Università di Pisa

Dipartimento di Informatica

Technical Report: TR-06-07

Process Bisimulation *via* a Graphical Encoding

Filippo Bonchi[1] Fabio Gadducci[1] Barbara König[2]

July 26, 2006

ADDRESS: via F. Buonarroti 2, 56127 Pisa, Italy. TEL: +39 050 2212700 FAX: +39 050 2212726

Process Bisimulation *via* a Graphical Encoding[⋆]

Filippo Bonchi[1], Fabio Gadducci[1], and Barbara König[2]

Dipartimento di Informatica, Università di Pisa Interaktive Systeme, Universität Duisburg-Essen

Abstract. The paper presents a case study on the synthesis of labelled transition systems (LTSs) for process calculi, choosing as testbed Milner's Calculus of Communicating System (CCS). The proposal is based on a graphical encoding: each CCS process is mapped into a graph equipped with suitable *interfaces*, such that the denotation is fully abstract with respect to the usual structural congruence.
Graphs with interfaces are amenable to the synthesis mechanism based on *borrowed contexts* (BCs), proposed by Ehrig and König (which are an instance of *relative pushouts*, originally introduced by Milner and Leifer). The BC mechanism allows the effective construction of an LTS that has graphs with interfaces as both states and labels, and such that the associated bisimilarity is automatically a congruence.
Our paper focuses on the analysis of the LTS distilled by exploiting the encoding of CCS processes: besides offering some technical contributions towards the simplification of the BC mechanism, the key result of our work is the proof that the bisimilarity on processes obtained via BCs coincides with the standard strong bisimilarity for CCS.

1 Introduction

The dynamics of a computational device is often defined by a *reduction system* (RS): a set, representing the space of possible states of the device; and a relation among these states, representing the possible evolutions of the device. This is e.g. the case of the paradigmatic functional language, the λ-calculus: the *β-reduction rule* $(\lambda x.M)N \Rightarrow M[N/x]$ models the application of a functional process $\lambda x.M$ to the actual argument N, and the reduction relation is then obtained by freely instantiating and contextualising the rule.

While RSs have the advantage of conveying the semantics with relatively few compact rules, their main drawback is poor compositionality, in the sense that the dynamic behaviour of arbitrary standalone terms can be interpreted

⋆ Research partially supported by the DFG project SANDS, the EU RTN 2-2001-00346 SegraVis (*Syntactic and Semantic Integration of Visual Modelling Techniques*) and IST 2004-16004 SEnSOria (*Software Engineering for Service-Oriented Overlay Computers*), the MIUR PRIN 2005015824 ART (*Analysing Reduction systems using Transition Systems*) and the CRUI/DAAD Programma Vigoni project "Models based on Graph Transformation Systems: Analysis and Verification".

only by inserting them in the appropriate context, where a reduction may take place. In fact, simply using the reduction relation for defining equivalences between components fails to obtain a compositional framework, and in order to recover a suitable congruence it is often necessary to verify the behaviour of single components under any viable execution context. This is the road leading from contextual equivalences for the λ-calculus to barbed and dynamic equivalences for the π-calculus. In these approaches, though, proofs of equivalence are often tedious and involuted, and they are left to the ingenuity of the researcher.

A standard way out of the impasse, reducing the complexity of such analyses, is to express the behaviour of a computational device by a *labelled transition system* (LTS). Should the label associated to a component evolution faithfully express how that component might interact with the whole of the system, it would be possible to analyse *in vitro* the behaviour of a single component, without considering all contexts. Thus, a "well-behaved" LTS represents a fundamental step towards a compositional semantics of the computational device. It is not always straightforward, though, to identify the right "label" that should be distilled, starting from a previously defined RS. Indeed, after Milner's proposal of an alternative semantics for the π-calculus [22] based on reactive rules modulo a structural congruence on processes, inspired by the CHAM paradigm [4], an ongoing stream of research has been investigating the relationship between the LTS semantics for process calculi and their more abstract RS semantics.

Early attempts by Sewell [27] devised a strategy for obtaining an LTS from an RS by adding contexts as labels on transitions. The technique was refined by Leifer and Milner [20] who introduced *relative pushouts* (RPOs) in order to capture the notion of *minimal context* activating a reduction. The generality of this proposal (and its bicategorical formulation due to Sassone and Sobocinski [25]) allows it to be applied to a large class of formalisms. More importantly, such attempts share the basic property of synthesising a congruent bisimulation equivalence, thus ensuring that the resulting LTS semantics is compositional. However, for the time being there are few case studies which either involve rich calculi, or succeed in making comparisons with standard behavioural equivalences. To tackle a fully-fledged case study is the main aim of this paper.

Our starting point for the synthesis of an LTS are the graphical techniques proposed for modelling the reduction semantics of nominal calculi in [13, 16]: processes are encoded in *graphs with interfaces*, an instance of *cospan categories* [14], and process reduction is simulated by *double-pushout* (DPO) rewriting [1]. Since the category of cospans over graphs admits RPOs [26], its choice as the domain of the encoding for nominal calculi ensures that the synthesis of an LTS can be performed, and that a *compositional* observational equivalence is obtained.

The key technical point is the use of the *borrowed context* (BC) technique [11] as a tool to equip graph transformation in the DPO style with an LTS semantics. Graphs with interfaces are amenable to the synthesis mechanism based on BCs (which are in turn an instance of RPOs): this allows the construction of an LTS that has graphs with interfaces as both states and labels, and such that the associated bisimilarity is automatically a congruence. Exploiting the

BC technique, also large case studies may be taken into account: until now the difficulties in the presentation of the LTSs obtained via the use of RPOs forced to restrict the analysis to simple case studies, relying either on standard (ground) term rewriting [20], or on extremely simplified variants of process calculi [25]: more elaborated proposals using bigraphs [23, 18] result in infinitely branching LTSs, banning recursive processes or failing to capture standard bisimilarity.

Summing up, the aim of our work is straightforward: to present a fully-fledged case study on the synthesis of LTSs for process calculi, choosing as testbed Milner's Calculus of Communicating System (CCS). More precisely, the paper focuses on the analysis of the LTS obtained by exploiting the BC technique and the encoding of CCS (recursive) processes into unstructured graphs, along the lines of the methodology sketched above. Besides offering some technical contributions towards the simplification of the BC synthesis mechanism, the key result is the proof that the bisimilarity on (recursive) processes obtained via BCs coincides with the standard strong bisimilarity for CCS. We believe that our work may offer novel insights on the synthesis of LTSs, as well as offering further evidence of the adequacy of graph-based formalisms for system design and verification.

The structure of the paper follows. Section 2 recalls the syntax as well as the RS and the LTS semantics of Milner's CCS. Section 3 introduces graphs with interfaces and Section 4 illustrates the encoding of (recursive) processes in them. Then Section 5 introduces DPO rewriting on graphs with interfaces as well as the BC technique for distilling an LTS. A graph rewriting system for CCS that is able to simulate process reduction is defined in Section 6. Finally, Section 7 presents our use of the graphical encoding for providing an alternative LTS semantics for CCS, by means of the BC synthesis mechanism: the induced bisimulation on (encodings of recursive) processes is proved to coincide with the standard CCS strong bisimulation. The final section outlines future research avenues, while the appendices contain the proofs and most of the categorical notions used in the paper.

This paper is an extended version of [5].

2 Two Operational Semantics for CCS

This section introduces CCS [21] and two alternative operational semantics: the classical LTS semantics and the reduction semantics.

Definition 1 (processes). *Let $\mathcal{N}$ be a set of* names *(ranged over by $a, b, c, \ldots$); $\tau \notin \mathcal{N}$ an* invisible *name; $\Delta = \{a, \overline{a} \mid a \in \mathcal{N}\} \uplus \{\tau\}$ a set of prefixes (ranged over by δ); and finally, X a set of* agent variables *(ranged over by $x, y, w \ldots$). An* open process *P is a term generated by the (mutually recursive) syntax*

$$P ::= M,\ (\nu a)P,\ P_1 \mid P_2,\ rec_x.P \quad M ::= \mathtt{0},\ \delta.P,\ M_1 + M_2,\ \delta.x$$

A process *is a term such that each occurrence of an agent variable x is in the scope of a rec_x-operator. We let $P, Q, R, \ldots$ range over the set $\mathcal{P}$ of processes, and $M, N, O \ldots$ range over the set $\mathcal{S}$ of* summations.

The standard definition for the set of free names of a process P, denoted by $\mathtt{fn}(P)$, is assumed. Similarly for α-conversion with respect to the *restriction* operators $(\nu a)P$: the name a is bound in P, and it can be freely α-converted.

The classical observational semantics, *bisimilarity*, is given over an inductively defined *labelled transition system* (LTS). We spell out the LTS, and denote by $\sim_{CCS}$ the standard strong bisimilarity, without formally introducing it.

Definition 2 (labelled transition system). *The* transition relation *for processes is the relation $L_{CCS} \subseteq \mathcal{P} \times \Delta \times \mathcal{P}$ inductively generated by the set of axioms and inference rules below (where $P \xrightarrow{\delta} Q$ means that $\langle P, \delta, Q\rangle \in L_{CCS}$).*

$$\frac{}{\delta.P \xrightarrow{\delta} P} \qquad \frac{P \xrightarrow{a} Q,\ R \xrightarrow{\overline{a}} S}{P \mid R \xrightarrow{\tau} Q \mid S} \qquad \frac{P \xrightarrow{\delta} Q}{(\nu a)P \xrightarrow{\delta} (\nu a)Q}\ \ a \notin \mathtt{fn}(\delta.\mathtt{0})$$

$$\frac{P \xrightarrow{\delta} Q}{P \mid R \xrightarrow{\delta} Q \mid R} \qquad \frac{P \xrightarrow{\delta} Q}{P + R \xrightarrow{\delta} Q} \qquad \frac{P\,[^{rec_x.P}/_x] \xrightarrow{\delta} Q}{rec_x.P \xrightarrow{\delta} Q}$$

As usual, we avoided presenting the symmetric counterparts of those three inference rules involving the parallel and sum operators; moreover, the substitution operator is supposed not to capture any name, possibly through α-conversion.

The behavior of a process P can also be described as a relation over *abstract processes*, obtained by closing a set of basic rules under structural congruence.

Definition 3 (structural congruence). *The* structural congruence *for processes is the relation $\equiv\ \subseteq \mathcal{P} \times \mathcal{P}$, closed under process construction and α-conversion, inductively generated by the set of axioms below.*

$$P \mid Q = Q \mid P \qquad P \mid (Q \mid R) = (P \mid Q) \mid R \qquad P \mid \mathtt{0} = P$$

$$M + N = N + M \qquad M + (N + O) = (M + N) + O \qquad M + \mathtt{0} = M$$

$$(\nu a)(\nu b)P = (\nu b)(\nu a)P \qquad (\nu a)(P \mid Q) = P \mid (\nu a)Q \ \ \text{for } a \notin \mathtt{fn}(P) \qquad (\nu a)\mathtt{0} = \mathtt{0}$$

$$(\nu a)(M + \delta.P) = M + \delta.(\nu a)P \ \ \text{for } a \notin \mathtt{fn}(M + \delta.\mathtt{0}) \qquad rec_x.P = P[^{rec_x.P}/_x]$$

Definition 4 (reduction semantics). *The* reduction relation *for processes is the relation $R_{CCS} \subseteq \mathcal{P}\times\mathcal{P}$, closed under the structural congruence $\equiv$, inductively generated by the set of axioms and inference rules below (where $P \rightarrow Q$ means that $\langle P, Q\rangle \in R_{CCS}$).*

$$\frac{}{a.P + M \mid \overline{a}.Q + N \rightarrow P \mid Q} \qquad \frac{}{\tau.P + M \rightarrow P}$$

$$\frac{P \rightarrow Q}{(\nu a)P \rightarrow (\nu a)Q} \qquad \frac{P \rightarrow Q}{P \mid R \rightarrow Q \mid R}$$

There is a main difference with respect to the standard reduction semantics for CCS, namely, the axiom schema concerning the distributivity of the restriction operators with respect to the prefix operators, even if they have been already considered in the literature, see e.g. [12]. These equalities do not change substantially the reduction semantics, and they indeed hold in all the observational equivalences we are aware of. In particular, two congruent processes are also strongly bisimilar. Most importantly, they allow a simplified presentation of the graphical encoding: we refer the reader to [16] for a more articulate analysis.

The LTS semantics specifies how a system, seen as a single component, may interact with the environment, and it allows the definition of an observational equivalence by means of bisimilarity. On the other hand, the RS semantics specifies how a system, seen as a whole, evolves. The latter is usually more natural, but it does not take in account the interactions, and consequently, does not provide any "good" notion of behavioral equivalence. The main aim of the theory of reactive systems proposed by Milner in [20] is to systematically derive an LTS from an RS semantics. In this paper, exploiting a graphical encoding of processes, we derive an LTS from a graph rewriting semantics. More precisely, in the next sections we introduce a graphical encoding of CCS processes which preserves the reduction semantics. The encoding is then used to distill an LTS with pairs of graph morphisms as labels: the main result of the paper states that the resulting bisimilarity coincides with the standard strong bisimilarity.

