constructions. We generated random KAT expressions over two sets of seven primitive tests and seven atomic elements, with seventy connectives, and excluding explicit occurrences of the constants 0 and 1. A hundred pairs of random expressions were checked for equality after being saturated by adding the constant $\Sigma^\star$ on both sides. (A difficulty here is that random pairs of expressions are almost always distinguished by a very short guarded string, which is found almost immediately thanks to the breadth-first strategy, independently from the size of the expressions and from the up-to techniques at work. Instead, we would like to evaluate the algorithms based on their running time on more interesting pairs, where the expressions are either equivalent or distinguished only by long guarded strings. By saturating the expressions with the constant $\Sigma^\star$, we artificially make the expressions equivalent. Moreover, looking at an execution of the presented algorithms on such saturated pairs, what happens is that the output test (line 10 on Figure 1) always succeeds, so that the algorithms stop only once the whole automata have been explored and a bisimulation has been found. Moreover, an analysis of the various automata constructions shows that the automata constructed for an expression p are very similar to the automata constructed for the expression $p + \Sigma^\star$: exploring the latter is as hard as exploring the former.)

The results are displayed in Table 1: for each construction and for each of the two symbolic algorithms, we give the total number of iterations (i.e., the number of times we execute line 10 in Figure 5), and the global running time[2]. Each construction is associated to two lines, depending on whether we first put expressions in strict star form or not. We additionally provide the total number of NFA states generated by Antimirov' and Ilie and Yu's constructions, as well as the total number of DFA states generated for the three constructions.

One can notice than Antimirov' partial derivatives provide the fastest algorithms. Ilie and Yu's construction yield approximately

[2] Theses experiments were performed on a MacBook Pro, OS X 10.9.5, 2,4GHz Intel Core i7, 4Go 1333MHz DDR3, OCaml 4.02.1

the same number of iterations as Antimirov' partial derivatives, but require more time: computing the transitive closure for epsilon removal is a costly operation. Brzozowski's construction gives poor results both in terms of time and iterations: the produced automata are larger, and more difficult to compute.

Concerning the equivalence algorithm, one notices that using disjoint set forests significantly reduces the number of iterations. There is almost no difference in the running times with the first two constructions, because most of the time is spent in constructing the automata rather than checking them for equivalence. This is no longer true with Brzozowski's construction, for which the automata are sufficiently big to observe a difference.

6. Directions for future work

The equational theory of KAT is PSPACE-complete, but none of the presented algorithms are PSPACE (just because of the use BDDs, but also because the bisimulation candidate, which has to be stored, can be exponentially large). Experiments however suggest that they can be useful in practice: the symbolic DFA produced by the various constructions proposed in this paper tend to be of reasonable size. Quantifying this empirical observation in a formal way seems extremely difficult.

A natural extension of this work would be to apply the proposed algorithms to KAT+B! [19] and NetKAT [2], two extensions of KAT with important applications in verification: while programs with mutable tests in the former case, and network programming in the later case.

KAT+B! has a EXPSPACE-complete equational theory, and its structure makes explicit algorithms completely useless. Designing symbolic algorithms for KAT+B! seems challenging.

NetKAT remains PSPACE-complete, and Foster et al. propose in the present volume a coalgebraic decision procedure relying on a variation of Antimirov' derivatives [17]. To get a practical algorithm, they represent automata transitions using sparse matrices, and they exploit some form of symbolic treatment by using what they call "bases". KAT can be encoded into NetKAT, so that their algorithm could be used for KAT. This encoding is however not streamlined, and it is non-trivial to understand the behaviour of their algorithm on the resulting instances. Conversely, adapting the algorithms presented in the present paper to cope with NetKAT seems feasible, although not straightforward. Concerning the symbolic treatment of automata, our use of BDDs seems more powerful and less ad-hoc than their use of bases, but the precise relationship remains unclear, and we leave its formal analysis for future work.

Moving away from KAT specificities, we leave open the question of the complexity of our symbolic variant of Hopcroft and Karp's algorithm (Figure 7). Tarjan proved that their algorithm is almost linear in amortised time complexity, and he made a list of heuristics for linking and path compression schemes that lead to that complexity [40]; together with Goel, Khanna and Larkin, he recently showed that this complexity is still reached (asymptotically) with randomized linking [18]. A similar study for the symbolic counterpart we propose here remains to be done.

Acknowledgments

We are grateful to the anonymous referees who provided thorough and detailed reviews, and in particular to the one who noticed the relationship between the present work and Rémy's type inference algorithm for row types.

References

[1] P. A. Abdulla, Y.-F. Chen, L. Holík, R. Mayr, and T. Vojnar. When simulation meets antichains. In *Proc. TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 158–174. Springer Verlag, 2010.

[2] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker. Netkat: semantic foundations for networks. In *Proc. POPL*, pages 113–126. ACM, 2014.

[3] A. Angus and D. Kozen. Kleene algebra with tests and program schematology. Technical Report TR2001-1844, CS Dpt., Cornell University, July 2001.

[4] V. M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theoretical Computer Science*, 155(2):291–319, 1996.