Example 1. We introduce now a very simple example, the process defined as $rec_x.(\nu a)(\overline{a}.x \mid a.\mathtt{0} + b.\mathtt{0})$, which seems to us well-suited for illustrating both the labelled and the reduction semantics of the calculus, as well as the graphical encoding of processes presented in the next sections. The sub-process on the left is ready to send via (a) channel (named) a, and the sub-process on the right to receive on the same channel. Thus, after an unfolding step for the recursion operator, a possible commitment of the process consists of a synchronization on a, and the resulting process is structurally congruent to the original one. Note that, due to restriction, only the synchronisation is available for the two processes on channel a. The sub-process on the right, though, is also able to perform a single receive action on channel b, resulting in the terminal state $\mathtt{0}$ for the labelled semantics.

3 Graphs and their Extension with Interfaces

We recall a few definitions concerning (typed hyper-)graphs, and their extension with *interfaces*, referring to [7] for a more detailed introduction.

Definition 5 (graphs). *A* (hyper-)graph *is a four-tuple $\langle V, E, s, t\rangle$ where V is the set of nodes, E is the set of edges and $s, t : E \to V^*$ are the source and target functions. An (hyper-)graph morphism is a pair of functions $\langle f_V, f_E\rangle$ preserving the source and target functions.*

The corresponding category is denoted by **Graph**. However, we often consider *typed graphs* [8], i.e., graphs labelled over a structure that is itself a graph.

Definition 6 (typed graphs). *Let T be a graph. A* typed graph *G over T is a graph $|G|$, together with a graph morphism $t_G : |G| \to T$. A* morphism *between*

T-typed graphs $f : G_1 \to G_2$ is a graph morphism $f : |G_1| \to |G_2|$ consistent with the typing, i.e., such that $t_{G_1} = t_{G_2} \circ f$.

The category of graphs typed over T is denoted T-**Graph**: it coincides with the slice category **Graph** $\downarrow T$. In the following, a chosen type graph T is assumed.

In order to inductively define the encoding for processes, we need to provide operations over typed graphs. The first step is to equip them with suitable "handles" for interacting with an environment.

Definition 7 (graphs with interfaces). *Let J, K be typed graphs. A* graph with input interface J and output interface K *is a triple $\mathbb{G} = \langle j, G, k \rangle$, for G a typed graph and $j : J \to G$, $k : K \to G$ the* input *and* output *morphisms.*

Let $\mathbb{G}$ and $\mathbb{H}$ be graphs with the same interfaces. An interface graph morphism *$f : \mathbb{G} \Rightarrow \mathbb{H}$ is a typed graph morphism $f : G \to H$ between the underlying graphs that preserves the input and output morphisms.*

We let $J \xrightarrow{j} G \xleftarrow{k} K$ denote a graph with interfaces J and K.[1] If the interfaces J, K are *discrete*, i.e., they contain only nodes, we simply represent them by sets. Moreover, if K is the empty set, we often denote a graph with interfaces simply as a graph morphism $J \to G$. In order to define our encoding of processes, we introduce two binary operators on graphs with discrete interfaces.

Definition 8 (two composition operators). *Let $\mathbb{G} = I \xrightarrow{j} G \xleftarrow{k} K$ and $\mathbb{G}' = K \xrightarrow{j'} G' \xleftarrow{k'} J$ be graphs with discrete interfaces. Then, their* sequential composition *is the graph with discrete interfaces $\mathbb{G} \circ \mathbb{G}' = I \xrightarrow{j''} G'' \xleftarrow{k''} J$, for G'' the disjoint union $G \uplus G'$, modulo the equivalence on nodes induced by $k(x) = j'(x)$ for all $x \in N_{G'}$, and j'', k'' the uniquely induced arrows.*

Let $\mathbb{G} = J \xrightarrow{j} G \xleftarrow{k} K$ and $\mathbb{H} = J' \xrightarrow{j'} H \xleftarrow{k'} K'$ be graphs with discrete interfaces. Then, their parallel composition *is the graph with discrete interfaces $\mathbb{G} \otimes \mathbb{H} = (J \cup J') \xrightarrow{j''} V \xleftarrow{k''} (K \cup K')$, for V the disjoint union $G \uplus H$, modulo the equivalence on nodes induced by $j(x) = j'(x)$ for all $x \in N_J \cap N_{J'}$ and $k(y) = k'(y)$ for all $y \in N_K \cap N_{K'}$, and j'', k'' the uniquely induced arrows.*

Intuitively, the sequential composition $\mathbb{G} \circ \mathbb{G}'$ is obtained by taking the disjoint union of the graphs underlying $\mathbb{G}$ and $\mathbb{G}'$, and gluing the outputs of $\mathbb{G}$ with the corresponding inputs of $\mathbb{G}'$. Similarly, the parallel composition $\mathbb{G} \otimes \mathbb{H}$ is obtained by taking the disjoint union of the graphs underlying $\mathbb{G}$ and $\mathbb{H}$, and gluing the inputs (outputs) of $\mathbb{G}$ with the corresponding inputs (outputs) of $\mathbb{H}$. Note that the two operations are defined on "concrete" graphs, even if the result is independent of the choice of the representatives of the inner graphs, up to isomorphism.

[1] With an abuse of notation, we sometimes refer to the image of the input and output morphisms as inputs and outputs, respectively. More importantly, in the following we often refer implicitly to a graph with interfaces as the representative of its isomorphism class, still using the same symbols to denote it and its components.

A *graph expression* is a term over the syntax containing all graphs with discrete interfaces as constants, and parallel and sequential composition as binary operators. An expression is *well-formed* if all the occurrences of those operators are defined for the interfaces of their arguments, according to Definition 8; its interfaces are computed inductively from the interfaces of the graphs occurring in it, and its *value* is the graph obtained by evaluating all the operators in it.

4 From Processes to Graphs with Interfaces

This section presents our graphical encoding for CCS processes. After introducing a suitable type graph, shown in Fig. 1, the composition operators previously defined are exploited. This corresponds to a variant of the usual construction of the tree for a term of a free algebra: names are interpreted as variables, so that they are mapped to leaves of the graph and can be safely shared.

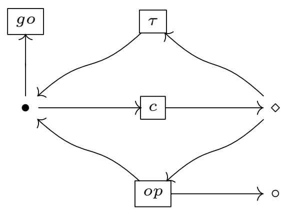


Fig. 1. The type graph T_{CCS} (for $op \in \{rcv, snd\}$).

Intuitively, a graph having as root a node of type $\bullet$ ($\diamond$) corresponds to a process (to a summation, respectively), while each node of type $\circ$ basically represents a name. Note that the edge *op* stands for a concise representation of two operators, namely *snd* and *rcv*, simulating the two prefixes. There is no operator for simulating either parallel composition or non-deterministic choice. Instead, the operator c is a syntactical device for "coercing" the occurrence of a summation inside a process context (a standard device from algebraic specifications). Finally, the operator *go* is another syntactical device for detecting the "entry" point of the computation, thus avoiding to perform any reduction below the outermost prefix operators: it is later needed for modeling the RS semantics.

The second step is the characterization of a class of graphs, such that all processes can be encoded into an expression containing only those graphs as constants, and parallel and sequential composition as binary operators. Let $p, s \notin \mathcal{N}$: our choice of graphs as constants is depicted in Fig. 2, for all $a \in \mathcal{N}$.

Finally, let us use id_Γ and $\mathtt{0}_\Gamma$ as a shorthand for $\bigotimes_{a \in \Gamma} id_a$ and $\bigotimes_{a \in \Gamma} \mathtt{0}_a$, respectively, for a finite set of names $\Gamma \subseteq \mathcal{N}$ (since the ordering is immaterial). The encoding of processes into graphs with interfaces, mapping each finite process into a graph expression, is presented below.

Definition 9 (encoding for finite processes). *Let P be a finite process, and let Γ be a set of names, such that $\mathtt{fn}(P) \subseteq \Gamma$. The (mutually recursive) encodings*

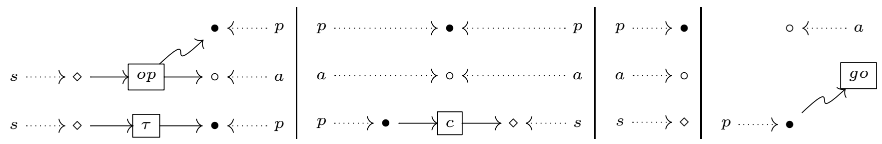


Fig. 2. Graphs op_a (for $op \in \{rcv, snd\}$) and τ; id_p, id_a, and c; 0_p, 0_a, and 0_s; ν_a and go (from left to right and top to bottom).

$[\![P]\!]^p_\Gamma$ and $[\![M]\!]^s_\Gamma$, mapping a process P into a graph with interfaces, are defined by structural induction according to the rules below.

$$
\begin{array}{ll}
[\![M]\!]^p_\Gamma & = \begin{cases} 0_p \otimes 0_\Gamma & \textit{if } \mathtt{fn}(M) = \emptyset \\ (c \otimes id_\Gamma) \circ [\![M]\!]^s_\Gamma & \textit{otherwise} \end{cases} \\
[\![(\nu a)P]\!]^p_\Gamma & = \begin{cases} [\![P]\!]^p_\Gamma & \textit{if } a \notin \mathtt{fn}(P) \\ (id_p \otimes \nu_b \otimes id_\Gamma) \circ [\![P\{^b/_a\}]\!]^p_{\{b\} \uplus \Gamma} \textit{ for } b \notin \Gamma & \textit{otherwise} \end{cases} \\
[\![P \mid Q]\!]^p_\Gamma & = [\![P]\!]^p_\Gamma \otimes [\![Q]\!]^p_\Gamma \\
[\![M + N]\!]^s_\Gamma & = [\![M]\!]^s_\Gamma \otimes [\![N]\!]^s_\Gamma \\
[\![0]\!]^s_\Gamma & = 0_s \otimes 0_\Gamma \\
[\![\tau.P]\!]^s_\Gamma & = (\tau \otimes id_\Gamma) \circ [\![P]\!]^p_\Gamma \\
[\![a.P]\!]^s_\Gamma & = (rcv_a \otimes id_\Gamma) \circ [\![P]\!]^p_\Gamma \\
[\![\overline{a}.P]\!]^s_\Gamma & = (snd_a \otimes id_\Gamma) \circ [\![P]\!]^p_\Gamma
\end{array}
$$

Note the conditional rule for the mapping of $[\![M]\!]^p_\Gamma$. This is required by the use of 0 as the neutral element for both the parallel and the non-deterministic operator: in fact, the syntactical requirement $\mathtt{fn}(M) = \emptyset$ coincides with the semantical constraint $M \equiv 0$.

The mapping is well-defined, since the resulting graph expression is well-formed; moreover, the encoding $[\![P]\!]^p_\Gamma$ is a graph with interfaces $(\{p\} \cup \Gamma, \emptyset)$. Our encoding is sound and complete (even if not surjective), as stated by the proposition below (adapted from [13]).

Proposition 1. *Let P, Q be finite processes, and let Γ be a set of names, such that $\mathtt{fn}(P) \cup \mathtt{fn}(Q) \subseteq \Gamma$. Then, $P \equiv Q$ if and only if $[\![P]\!]^p_\Gamma = [\![Q]\!]^p_\Gamma$.*

Note in particular how the lack of restriction operators is dealt with simply by manipulating the interfaces, even if the price to pay is the presence of "floating" axioms for prefixes, as shown by Fig. 3.

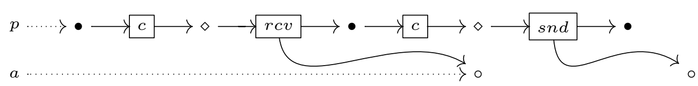


Fig. 3. Encoding for both $[\![(\nu b)a.\overline{b}.0]\!]^p_{\{a\}}$ and $[\![a.(\nu b)\overline{b}.0]\!]^p_{\{a\}}$.

4.1 Tackling recursive processes.

In order to show how recursive processes can be encoded as suitable infinite graphs, the first step is to consider a complete partial order on graphs.

Definition 10 (graph order). *Let* $\mathbb{G}$, $\mathbb{H}$ *be graphs with interfaces* (J, K). *Then,* $\mathbb{G} \sqsubseteq_{J,K} \mathbb{H}$ *if there exists a mono* $f : \mathbb{G} \Rightarrow \mathbb{H}$.

Thus, we consider the standard subgraph relationship, partitioned over interfaces. These partial orders are complete with respect to ω-chains, and it is noteworthy that the encoding $[\![0]\!]^p_\Gamma$ is the bottom of the order for those graphs with interfaces $(\{p\} \cup \Gamma, \emptyset)$.

Definition 11. *Let* $P[x]$ *be an open process, such that the single agent variable* x *may occur free in* P. *Let* $\mathcal{C} = \{[\![P_i]\!]^p_\Gamma \mid i \in \mathbb{N}\}$ *be a chain where* $P_0 = P[^0/_x]$ *and* $P_{i+1} = P[^{P_i}/_x]$. *Then,* $[\![rec_x.P]\!]^p_\Gamma$ *denotes the least upper bound of* $\mathcal{C}$.

In other terms, each open process $P[x]$ defines an ω-chain on the graphs with interfaces $(\{p\} \cup \Gamma, \emptyset)$, and $[\![rec_x.P]\!]^p_\Gamma$ is the least upper bound of this chain, computed as the least fixed point starting from the bottom element, i.e., $[\![0]\!]^p_\Gamma$.

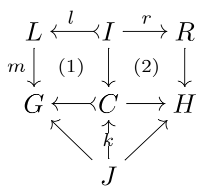
Of course, two recursive expressions may be mapped to isomorphic graphs with interfaces, even if they are not structurally congruent, nor can be unfolded to the same expression. Nevertheless, the extended encoding is clearly still sound.

5 On Graphs with Interfaces and Borrowed Contexts

This section introduces the *double-pushout* (DPO) approach to the rewriting of graphs with interfaces and its extension with *borrowed contexts* (BCs).

Definition 12 (graph production). *A* T*-typed* graph production *is a span* $L \overset{l}{\leftarrowtail} I \overset{r}{\longrightarrow} R$ *with* l *mono in* T-**Graph**. *A* typed *graph transformation system (*GTS*)* $\mathcal{G}$ *is a tuple* $\langle T, P, \pi \rangle$ *where* T *is the type graph,* P *is a set of production names and* π *is a function mapping each name to a* T*-typed production.*

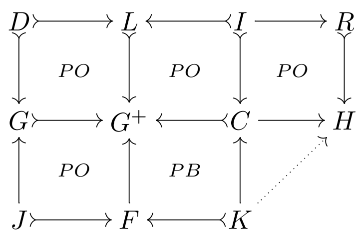
Definition 13 (derivation of graphs with interfaces). *Let* $J \to G$ *and* $J \to H$ *be two graphs with interfaces. Given a production* $p : L \overset{l}{\leftarrowtail} I \overset{r}{\longrightarrow} R$, *a* match *of* p *in* G *is a morphism* $m : L \to G$. *A* direct derivation *from* $J \to G$ *to* $J \to H$ *via* p *and* m *is a diagram as depicted in the right, where (1) and (2) are pushouts and the bottom triangles commute. In this case we write* $J \to G \Longrightarrow J \to H$.



The morphism $k : J \to C$ which makes the left triangle commute is unique, whenever it exists. If such a morphism does not exist, then the rewriting step is not feasible. Moreover, note that the canonical DPO derivations can be seen as a special instance of these, obtained considering as interface J the empty graph.

In these derivations, the left-hand side L of a production must occur completely in G. However, in a *borrowed context* (BC) derivation the graph L might occur partially in G, since the latter may interact with the environment through J in order to exactly match L. Those BCs are the "smallest" extra contexts needed to obtain the image of L in G. The mechanism was introduced in [11] in order to derive an LTS from direct derivations, using BCs as labels. The following definition is lifted from [28], extending the original one by including also morphisms that are not necessarily mono. Note that the labels derived in this way correspond to the labels derived via relative pushouts in a suitable category.

Definition 14 (rewriting with borrowed contexts). *Given a production $p : L \overset{l}{\leftarrowtail} I \overset{r}{\longrightarrow} R$, a graph with interfaces $J \to G$ and a mono $d : D \rightarrowtail L$, we say that $J \to G$* reduces to *$K \to H$ with transition label $J \rightarrowtail F \leftarrowtail K$* via *$p$ and d if there are graphs G^+, C and additional morphisms such that the diagram below commutes and the squares are either pushouts (PO) or pullbacks (PB). In this case we write $J \to G \xrightarrow{J \rightarrowtail F \leftarrowtail K} K \to H$, which is also called* rewriting step with borrowed context.



Consider the diagram above. The upper left-hand square merges the left-hand side L and the graph G to be rewritten according to a partial match $G \leftarrowtail D \rightarrowtail L$. The resulting graph G^+ contains a total match of L and can be rewritten as in the standard DPO approach, producing the two remaining squares in the upper row. The pushout in the lower row gives us the borrowed (or minimal) context F which is missing in order to obtain a total match of L, along with a morphism $J \rightarrowtail F$ indicating how F should be pasted to G. Finally, we need an interface for the resulting graph H, which can be obtained by "intersecting" the borrowed context F and the graph C via a pullback.

Note that two pushout complements that are needed in Definition 14, namely C and F, may not exist. In this case, the rewriting step is not feasible.

6 From Process Reductions to Graph Rewrites

Following [13], this section introduces the rewriting system $\mathcal{R}_{CCS}$, showing how it simulates the reduction semantics for processes: it is quite simple, since it contains just two rules, depicted in Fig. 4. The first rule models a synchronisation, whereas the second models a τ-transition. Note that, in order to disable reduction inside prefixes, we enrich our encoding, attaching an edge *go* on the root node of each process. So, let $[\![P]\!]^g_\Gamma = [\![P]\!]^p_\Gamma \otimes go$. Moreover, for any graph $\mathbb{G}$ with interfaces $(\{p\} \cup \Gamma, \emptyset)$, let $reach(\mathbb{G})$ be the graph with the same interfaces reachable from the image of the interface $\{p\} \cup \Gamma$.

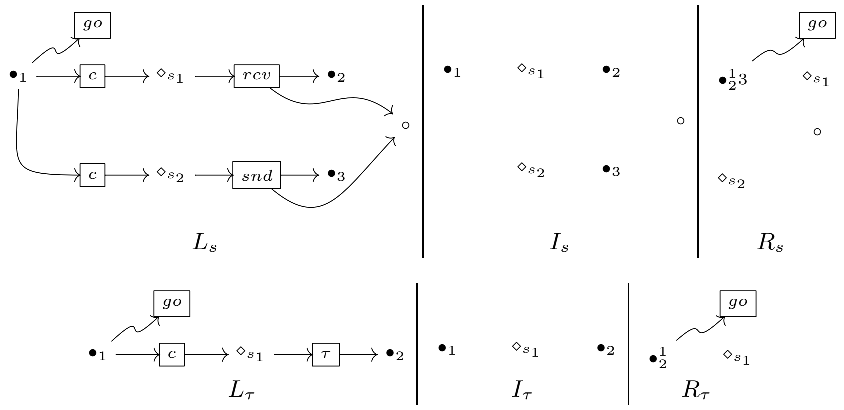


Fig. 4. The productions *synch*: $L_s \leftarrowtail I_s \rightarrow R_s$ and τ: $L_\tau \leftarrowtail I_\tau \rightarrow R_\tau$.

It seems noteworthy that two rules suffice for recasting the reduction semantics of the calculus. First of all, the structural rules are taken care of by the fact that graph morphisms allow for embedding a graph into a larger one, thus simulating the closure of reduction by context. Second, no distinct instance of the rules is needed, since graph isomorphism takes care of the closure with respect to structural congruence, as well as of the renaming of the free name.

Proposition 2 (reductions vs. rewrites). *Let P be a processes, and let Γ be a set of actions such that $\mathtt{fn}(P) \subseteq \Gamma$. If $P \rightarrow Q$, then $\mathcal{R}_{CCS}$ entails a direct derivation $[\![P]\!]^g_\Gamma \Longrightarrow G$ via an injective match, such that $reach(G) = [\![Q]\!]^g_\Gamma$. Vice versa, if $\mathcal{R}_{CCS}$ entails a direct derivation $[\![P]\!]^g_\Gamma \Longrightarrow G$ via an injective match, then there exists a process Q such that $P \rightarrow Q$ and $reach(G) = [\![Q]\!]^g_\Gamma$.*

The correspondence holds since the *go* operator forces the match to be applied only on top, thus forbidding the occurrence of a reduction inside the outermost prefixes. The condition on reachability is needed since, during the reduction, some process components may be discarded, in correspondence of the solving of non-deterministic choices. The restriction to injective matches is necessary in order to ensure that the two edges labelled by c can never be merged together. Intuitively, allowing their coalescing would correspond to the synchronization of two summations, i.e., as allowing a reduction $a.P + \overline{a}.Q \rightarrow P \mid Q$.

Example 2 (rule application). Let P_1 be the process $(\nu a)(a.((\nu c)c.0 \mid (\overline{c}.0{+}b.0)) \mid (\overline{a}.0 + b.0))$: it corresponds to the second element of the chain associated to the open term $P[x] = (\nu a)(a.x \mid (\overline{a}.0 + b.0))$, according to Definition 11. The graph with interfaces $[\![P_1]\!]^g_{\{b\}}$ is concisely represented in Fig. 5: those nodes in the image of the input morphism are denoted so by a label (either p or the free name of the process, b). The application of a rewriting step, resulting in the graph at the bottom, simulates the following reduction, where communication on channel a takes place:

$$(\nu a)(a.((\nu c)(c.0 \mid (\overline{c}.0 + b.0))) \mid (\overline{a}.0 + b.0)) \rightarrow (\nu c)(c.0 \mid (\overline{c}.0 + b.0)).$$

Restricting to the reachable graph (i.e., removing isolated nodes and the leftmost edge labelled by snd) results in the graph $[\![(\nu c)(c.\mathsf{0} \mid (\overline{c}.\mathsf{0} + b.\mathsf{0}))]\!]^{g}_{\{b\}}$.

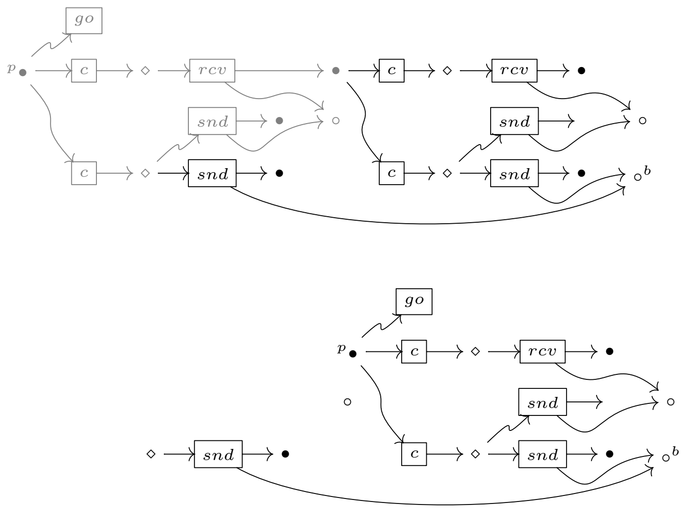


Fig. 5. A rewriting step, simulating a reduction. The grey part denotes the redex.

7 The Synthesised Transition System

This section contains the main results of our paper. Its aim is to apply the BC synthesis mechanism to $\mathcal{R}_{CCS}$, and then to analyse the resulting LTS. Proving along the way a few general results on the technique, we show that the LTS is finitely branching (when quotiented up to isomorphism) and equivalent to a succinct $\rightarrow_C$ whose transitions have a direct interpretation as process transitions. The main theorem of the section states that $\rightarrow_C$ induces on (the encoding of) processes the standard strong bisimilarity.

7.1 Examples of Borrowing

This section analyses how the synthesis mechanism can be applied to our running example $rec_x.(\nu a)(\overline{a}.x \mid a.\mathsf{0}{+}b.\mathsf{0})$. Since the encoded graph is infinite, we consider $J \rightarrowtail G = [\![P_0]\!]^{g}_{\{b\}}$ where $P_0 = (\nu a)(a.\mathsf{0} \mid (\overline{a}.\mathsf{0} + b.\mathsf{0}))$ is the first element of the chain associated to the open term $P[x] = (\nu a)(a.x \mid (\overline{a}.\mathsf{0} + b.\mathsf{0}))$.

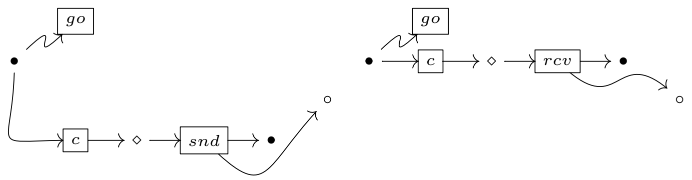
Figs. 10, 11 and 12 show three borrowed contexts derivations for the graph $J \rightarrowtail G$. Here, we discuss the possible transitions with source $J \rightarrowtail G$ that are induced by the synchronization rule $L_s \leftarrow I_s \rightarrow R_s$. Since for each pair of monos $D \rightarrowtail L_s$ and $D \rightarrowtail G$ a labelled transition might exist, it is important to precisely characterize all those possible transitions.

First of all, take as D the entire left-hand side L_s and note that there is only one possible map into G. The construction of the BC transition is shown in Fig. 10: G^+ is exactly the same as G, and C and H are as expected, i.e., as shown in the reduction step of Example 2. In this case, the graph does not need any context for the reaction, since the entire left-hand side L_s occurs in G, and thus, the label of this transition is the identity context, i.e., $id_p \otimes id_b$. Intuitively, this corresponds to the canonical transition labelled τ.