[5] S. L. Bloom, Z. Ésik, and G. Stefanescu. Notes on equational theories of relations. *Algebra Universalis*, 33(1):98–126, 1995.

[6] F. Bonchi and D. Pous. Checking NFA equivalence with bisimulations up to congruence. In *Proc. POPL*, pages 457–468. ACM, 2013.

[7] A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *Proc. CAV*, volume 3114 of *Lecture Notes in Computer Science*, pages 372–386. Springer Verlag, 2004.

[8] A. Brüggemann-Klein. Regular expressions into finite automata. *Theoretical Computer Science*, 120(2):197–213, 1993.

[9] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[10] R. E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[11] J. A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494, 1964.

[12] E. Cohen, D. Kozen, and F. Smith. The complexity of Kleene algebra with tests. Technical Report TR96-1598, CS Dpt., Cornell University, 1996.

[13] L. D'Antoni and M. Veanes. Minimization of symbolic automata. In *POPL*, pages 541–554. ACM, 2014.

[14] L. Doyen and J.-F. Raskin. Antichain Algorithms for Finite Automata. In *Proc. TACAS*, volume 6015 of *Lecture Notes in Computer Science*. Springer Verlag, 2010.

[15] Z. Ésik and L. Bernátsky. Equational properties of Kleene algebras of relations with conversion. *Theoretical Computer Science*, 137(2):237–251, 1995.

[16] J.-C. Filliâtre and S. Conchon. Type-safe modular hash-consing. In *ML*, pages 12–19. ACM, 2006.

[17] N. Foster, D. Kozen, M. Milano, A. Silva, and L. Thompson. A coalgebraic decision procedure for NetKAT. In *Proc. POPL*. ACM, 2015.

[18] A. Goel, S. Khanna, D. Larkin, and R. E. Tarjan. Disjoint set union with randomized linking. In *Proc. SODA*, pages 1005–1017. SIAM, 2014.

[19] N. B. B. Grathwohl, D. Kozen, and K. Mamouras. KAT + B! In *Proc. CSL-LICS*. ACM, July 2014.

[20] J. G. Henriksen, J. L. Jensen, M. E. Jørgensen, N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *TACAS*, volume 1019 of *Lecture Notes in Computer Science*, pages 89–110. Springer Verlag, 1995.

[21] P. Höfner and B. Möller. Dijkstra, Floyd and Warshall meet Kleene. *Formal Aspects of Computing*, 24(4-6):459–476, 2012.

[22] J. E. Hopcroft. An $n\log n$ algorithm for minimizing states in a finite automaton. Technical report, Stanford University, 1971.

[23] J. E. Hopcroft and R. M. Karp. A linear algorithm for testing equivalence of finite automata. Technical Report 114, Cornell University, December 1971.

[24] G. Huet. *Résolution d'équations dans les langages d'ordre 1,2, ... ,ω*. PhD thesis, Université Paris VII, 1976. Thèse d'État.

[25] L. Ilie and S. Yu. Follow automata. *Information and Computation*, 186(1):140–162, 2003.

[26] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994.

[27] D. Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997.

[28] D. Kozen. On the coalgebraic theory of Kleene algebra with tests. Technical report, CIS, Cornell University, March 2008.

[29] D. Kozen and M.-C. Patron. Certification of compiler optimizations using Kleene algebra with tests. In *Proc. CL2000*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 568–582. Springer Verlag, 2000.

[30] D. Kozen and F. Smith. Kleene algebra with tests: Completeness and decidability. In *Proc. CSL*, volume 1258 of *Lecture Notes in Computer Science*, pages 244–259. Springer Verlag, September 1996.

[31] E. F. Moore. Gedanken-experiments on sequential machines. *Automata Studies, Annals of Mathematical Studies*, 34:129–153, 1956.

[32] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[33] F. Pottier and D. Rémy. *Advanced Topics in Types and Programming Languages*, chapter The Essence of ML Type Inference. MIT Press, 2004.

[34] D. Pous. Kleene Algebra with Tests and Coq tools for while programs. In *Proc. ITP*, volume 7998 of *Lecture Notes in Computer Science*, pages 180–196. Springer Verlag, 2013.

[35] D. Pous. Web appendix to this paper, with Ocaml implementation of the proposed algorithms, 2014. `http://perso.ens-lyon.fr/damien.pous/symbolickat`.

[36] D. Pous and D. Sangiorgi. *Advanced Topics in Bisimulation and Coinduction*, chapter about "Enhancements of the coinductive proof method". Cambridge University Press, 2011.

[37] D. Rémy. *Algèbres Touffues. Application au Typage Polymorphe des Objets Enregistrements dans les Langages Fonctionnels*. PhD thesis, Université Paris VII, 1990. Thèse de doctorat.

[38] D. Rémy. Extension of ML type system with a sorted equational theory on types, 1992. Research Report 1766.

[39] D. Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8:447–479, 1998.

[40] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.

[41] M. Veanes. Applications of symbolic finite automata. In *CIAA*, volume 7982 of *Lecture Notes in Computer Science*, pages 16–23. Springer Verlag, 2013.

[42] M. D. Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *Proc. CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 17–30. Springer Verlag, 2006.