Now take as D the subgraph *SND* in Fig. 6, and the map into the subgraph of G representing the send action on channel b. This choice generates the transition illustrated in Fig. 11: G^+ is the graph G in parallel with a process receiving on channel b; as usual, C contains all the components of the graph G that are not contained in D and H contains the continuation of the processes in parallel. Now, the process encoded in G interacts with the environment: the resulting transition is labelled with a process performing a receive action on channel b.

Let us now consider the mapping of *SND* into the subgraph of G representing the send action on the restricted channel a (in Fig. 11 in graph G, the node corresponding to a is the node above the node labelled b). We have as G^+ the whole G in parallel with a receive prefix on a. However, the pushout complement for $J \rightarrowtail G \rightarrowtail G^+$ does not exist, because the name a is restricted, i.e., it does not appear in the interface J. Thus, this embedding cannot generate any transition: this corresponds, intuitively, to the impossibility for a process of performing an action on some channel a under the restriction (νa).

Note that transitions without counterpart in the canonical operational semantics of CCS can be derived. Consider as D only the root node. There is only a trivial mapping to G, which generates the transition shown in Fig. 12: G^+ is the graph G in parallel with two processes that synchronize on a fresh channel c. The resulting graph H is the starting graph G together with c, and the resulting label is the synchronization of two processes on the channel c. This kind of transitions are often called *not engaged transitions* in the literature of bigraphs [18] (and *independent* in [11]), since they can be performed by any process. They are a standard component of the theory of reactive systems and can be discarded since they do not change the bisimulation relation.



SND *RCV*

Fig. 6. Two subgraphs of L_s.

7.2 Reducing the borrowing

As shown in Section 7.1, in order to know all the possible transitions originating from a graph with interfaces $J \to G$, all the subgraphs D's of L_s and L_τ and all the mono mappings into G should be analysed. To shorten this long and tedious procedure, we show here two pruning techniques for restricting the space of possible D's.

First, note that those items of a left-hand side L that are not in D have to be glued to G through J. Thus, consider a node n of D corresponding to a node n' in L such that n' is the source or the target of some edge e that does not occur in D. Since the edge e is in L but not in D, it must be added to G through J, and thus n must be also in J. A node such as n is called a *boundary node*.

Let us now consider *SND* —as shown in Fig. 6— as a subgraph of L_s. Its root is a boundary node since it has an ingoing edge that occurs in L_s but not in *SND*. Also the name (represented by a node ∘) in *SND* is a boundary node, since in L_s there is an ingoing edge that does not occur in *SND*. Hence this node must be mapped to a node occurring in the interface J of G. This is exactly the reason why there is a transition embedding *SND* into the process sending on b (shown in Fig. 11) and no transition mapping *SND* to the process sending on a.

The notion of boundary nodes is formally captured by the categorical notion of *initial pushout* (formally defined in Appendix A). Since our category has initial pushouts, the previous discussion is formalized by the proposition below.

Proposition 3. *Let $p : L \overset{l}{\hookleftarrow} I \overset{r}{\longrightarrow} R$ be a production and $d : D \rightarrowtail L$ a mono such that square (1) in Fig. 7 is the initial pushout of d. If a graph $J \to G$ can perform a* BC *rewriting step* via *p and d then there exist a mono $D \rightarrowtail G$ and a morphism $J_D \to J$ such that square (2) in Fig. 7 commutes.*

Proof. This trivially follows from Lemma 2 and Lemma 3 in Appendix A. □

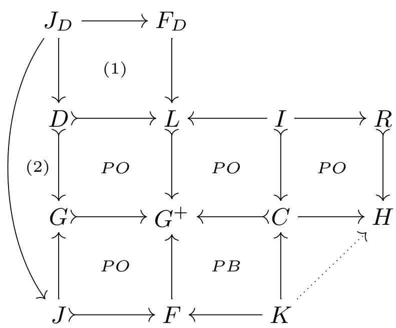


Fig. 7. The BC construction together with square (1) (the initial pushout of $D \rightarrowtail L$) and square (2) (a commuting square).

The above proposition holds in any rewriting system. However, we can find for $\mathcal{R}_{CCS}$ a necessary and sufficient condition to perform a BC rewriting step.

Corollary 1. *A graph $J \to G$ can perform a* BC *rewriting step in $\mathcal{R}_{CCS}$ if and only if there exist*

- *a mono $D \rightarrowtail L$ (where L is the left hand side of some production in $\mathcal{R}_{CCS}$),*
- *a mono $D \rightarrowtail G$,*
- *a morphism $J_D \to J$ (where J_D is the initial pushout of $D \rightarrowtail L$) such that square (2) in Fig. 7 commutes.*

Proof. By Definition 14, a graph $J \to G$ can perform a BC rewriting step if and only if there exist a mono $D \rightarrowtail G$ and a mono $D \rightarrowtail L$ such that the diagram of Definition 14 can be constructed.

Since pushouts and pullbacks always exist, for any choice of $D \rightarrowtail L$ and $D \rightarrowtail G$ problems might arise only with pushout complements. Now note that for both the rules of $\mathcal{R}_{CCS}$ the pushout complement $I \rightarrowtail L \rightarrowtail G^+$ always exists because all the nodes of L are in I. Thus, we have a transition if and only if there exists the pushout complement $J \to G \rightarrowtail G^+$. Since our category has initial pushouts, we can always construct a square such as (1) in Fig. 7. By Lemma 2 (in Appendix A), the square J_D, F_D, G^+, G is an initial pushout of $G \rightarrowtail G^+$. Now, by Lemma 3 (also in Appendix A), we have that the pushout complement of $J \to G \rightarrowtail G^+$ exists if and only if there exists a $J_D \to J$ such that square (2) of Fig. 7 commutes. □

This corollary allows us to heavily prune the space of all possible D's. As far as our case study is concerned, we can exclude all those D's having among boundary nodes a summation node (depicted by ⋄) since these never appear in the interface J of a graph resulting from the encoding of some process. For the same reason, we can exclude all those D's having among their boundary nodes a continuation process node (any of those two nodes depicted by • that are not the root) observing that the only process node in the interface J is the root node.

A further pruning —partially based on proof techniques presented in [11]— is performed by excluding all those D's which generate a BC transition that is not relevant for the bisimilarity. In general terms, we may always exclude all the D's that contain only nodes, since those D's can be embedded in every graph (with the same interface) generating the same transitions. Concerning our case study, those transitions generated by a D having the root node without the edge labelled *go* are also not relevant. In fact, a graph can perform a BC transition using such a D if and only if it can perform a transition using the same D with a *go* edge outgoing from the root. Note indeed that the resulting states of these two transitions only differ for the number of *go* edges attached to the root: the state resulting after the first transition has two *go*'s, the state resulting after the second transition only one. These states are bisimilar, since the number of *go*'s does not change the behavior, as stated by Lemma 12 in Appendix C.

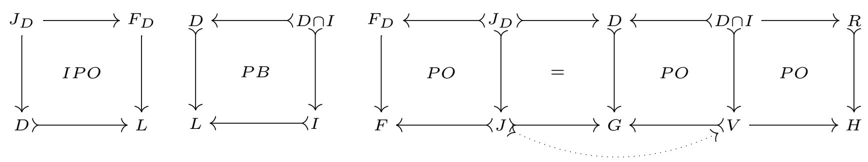
The previous remarks are summed up by the following lemma.

Lemma 1. *Bisimilarity on the* LTS *synthesized by* BC*s coincides with bisimilarity on the* LTS *obtained by considering as partial matches D the graphs L_s, SND and RCV (shown in Fig. 6) as subgraphs of L_s, and the graph L_τ as subgraph of L_τ.*

Proof. Trivial consequence of Proposition 5 presented in the next section. □

7.3 Strong bisimilarity vs. BC bisimilarity

Exploiting the remarks of the previous section, we first introduce a concise LTS containing only those BC transitions that are needed to establish the borrowed bisimilarity. Then, we use this concise LTS to prove our main theorem on the correspondence between the borrowed and the CCS bisimilarity.



(i) (ii) (iii)

Fig. 8. Diagrams used in the propositions of Section 7.

Proposition 4. *Let $p : L \leftarrowtail I \rightarrow R$ be a production of $\mathcal{R}_{CCS}$; $d : D \rightarrowtail L$ a mono such that in Fig. 8, diagram (i) is the initial pushout of d and diagram (ii) is a pullback; and $J \rightarrowtail G$ a graph with interfaces. Then there exists a K such that $J \rightarrowtail G \xrightarrow{J \rightarrowtail F \leftarrowtail K} K \rightarrow H$ via p and d if and only if there exists a mono $D \rightarrowtail G$, a graph V and a morphism $J_D \rightarrow J$ such that the central square of diagram (iii) in Fig. 8 commutes and F and H are constructed as illustrated there.*[2]

Proof. By Corollary 1, once a production $p : L \leftarrowtail I \rightarrow R$ and a mono $d : D \rightarrowtail L$ are chosen, a graph $J \rightarrow G$ can perform a BC rewriting step if and only if there exists a mono $D \rightarrowtail G$ making the central square of the diagram (iii) in Fig. 8 commute. Now we have to show that both F and H can be constructed as described by the diagram (iii) in Fig. 8 if and only if they can be built by the BC construction.

We first prove this for F. Consider Fig. 7, where square (1) is the initial pushout of $d : D \rightarrowtail L$.

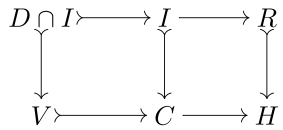
Note that the square J_D, F_D, G^+ and G is a pushout, by the composition property of pushouts. Now let F be the pushout of $J_D \rightarrow F_D$ and $J_D \rightarrow J$, then by the decomposition property of pushouts, also J, G, G^+ and F is a pushout. This proves that if F can be built by this new construction, then it can be built also with the standard BC construction.

Now we have to show the other implication. Since the morphism $J \rightarrowtail G$ is mono, then there exists only one pushout complement of $J \rightarrowtail G \rightarrowtail G^+$, that is

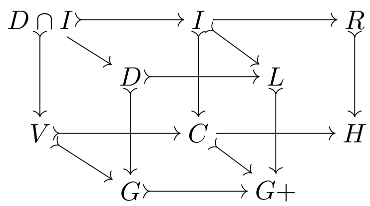
[2] Note that—as detailed later—the arrow $J \rightarrowtail V$ always exists.

exactly the pushout of $J_D \rightarrowtail F_D$ and $J_D \rightarrowtail J$. Note that if $J \to G$ is not mono, our construction is still correct, but it is not complete, i.e., some BC transitions might exist that cannot be obtained via the new construction.

Next we show that if H is built by our construction then H could be built also with the standard BC construction. The morphism $D \cap I \to R$ is divided by I. Thus we get the following diagram where the two squares are pushouts.



Now we can construct G^+ as the pushout of $D \rightarrowtail L$ and $D \rightarrowtail G$. There exists a unique morphism $C \to G+$ such that diagram below commutes.



Note that the left and the front faces are pushouts, and so is the diagonal (the composition of the two faces). Then the back face is a pushout by construction, and thus, by pushout decomposition, also the right face is a pushout. So we have that also H is obtained by the standard double-pushout construction.

Now suppose that H can be constructed by the BC construction. Consider the cube above. The front and the right faces are pushouts, and the extreme right square is also a pushout. Now construct the top and the bottom face of the cube as pullbacks respectively of $I \rightarrowtail L \leftarrowtail D$ and $C \rightarrowtail G^+ \leftarrowtail G$. Now we have that there exists a unique $D \cap I \rightarrowtail V$ such that the diagram commutes. In order to prove that this transition can be derived by our construction we need to prove that the back and the right face of the cube are pushouts.

Now we prove that also the back face of the cube is a pullback. In fact, the front face is a pullback, because it is a pushout along mono, and by pullback composition, the square $D \cap I$, I, G^+, G is a pullback. Since the bottom face is a pullback by construction, we have, by pullback decomposition, that also the back face is a pullback. Now rotate the whole cube, in a such way that the right face becomes the bottom face. The bottom face is now a pushout along mono, and hence a Van Kampen square (see Definition 17 in Appendix B). The lateral faces of the rotated cube are all pullbacks (some of them by construction and some others because they are pushouts along monos) and then by the Van Kampen property, also the top face (in the depicted diagram it is the right face) is a pushout. By composition and decomposition of pushouts, it trivially follows that also the back face (of the depicted cube) is a pushout.

Note that the construction of H is independent of the interface J, and thus this proof can be used also for those graphs where $J \to G$ is not mono. □

The proposition above is a key step in the definition of a concise LTS. In fact, it tells us how to construct the label F and the resulting state H, just starting from a set of minimal rules of the form $F_D \leftarrowtail J_D \rightarrowtail D \leftarrowtail D \cap I \to R$. Given a mono $D \rightarrowtail G$, the resulting state H can be computed in a DPO step, i.e., all the items of G matched by D and not in $D \cap I$ are removed and replaced by R. This transition is possible only if there exists a mono morphism $J_D \rightarrowtail J$ such that the central diagram commutes. In this case, the resulting label F is computed as the pushout of the minimal label $J_D \rightarrowtail F_D$ and $J_D \rightarrowtail J$.

We thus now define a concise transition system, starting from the set of rules, of the form $F_D \leftarrowtail J_D \rightarrowtail D \leftarrowtail D \cap I \to R$, that are depicted in Fig. 9. The main difference with respect to the standard transition system is that the interface J of a graph is never enlarged by a transition, but always remains the same.

Definition 15 (concise transition system). *Let the graph D be either SND, RCV, L_s or L_τ; and let J_D, F_D, $D \cap I$ and R be the graphs defined according to Fig. 9. Then, $J \rightarrowtail G \xrightarrow{J \rightarrowtail F \leftarrowtail J}_C J \to H$ if and only if a diagram as the one illustrated in Fig. 8 (iii) can be constructed, where the morphism $J \to H$ is uniquely induced by $H \leftarrow V \rightarrowtail G \leftarrowtail J$.*

Note that the pushout complement of $D \cap I \rightarrowtail D \rightarrowtail G$ always exists because for each D as in Fig. 9 all the nodes of $D \cap I$ are in D, and thus we have a transition for each $D \rightarrowtail G$ and for each $J_D \rightarrowtail J$ such that the central diagram commutes. Moreover, the morphism $J \rightarrowtail V$ always exists (since J is discrete and V contains all nodes of G) and it is unique (since $V \rightarrowtail G$ is mono).

More precisely, consider either *SND* or *RCV* as D: the existence of a morphism $J_D \rightarrowtail J$ means that the name used in the synchronisation must occur in the interface. Whenever either L_s or L_τ is D, J_D is the empty graph $\emptyset$ and thus a morphism always exists. In these two latter cases the label of the transition is always the span of identities on J and the resulting state is exactly the state obtained from a DPO direct derivation.

In order to grasp the difference between $\to$ and $\to_C$, consider the states $K \to H$ resulting from the BC transition shown in Fig. 11. The interface K is the original interface J plus a summation node ($\diamond$) pointing to an isolated summation node, and a new process node ($\bullet$) pointing to the root. Intuitively, this transition can be described as $rec_x.(\nu a)(\overline{a}.x \mid a.0 + b.0) \xrightarrow{-|\bar{b}.P+M} P$, where P and M are meta-variables denoting respectively a process and a summation. The concise LTS forgets about P and M, and the transition represented in $\to_C$ is $rec_x.(\nu a)(\overline{a}.x \mid a.0+b.0) \xrightarrow{-|\bar{b}.0}_C 0$. This operation is performed without changing the resulting bisimilarity, as stated below.

Proposition 5. *Let $\sim$ be the* BC *bisimilarity, and let $\sim_C$ be the bisimilarity defined on $\to_C$. Then $\sim_C$ and $\sim$ coincide for all those graphs with discrete interfaces belonging to the image of our encoding.*

Proof. See appendix.

The previous proposition allows a simpler proof of the correspondence between strong bisimilarity for CCS and the one resulting from the BC construction.

Theorem 1. *Let P, Q be processes, and let Γ be a set of names, such that* $\mathtt{fn}(P) \cup \mathtt{fn}(Q) \subseteq \Gamma$*. Then* $[\![P]\!]^g_\Gamma \sim [\![Q]\!]^g_\Gamma$ *if and only if* $P \sim_{CCS} Q$*.*

Proof. Here we give just a brief sketch of the proof. First of all, note that the set of inference rules below define the same LTS as that in Definition 2, for $A \subseteq \mathcal{N}$ a finite set of names, Q, R and S processes, and M and N summations.

$$\frac{P \equiv (\nu A)((\tau.Q + M) \mid R)}{P \xrightarrow{\tau} (\nu A)(Q \mid R)} \qquad \frac{P \equiv (\nu A)((\bar{a}.Q + M) \mid (a.R + N) \mid S)}{P \xrightarrow{\tau} (\nu A)(Q \mid R \mid S)}$$

$$\frac{P \equiv (\nu A)((a.Q + M) \mid R) \quad a \notin A}{P \xrightarrow{a} (\nu A)(Q \mid R)} \qquad \frac{P \equiv (\nu A)((\bar{a}.Q + M) \mid R) \quad a \notin A}{P \xrightarrow{\bar{a}} (\nu A)(Q \mid R)}$$

The correspondence between the concise LTS $\rightarrow_C$ and the standard LTS of CCS is then quite evident, since each of those inference rules above exactly corresponds to a rule $R \leftarrow D \cap I \rightarrowtail D \leftarrowtail J_D \rightarrowtail F_D$ in Fig. 9.

For instance, the third rule above corresponds to the third row $D = RCV$ in Fig. 9. Indeed, $P \equiv (\nu A)((a.Q + M) \mid R)$ if and only if RCV can be embedded in G where $J \rightarrowtail G$ is $[\![P]\!]^g_\Gamma$. The condition $a \notin A$ is satisfied if and only if a occurs in the interface J, i.e., if and only if there exists a mono morphism $J_{RCV} \rightarrowtail J$ such that everything commutes. If such a condition is satisfied a transition in $\rightarrow_C$ is performed with label $J \rightarrowtail F \leftarrowtail J$ where $J \rightarrowtail F$ is (part of) the pushout of $J_{RCV} \rightarrowtail J$ and $J_{RCV} \rightarrowtail F_{RCV}$. Since the latter morphism is fixed, $J \rightarrowtail F$ depends only on $J_{RCV} \rightarrowtail J$, i.e., it depends only on the name of J corresponding to the unique name of J_{RCV}, that here we have called a. Then, for each graph with interface J such that RCV occurs inside, and such that the unique name of RCV occurs in J with name a, a transition is performed with a label depending only on a. Roughly, this label can be thought of as a context corresponding to $[\![- \mid \bar{a}.0]\!]^g_\Gamma$ with $J = \{p\} \cup \Gamma$. The resulting state $(\nu A)(Q \mid R)$ does not exactly correspond to the state resulting from $\rightarrow_C$, since the latter contains those graphs that represent discarded choices. However, these summations are not connected anymore to the reachable graph and to the *go*-edge, and thus they do not influence in any way the behavior of the resulting graph.

The second rule corresponds to the second row $D = L_s$. In fact, $P \equiv (\nu A)((\bar{a}.Q + M) \mid (a.R + N) \mid S)$ if and only if L_s can be embedded into G where $J \rightarrowtail G$ is $[\![P]\!]^g_\Gamma$. There are no other conditions on this rule and this is exactly expressed by the fact that J_{L_s} is the empty graph $\emptyset$. The τ-label exactly corresponds to the label of $\rightarrow_C$ given by the span of identities on J. □

8 Conclusions and Further Work

Our paper presents a case study in the synthesis of LTSs for process calculi. A sound and complete graphical encoding for processes is exploited in order to

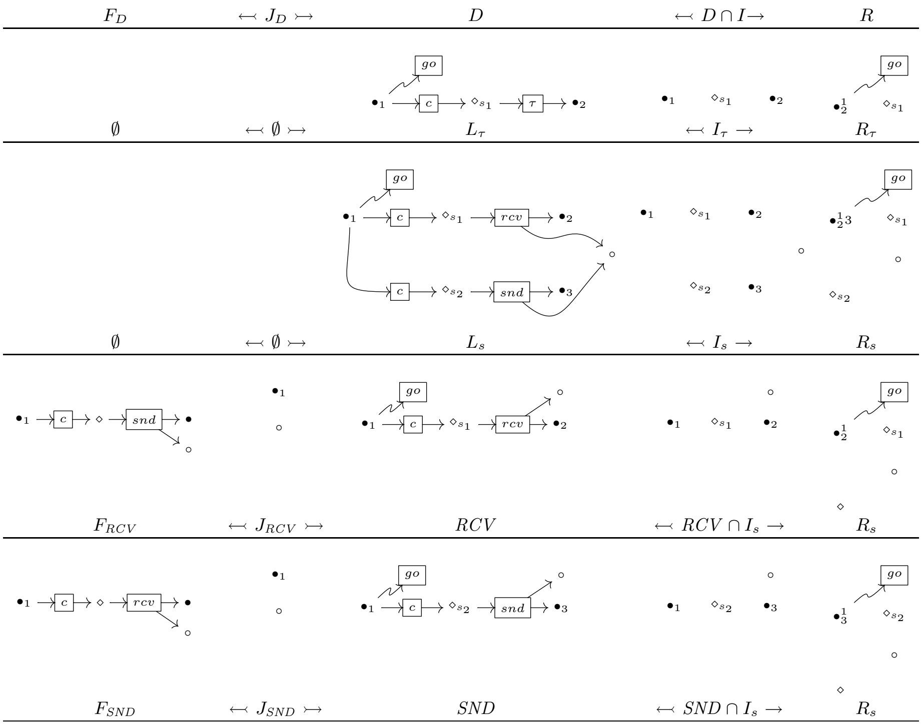


Fig. 9. The derivation rules for the concise LTS (∅ denotes the empty graph).

apply the BC mechanism for automatically deriving an LTS: states are graphs with interfaces, labels are cospans of graph morphisms, and two (encodings of) processes are strongly bisimilar in the distilled LTS if and only if they are also strongly bisimilar according to the standard LTS.

We consider our case study to be relevant for the reasons outlined below.

Technically, its importance lies in the pruning techniques that have been developed in order to cut to a manageable size the borrowed LTS: they exploit abstract categorical definitions, such as initial pushouts, yet resulting in a simplified LTS with the same bisimulation relation (see Proposition 3).

Methodologically, its relevance is due to its focussing on a fully-fledged case study, including also possibly recursive processes: most examples in the literature restrain themselves to the finite fragment of a calculus, as it happens for the encoding of CCS processes into bigraphs presented by Milner in [23].

In order to further illustrate the advantages (and the possibilities for future developments) of our approach, let us consider the latter proposal, similar in aim to our work. It is noteworthy that the encoding into graphs with interfaces allows the use of two rewriting rules only: intuitively, these rules are *non-ground* since they can be both contextualized *and* instantiated. This feature results in synthetising a finitely branching (also for possibly recursive processes) LTS: this seems one of the key advantages of our technique when compared to the bigraphical approach, where reaction rules must be ground, hence infinite in number and inducing an infinitely branching LTS already for finite processes. As far as we are aware, in all the encodings of calculi in the theory of reactive system, there are infinitely many rules (represented by rule schemata). The only exceptions we know are the present paper and the encoding of Logic Programming presented in [6].

This non-groundness supports our hope to use the BC mechanism for distilling a set of inference rules, instead of characterizing directly the set of possible labelled transitions. This should be obtained by extending Proposition 4 and offering an explicit construction of the interface K for the target state of a transition: its construction was irrelevant for our purposes here, since the reuse of the interface J of the starting state does not change the bisimilarity. A related composition result is presented in [2].

Finally, we consider promising the combined use of a graphical encoding (into graphs with interfaces) and of the BC techniques, and we plan to test its expressiveness by capturing also nominal calculi. We feel confident that our approach could be safely extended to those calculi whose distinct feature is name fusion [24], while it might fail for calculi where a more flexible notion of name scoping is needed, as suggested by preliminary results on the π-calculus in [17].

References

1. P. Baldan, A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Concurrent semantics of algebraic graph transformation. In H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 3, pages 107–187. World Scientific, 1999.

2. P. Baldan, H. Ehrig, and B. König. Composition and decomposition of DPO transformations with borrowed context. In A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozemberg, editors, *Graph Transformation*, volume 4178 of *Lect. Notes in Comp. Sci.*, pages 153–167. Springer, 2006.
3. M. Barr and C. Wells. *Category Theory for Computing Science*. Les Publications CMR, 1999.
4. G. Berry and G. Boudol. The chemical abstract machine. *Theor. Comp. Sci.*, 96:217–248, 1992.
5. F. Bonchi, F. Gadducci, and B. König. Process bisimulation via a graphical encoding. In A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozemberg, editors, *Graph Transformation*, volume 4178 of *Lect. Notes in Comp. Sci.*, pages 168–183. Springer, 2006.
6. F. Bonchi, B. König, and U. Montanari. Saturated semantics for reactive systems. In *Logic in Computer Science*. IEEE Computer Society, 2006. To appear.
7. A. Corradini and F. Gadducci. An algebraic presentation of term graphs, via gs-monoidal categories. *Applied Categorical Structures*, 7:299–331, 1999.
8. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26:241–265, 1996.
9. H. Ehrig, K. Ehrig, U. Prange, and G. Täntzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. Springer, 2006.
10. H. Ehrig, A. Habel, J. Padberg, and U. Prange. Adhesive high-level replacement categories and systems. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozemberg, editors, *Graph Transformation*, volume 3256 of *Lect. Notes in Comp. Sci.*, pages 144–160. Springer, 2004.
11. H. Ehrig and B. König. Deriving bisimulation congruences in the DPO approach to graph rewriting. In I. Walukiewicz, editor, *Foundations of Software Science and Computation Structures*, volume 2987 of *Lect. Notes in Comp. Sci.*, pages 151–166. Springer, 2004.
12. J. Engelfriet and T. Gelsema. Multisets and structural congruence of the π-calculus with replication. *Theor. Comp. Sci.*, 211:311–337, 1999.
13. F. Gadducci. Term graph rewriting and the π-calculus. In A. Ohori, editor, *Programming Languages and Semantics*, volume 2895 of *Lect. Notes in Comp. Sci.*, pages 37–54. Springer, 2003.
14. F. Gadducci and R. Heckel. An inductive view of graph transformation. In F. Parisi-Presicce, editor, *Recent Trends in Algebraic Development Techniques*, volume 1376 of *Lect. Notes in Comp. Sci.*, pages 219–233. Springer, 1997.
15. F. Gadducci, R. Heckel, and M. Llabrés. A bi-categorical axiomatisation of concurrent graph rewriting. In M. Hofmann, D. Pavlovic̀, and G. Rosolini, editors, *Category Theory and Computer Science*, volume 29 of *Electr. Notes in Theor. Comp. Sci.* Elsevier Science, 1999.
16. F. Gadducci and U. Montanari. A concurrent graph semantics for mobile ambients. In S. Brookes and M. Mislove, editors, *Mathematical Foundations of Programming Semantics*, volume 45 of *Electr. Notes in Theor. Comp. Sci.* Elsevier Science, 2001.
17. F. Gadducci and U. Montanari. Observing reductions in nominal calculi *via* a graphical encoding of processes. In A. Middeldorp, V. van Oostrom, F. van Raamsdonk, and R.C. de Vrijer, editors, *Processes, terms and cycles (Klop Festschrift)*, volume 3838 of *Lect. Notes in Comp. Sci.*, pages 106–126. Springer, 2005.
18. O. H. Jensen and R. Milner. Bigraphs and transitions. In G. Morriset, editor, *Principles of Programming Languages*, pages 38–49. ACM Press, 2003.

19. S. Lack and P. Sobociński. Adhesive and quasiadhesive categories. *Informatique Théorique et Applications/Theoretical Informatics and Applications*, 39:511–545, 2005.
20. J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In C. Palamidessi, editor, *Concurrency Theory*, volume 1877 of *Lect. Notes in Comp. Sci.*, pages 243–258. Springer, 2000.
21. R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.
22. R. Milner. The polyadic π-calculus: A tutorial. In F.L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, volume 94 of *Nato ASI Series F*, pages 203–246. Springer, 1993.
23. R. Milner. Pure bigraphs: Structure and dynamics. *Information and Computation*, 204:60–122, 2006.
24. J. Parrow and B. Victor. The fusion calculus: Expressiveness and simmetry in mobile processes. In V. Pratt, editor, *Logic in Computer Science*, pages 176–185. IEEE Computer Society Press, 1998.
25. V. Sassone and P. Sobociński. Deriving bisimulation congruences using 2-categories. *Nordic Journal of Computing*, 10:163–183, 2003.
26. V. Sassone and P. Sobociński. Reactive systems over cospans. In *Logic in Computer Science*, pages 311–320. IEEE Computer Society Press, 2005.
27. P. Sewell. From rewrite rules to bisimulation congruences. *Theor. Comp. Sci.*, 274:183–230, 2004.
28. P. Sobociński. *Deriving bisimulation congruences from reduction systems.* PhD thesis, BRICS, Department of Computer Science, University of Aaurhus, 2004.

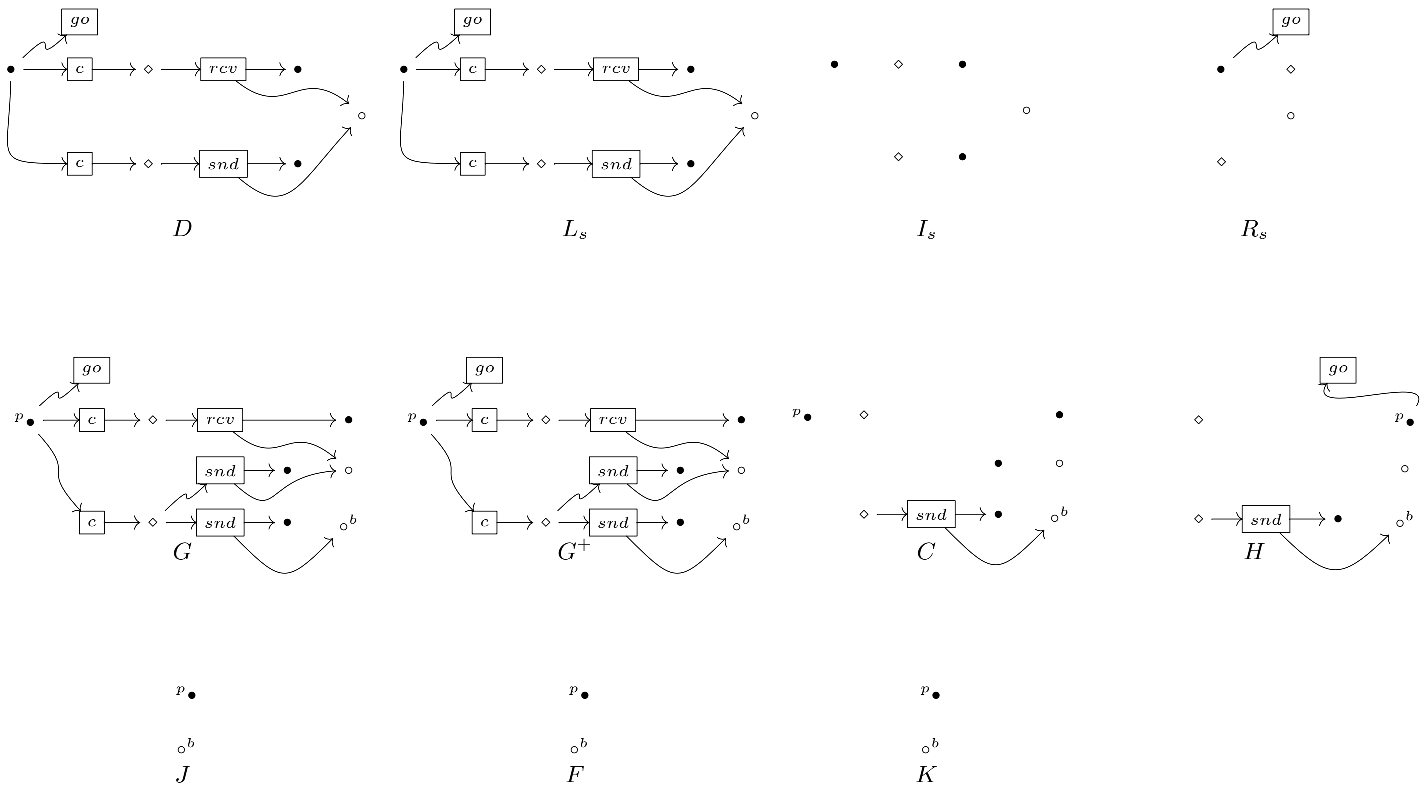


Fig. 10. The internal synchronization generates a span of identities as label.

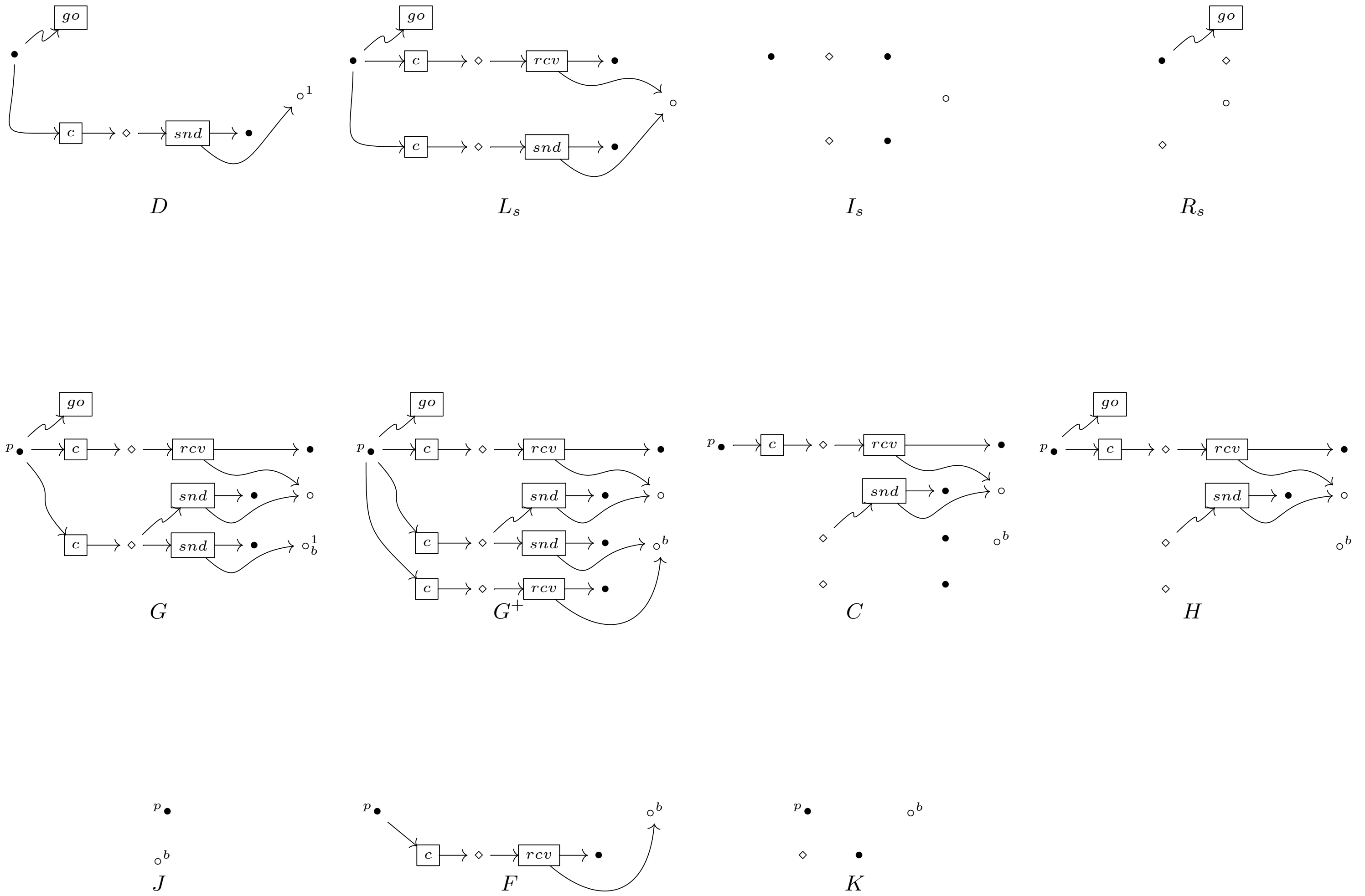


Fig. 11. This borrowed context transition represents a synchronization with the environment and its label is a receive action on b.

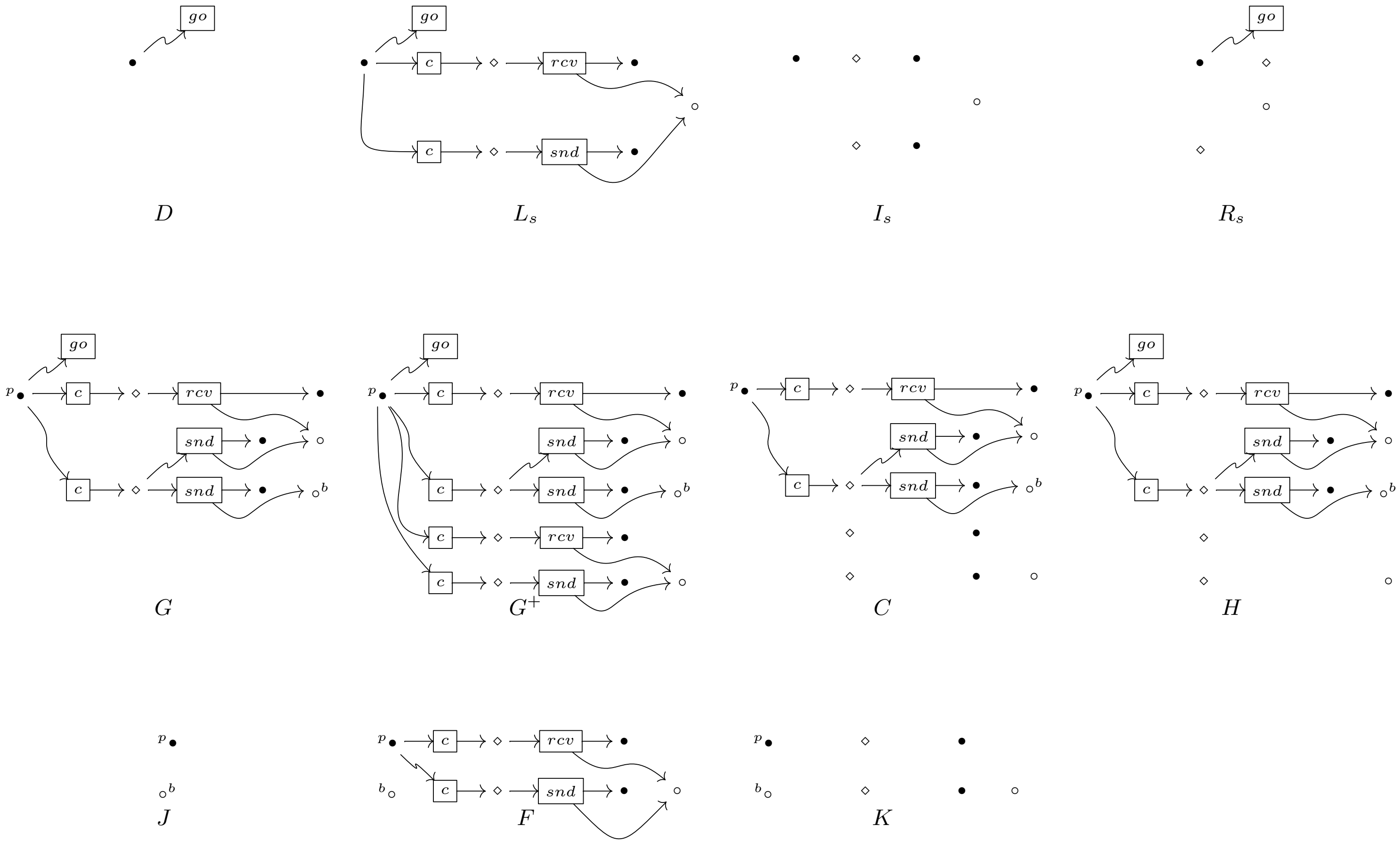


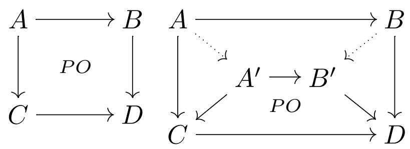
Fig. 12. A transition which is not engaged: its label contains the entire left-hand side L_s (except for the *go* edge).

Appendix A: Initial Pushouts

Here we briefly report the definition of initial pushout, and the two easy results proved in [9], which are useful in order to prove Proposition 3.

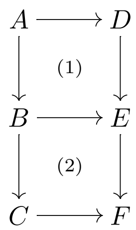
Note that the category of (typed) hypergraph we are working in has initial pushouts for all arrows.

Definition 16 (initial pushout). *Let the square (1) below be a pushout. It is an initial pushout of* $C \to D$ *if for every other pushout as in diagram (2) there exist two unique morphisms* $A \to A'$ *and* $B \to B'$ *such that diagram (ii) commutes.*

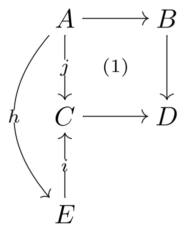


(1) (2)

Lemma 2. *Let the square (1) below be an initial pushout of* $B \to E$*, and the square (2) a pushout. Then the exterior square is an initial pushout of* $C \to F$*.*



Lemma 3. *Let the square (1) below be an initial pushout of* $C \to D$*. The pushout complement of* $E \to C \to D$ *exists if and only if there exists a morphism* $h : A \to E$ *such that* $i \circ h = j$*.*



Appendix B: On Adhesive Categories

We recall here the definition of adhesive categories [19]. We do not provide any introduction to basic categorical constructions such as products, pullbacks and pushouts, referring the reader to Sections 5 and 9 of [3].

Definition 17 (adhesive categories). *A category is called* adhesive *if*

- *it has pushouts along monos;*
- *it has pullbacks;*
- *pushouts along monos are* Van Kampen *(*VK*) squares.*

Referring to Fig. 13, a VK *square is a pushout like* (*i*)*, such that for each commuting cube as in* (*ii*) *having* (*i*) *as bottom face and the back faces of which are pullbacks, the front faces are pullbacks if and only if the top face is a pushout.*

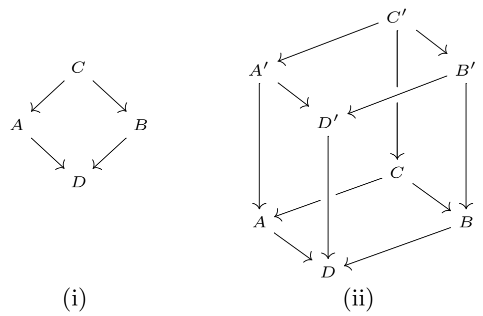


Fig. 13. A pushout square (*i*), and a commutative cube (*ii*).

There are at least two properties of interest for adhesive categories. The first is that adhesive categories subsume many properties of HLR categories [10]. This ensures that several results about parallelism are also valid for DPO rewriting in adhesive categories, if the rules are given by spans of monos [19].

The second fact is concerned with the associated category of *input-linear cospans* (i.e., pairs of arrows with common target, where the first is a mono). As already suggested in [14], any DPO rule can be represented by a pair of cospans, and the bicategory freely generated from the rules represents faithfully all the derivations obtained using monos as matches [15]. Furthermore, the resulting bicategory has relative pushouts [20], hence it is possible to derive automatically a well-behaved behavioral equivalence [26], namely, a bisimulation equivalence which is also a congruence with respect to the closure under (suitable) contexts.

In the context of the present paper we use the fact that the category of (typed) hypergraphs is adhesive and hence we can use all properties of adhesive categories in the proofs.

Appendix C: Proof of Proposition 5

The proof of Proposition 5 is rather long and technical, and thus we decided to report it in a separate section.

During the whole section we use D, C, G^+, H, F and K to denote the graphs used during the BC rewriting step of Definition 14.

Furhtermore we define *Reachable* as the set of all graphs that can be reached by borrowed context transitions from the encoding of some CCS process.

Definition 18 (Reachability). *Let $J \to G$ be a graph with interfaces. We say that $J \to G$ is reachable ($J \to G \in$ Reachable) if and only if it is the encoding of some* CCS *process or it can be reached through a* BC *rewriting step in $\mathcal{R}_{ccs}$ from a reachable graph.*

First of all, in order to avoid confusion, note that this definition is not related with the *reach* function defined in Section 6.

Note that *Reachable* is larger than the image of our encoding. This fact is mirrored in the rules simulating the reduction semantics, where all the discarded summations remain in the resulting graph as disconnected parts. However, for the resulting graph $K \to H$ also K may assume a somewhat strange shape. Consider as an example the state $K \to H$ resulting from the BC transition shown in Fig. 11. The interface K contains a summation node ($\diamond$) pointing to an isolated summation node, and a new process node ($\bullet$) pointing on the root. The following lemma describes how interface are structured in reachable graphs.

Lemma 4. *Let $i : J \to G$ be a reachable graph. Then the following holds*

1. *J is discrete,*
2. *i is mono on name and summation nodes (not necessarily on process nodes),*
3. *i sends summation nodes to isolated summation nodes.*

Proof.

1. The interface J is discrete in the encoding of all the CCS processes. Now suppose we have a graph with discrete interface and consider one of its possible transition. Since both I_s and I_τ are discrete, then all the edges involved in the rewriting step occur neither in C nor in K (since F contains only the nodes and edges needed for rewriting).
2. This property holds in the encoding of all CCS processes. Suppose we have a graph with i mono on name and summation nodes and consider a possible transition. The morphisms $F \to G^+$ and $K \to C$ are mono on names and summations. Since $I_s \to R_s$ and $I_\tau \to R_\tau$ are mono on names and summations, so will be also $C \to H$. Summing up, since $K \to C$ and $C \to H$ must be mono on names and summations, so is $K \to H$. Note that this does not hold for process nodes since the continuation nodes of I_s are fused in the root node in R_s.
3. This property holds for the encodings of all CCS processes (since in the encoding of processes there is no summation node in the interface). Let $i : J \to G$ be a graph where J contains summations nodes pointing to isolated nodes. Then all the edges attached to those nodes by the environment (as label F) will be removed during the rewriting step.

□

Some more steps are missing before we are ready to use Proposition 4, since there exist reachable graphs that do not have a mono interface.

This allows to derive some labels F with the canonical BC construction that can not be derived with the construction proposed in Proposition 4. In fact, if $J \to G$ is not mono there could be several pushout complements (i.e., several labels F), and some of them can not be derived with the construction proposed in Proposition 4. Consider as an example the diagrams in Fig. 14. Here we have several pushout complements of $J \to G \rightarrowtail G^+$

- F_p is also the pushout of (the obvious) $J_D \to F_D$ and of $j_p : J_D \to J$ that maps • of J_D to p of J,
- F_q is also the pushout of (the obvious) $J_D \to F_D$ and of $j_q : J_D \to J$ that maps • of J_D to q of J,
- $F_{p,q}$ cannot be constructed in a such way.

However in some particular cases, Proposition 4 still holds for non-mono matches.

Lemma 5. *Let $J \to G$ be a reachable graph. Then $J \to G \xrightarrow{J \rightarrowtail F \leftarrowtail K} K \to H$ is a* BC *rewriting step via $D = SND$ (or $D = RCV$) if and only if F and H can be constructed as stated by Proposition 4.*

Proof. It is shown in the proof of Proposition 4, that the construction of H is correct and complete also for non mono interfaces, while the construction of F is still correct but not anymore complete. The completeness does not hold because there could be some pushout complements of $J \to G \rightarrowtail G^+$ that can not be derived with the new construction, as the labels $F_{p,q}$ of Fig. 14. However, a case like that never happens taking $D = SND$ (or $D = RCV$), since in the possible labels there is only one edge attached to the root node. □

Lemma 5 defines a strong link between BC derivations and concise transitions generated by choosing $D = SND$ or $D = RCV$. However it does not give any information about how to obtain the resulting interfaces K.

Consider again the BC transition shown in Fig. 11. Intuitively, this transition can be described as $rec_x.(\nu a)(\overline{a}.x \mid a.0 + b.0) \xrightarrow{-|\bar{b}.P+M} 0 \mid P$. The concise LTS forgets about P and M, and the corresponding transition in $\to_C$ is $rec_x.(\nu a)(\overline{a}.x \mid a.0 + b.0) \xrightarrow{-|\bar{b}.0} 0$. The previous example is extended by the lemma below to all those derivations performed via a D that is either SND or RCV. In the following of this section we use SND, RCV, F_{SND}, F_{RCV}, J_{SND} and J_{RCV} to mean the graphs depicted in Fig. 9.

Lemma 6. *Let $J \to G$ be a reachable graph, and let $J \to G \xrightarrow{J \rightarrowtail F \leftarrowtail K} K \to H$ be a* BC *transition step via $D = SND$ (or $D = RCV$). Then*

- *F_{SND} (or F_{RCV}) occurs in F, i.e., there exists a mono arrow $F_{SND} \rightarrowtail F$ ($F_{RCV} \rightarrowtail F$);*

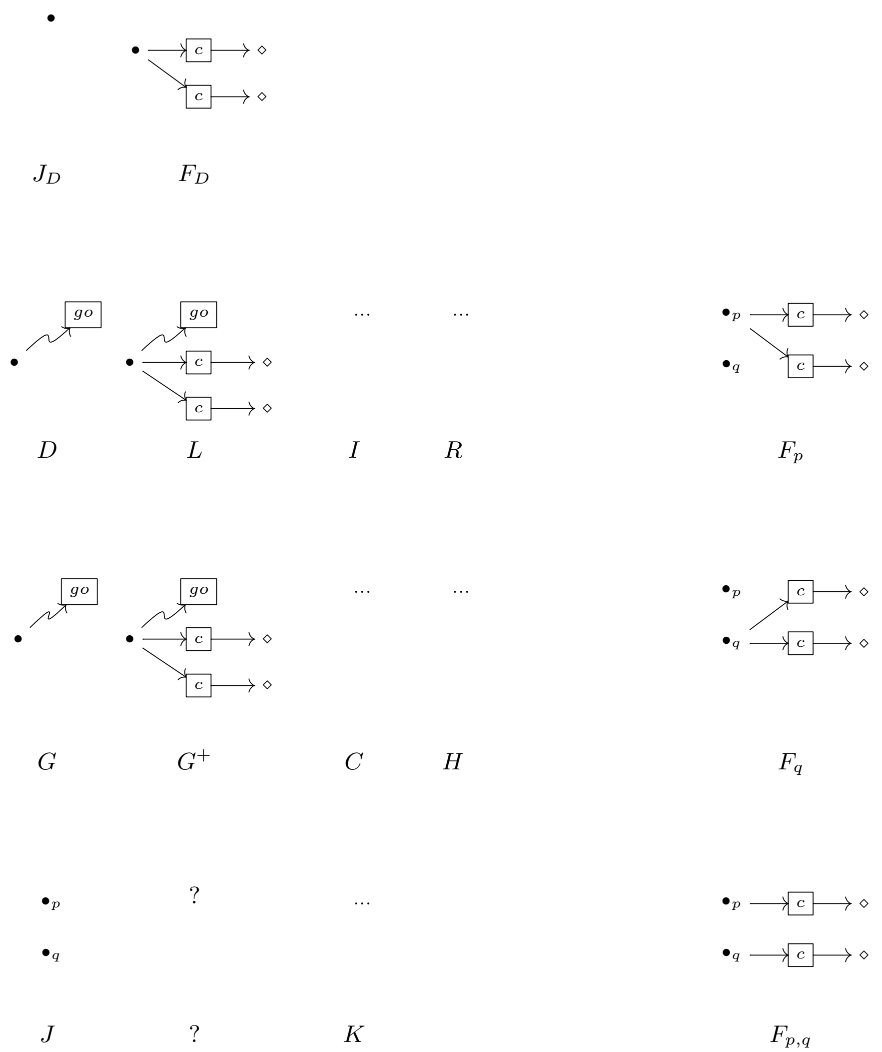


Fig. 14. The graphs D, L, G, G^+ and J are part of a BC derivation for a generic left hand side of a rule L. The upper square is the initial pushout of $D \rightarrowtail L$. The graphs F_p, F_q and $F_{p,q}$ are the possible labels associated to the derivation, i.e., the possible pushout complements of $J \to G \rightarrowtail G^+$, denoted by ? in the table.

- *K is isomorphic to $J + U$, where U is a discrete graph consisting only of a process node ($\bullet$) and a summation node ($\diamond$), and $+$ denotes the disjoint union;*
- *$K \to F$ coincides with $J \rightarrowtail F$ on J, further mapping $\bullet$ into the continuation node of F_{SND} (or F_{RCV}), and $\diamond$ into the summation node of F_{SND} (or F_{RCV});*
- *$K \to H$ maps $\bullet$ into the root node of H and $\diamond$ into an isolated summation node of H.*

Proof. By Lemma 5, the labels of a BC derivation generated choosing $D = SND$ (or $D = RCV$) can be constructed as the pushout of $J_{SND} \rightarrowtail F_{SND}$ and of a mapping $J_{SND} \rightarrowtail J$ that it is surely mono. Then the pushout F entirely contains F_{SND} as a subgraph.

Moreover, note that F contains all the nodes of J (remember that J is discrete since the graph $J \to G$ is reachable) and all the nodes of F_{SND}. Note that in F_{SND} there are a summation node ($\diamond$) and a continuation process ($\bullet$) node that do not occur in J_D: hence these do not occur in G and J. Then, the nodes of F are all the nodes of J plus $\diamond$ and $\bullet$.

Now note that all the nodes of F are present in G^+ and, since $L_s \leftarrowtail I_s$ preserve all the nodes, all the nodes of F occur also in C and hence also in K. □

The BC rewriting steps performed by a reachable graph $J \to G$ via $D = SND$ (or $D = RCV$) are thus in one to one correspondence with the transitions performed in the concise LTS. These latter transitions can be obtained from the BC transitions forgetting the nodes $\bullet$ and $\diamond$ occurring in K: in the following, we write $FORGET(J \rightarrowtail F \leftarrow K \to H)$ to denote this. (Note however, that these nodes are only deleted in K, but not in H.) On the other hand, the BC transitions can be obtained by the concise LTS by adding $\bullet$ and $\diamond$ (and the adequate mapping) to J (this is denoted by $FORGET^{-1}$).

The remark above is summed up by the following lemma.

Lemma 7. *Let $J \to G$ a reachable graph. Then $J \to G \xrightarrow{J \rightarrowtail F \leftarrowtail K} K \to H$ using $D = SND$ (or $D = RCV$) if and only if $J \to G \xrightarrow{J \rightarrowtail F \leftarrowtail J}_{C} J \to H$, where $J \rightarrowtail F \leftarrowtail J \to H = FORGET(J \rightarrowtail F \leftarrow K \to H)$.*

Proof. Trivially follows from Lemma 5 and Lemma 6. □

In the following $FORGET(K \to H)$ denotes the application of $FORGET$ only to the target graph with interfaces. The following two lemmas state that the forgetting and the enriching of the interface do not change the bisimilarity relation.

Lemma 8. *Let $K \to G$ and $K \to G'$ be two reachable graphs such that $J \to G = FORGET(K \to G)$ and $J \to G' = FORGET(K \to G')$. If $K \to G \sim K \to G'$, then $J \to G \sim J \to G'$.*

Proof. Let p and s be the process and summation nodes occurring in K and forgotten in J. If $J \to G$ performs a BC rewriting step, then this can be performed also by $K \to G$ without involving p and s. Since $K \to G$ is bisimilar to $K \to G'$, then also $K \to G'$ can perform this transition without involving p and s. Since this transition does not involve p and s, this can be performed also by $J \to H'$. □

Lemma 9. *Let $J \to G$ and $J \to G'$ be two reachable graphs such that $K \to G = FORGET^{-1}(J \to G)$ and $K \to G' = FORGET^{-1}(J \to G')$. If $J \to G \sim_C J \to G'$, then $K \to G \sim_C K \to G'$.*

Proof. Note that in $\to_C$ the label completely depends on the interface J and the chosen J_D, while the resulting states completely depend from the graph G. However, given a mono $D \rightarrowtail G$, the transition is allowed only if there exists a morphism $J_D \rightarrowtail J$ such that $J_D \rightarrowtail D \rightarrowtail G = J_D \rightarrowtail J \to G$.

Let p and s be respectively the process and summation nodes occurring in K and forgotten in J. The adding of s does not allow any other BC rewriting step, while p allows a new family of concise transitions of $K \to G$ that cannot be performed by $J \to G$. These transitions are added because there is a new morphism $J_D \rightarrowtail K$ such that $J_D \rightarrowtail D \rightarrowtail G = J_D \rightarrowtail K \to G$. These morphisms map the root node of J_D into p. However, all these new transitions can be equally added from $J \to G'$ to $K \to G'$. □

In the following of this section we write $J \rightarrowtail J \leftarrowtail J$ to mean the cospan of identities $id_J : J \to J$.

Lemma 10. *Let $J \to G$ be a reachable graph. Then, $J \to G$ is the source of a transition labelled with $J \rightarrowtail J \leftarrowtail J$ if and only if the transition is generated by choosing as D either L_s or L_τ.*

Proof. If $J \to G$ performs a transition labelled with id_J, then it does not need any structure from the environment and thus one of the left hand sides of the two rules must be completely embedded in G.

Now suppose that $L_s \rightarrowtail G$ then, in the borrowed context derivation diagram $G^+ = G$, and J is a pushout complement of $J \to G \rightarrowtail G$. Since $J \rightarrowtail F$ has to be mono, the graph J is the only possible choice for F.

Now, since all the nodes of L_s are in I_s, the pushout complement of $I_s \rightarrowtail L_s \rightarrowtail G$ exists and the resulting graph C contains all the nodes of G. Thus the pullback of $J \to G$ and $C \rightarrowtail G$ will be again J.

Analogously for L_τ. □

Lemma 11. *Let $J \to G$ be a reachable graph. Then, $J \to G \xrightarrow{J \rightarrowtail J \leftarrowtail J} J \to H$ if and only if $J \to G \xrightarrow{J \rightarrowtail J \leftarrowtail J}_C J \to H$.*

Proof. If $J \to G \xrightarrow{J \rightarrowtail J \leftarrowtail J} J \to H$ then, by Lemma 10, there exists $D \rightarrowtail G$ mono for D equal to either L_s or L_τ. Now note that if such a morphism exists then also $J \to G \xrightarrow{J \rightarrowtail J \leftarrowtail J}_C J \to H$ since J_D is the initial object $\emptyset$. Then the pushout of $id_\emptyset : \emptyset \to \emptyset$ and $!_J : \emptyset \rightarrowtail J$ is $id_J : J \rightarrowtail J$.

If $J \to G \xrightarrow{J \rightarrowtail J \leftarrowtail J}_C J \to H$ then there exists $D \rightarrowtail G$ mono for D equal to either L_s or L_τ. Then a BC transition using this D can be built, obtaining the identity cospan on J as a label. □

The following lemma is the last result that is needed in order to prove Proposition 5.

Lemma 12. *Let $J \to G$ be a reachable graph, and let $J \to G^n$ denote the same graph enriched with n edges labelled go which are attached to the root. Then, for any $n, m > 0$*

- *$J \to G^n \sim J \to G^m$, and*
- *$J \to G^n \sim_C J \to G^m$.*

Proof. Let $R = \{(J \to G^n, J \to G^m) \mid n, m > 0\}$. We show that R is a bisimulation. In fact, if $J \to G^m \xrightarrow{J \rightarrowtail F \leftarrowtail K} K \to H$, then H has m or $m+1$ *go* edges. Since the subgraph D may have at most one *go*, a transition with exactly the same label can be executed by $J \to G^n$, but it will arrive in a state having n or $n+1$ *go* edges. In any case the resulting pairs are contained in R.

For the second statement, note that the transitions of $\to_C$ are completely independent of the number of *go* edges. The only important point is that there exists at least one *go* edge attached to the root. □

Proposition 5. *Let $\sim$ be the BC bisimilarity, and let $\sim_C$ be the bisimilarity defined on $\to_C$. Then $\sim_C$ and $\sim$ coincide for all those graphs with discrete interfaces belonging to the image of our encoding.*

Proof. In order to show that $\sim \subseteq \sim_C$, we prove that the relation S is a bisimulation with respect to $\to_C$, where

$$S = \{(J \to G, J \to G') \mid J \to G \sim J \to G'\} \cap \mathit{Reachable}$$

If $J \to G \xrightarrow{J \rightarrowtail F \leftarrowtail J}_C J \to H$, then this transition has to be generated by a D.

If D is either L_s or L_τ then $J \to G \xrightarrow{J \rightarrowtail J \leftarrowtail J}_C J \to H$ and, by Lemma 11, $J \to G \xrightarrow{J \rightarrowtail J \leftarrowtail J} J \to H$. Now, since $J \to G \sim J \to G'$, then $J \to G' \xrightarrow{J \rightarrowtail J \leftarrowtail J} J \to H'$ with $J \to H \sim J \to H'$. Again by Lemma 11, we have that $J \to G' \xrightarrow{J \rightarrowtail J \leftarrowtail J}_C J \to H'$.

If D is either SND or RCV then, by Lemma 7, $J \to G \xrightarrow{J \rightarrowtail F \leftarrowtail K} K \to H$ where $J \rightarrowtail F \leftarrow K \to H = \mathit{FORGET}^{-1}(J \rightarrowtail F \leftarrowtail J \to H)$. Now, since $J \to G \sim J \to G'$, then $J \to G' \xrightarrow{J \rightarrowtail F \leftarrowtail K} K \to H'$ with $K \to H \sim K \to H'$. Again by Lemma 7, it follows that $J \to G' \xrightarrow{J \rightarrowtail F \leftarrowtail J}_C J \to H'$. Now by Lemma 8 and by $K \to H \sim K \to H'$, it follows that $J \to H \sim J \to H'$.

Now we prove that $\sim_C \subseteq \sim$, showing that the relation S is a bisimulation with respect to $\to$, where

$$S = \{(J \to G, J \to G') \mid J \to G \sim_C J \to G'\} \cap \mathit{Reachable}$$

If $J \to G \xrightarrow{J \rightarrowtail F \leftarrowtail K} K \to H$, then this transition must be generated by $D \rightarrowtail L$ and $D \rightarrowtail G$. The proof proceeds by case analysis on the possible D's.

If D is discrete, then all the nodes of D must be in the interface J. The labels resulting from these D's only depend on the interface J; then, these transitions can be equally performed by graphs having the same interface. Moreover the states resulting from these transition are again bisimilar with respect to $\to_C$, since these transitions do not modify the relevant items of the graphs with interfaces. In fact, these transitions only add isolated nodes both in the graphs and in the interfaces.

Now consider a D with edges. Since by Lemma 4, the summation nodes in the interface of reachable graphs always point to isolated summation nodes, we can exclude a priori all those D's having no isolated summation node as a boundary node.

Thus, the possible remaining D's are those graphs L_τ, L_s, SND and RCV depicted in Fig. 9, and their counterparts without the *go* edge L^g_τ, L^g_s, SND^g and RCV^g.

For the first four we proceed as before, using Lemma 9 instead of Lemma 8.

Now, let D be L^g_τ. Note that a reachable graph can perform a BC rewriting via such a D if and only if it can perform a rewriting via L_τ. Then the only difference between these two rewriting steps is that the first has a *go* edge attached to the root node in the label F, and an additional *go* edge attached to the root node in the resulting H. By Lemma 12 the two resulting states are always bisimilar, since the number of *go* edges does not change the behavior.

The same reasoning applies to L^g_s, SND^g and RCV^g. □