Process Bisimulation *via* a Graphical Encoding*

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

[1] Dipartimento di Informatica, Università di Pisa
[2] Institut für Informatik und 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 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

* Research partially supported by the DFG project SANDS, the EU RTN 2-2001-00346 SegraVis and IST 2004-16004 SEnSOria, and the MIUR PRIN 2005015824 ART.

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 [17] 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 [22] 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 [15] 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 [20]) 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 [10, 12]: processes are encoded in *graphs with interfaces*, an instance of *cospan categories* [11], and process reduction is simulated by *double-pushout* (DPO) rewriting [1]. Since the category of cospans over graphs admits RPOs [21], 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 [8] 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 [15], or on extremely simplified variants of process calculi [20]: more elaborated proposals using bigraphs [18, 14] 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 extended version of the paper [5] contains additional examples, categorical notations and detailed proofs.

2 Two Operational Semantics for CCS

This section introduces CCS [16] 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* $x, y, \ldots$*. An* open process P *is a term generated by the (mutually recursive) syntax*

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

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* $\langle P, \delta, Q\rangle \in L_{CCS}$*).*

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

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

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.*

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

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

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

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

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. [9]. 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 [12] 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 the 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 [15] 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.

3 Graphs and their Extension with Interfaces

We recall a few definitions concerning (typed hyper-)graphs, and their extension with *interfaces*, referring to [6] 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* [7], 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 interface 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 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.*

[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.

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, up-to isomorphism.

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 operators in it.

4 From Processes to Graphs with Interfaces

This section presents our graphical encoding for CCS processes. After presenting 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 an algebra: names are interpreted as variables, so that they are mapped to leaves of the graph and can be safely shared.

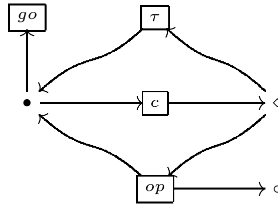


Fig. 1. The type graph T_{CCS} (for $op \in \{rcv, snd\}$).

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 $a \in \mathcal{N}$.

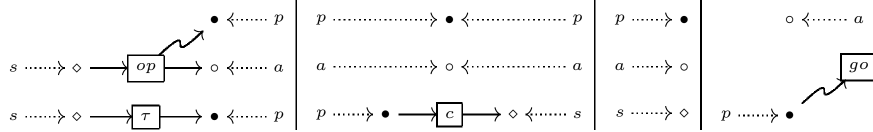


Fig. 2. Graphs op_a (for $op \in \{rcv, snd\}$) and τ; id_p, id_a, and c; $\mathtt{0}_p$, $\mathtt{0}_a$, and $\mathtt{0}_s$; ν_a and *go* (from left to right and top to bottom).

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

$$[\![M]\!]^p_\Gamma = \begin{cases} \mathtt{0}_p \otimes \mathtt{0}_\Gamma & \textit{if } \mathtt{fn}(M) = \emptyset \\ (c \otimes id_\Gamma) \circ [\![M]\!]^s_\Gamma & \textit{otherwise} \end{cases}$$

$$[\![(\nu a)P]\!]^p_\Gamma = \begin{cases} [\![P]\!]^p_\Gamma & \textit{if } a \notin \mathtt{fn}(P) \\ (id_p \otimes \nu_b \otimes id_\Gamma) \circ [\![P\{^b/_a\}]\!]^p_{\{b\}\uplus\Gamma} \ \textit{for } b \notin \Gamma & \textit{otherwise} \end{cases}$$

$$[\![P \mid Q]\!]^p_\Gamma = [\![P]\!]^p_\Gamma \otimes [\![Q]\!]^p_\Gamma \qquad [\![M + N]\!]^s_\Gamma = [\![M]\!]^s_\Gamma \otimes [\![N]\!]^s_\Gamma$$

$$[\![\mathtt{0}]\!]^s_\Gamma = \mathtt{0}_s \otimes \mathtt{0}_\Gamma \qquad [\![\tau.P]\!]^s_\Gamma = (\tau \otimes id_\Gamma) \circ [\![P]\!]^p_\Gamma$$

$$[\![a.P]\!]^s_\Gamma = (rcv_a \otimes id_\Gamma) \circ [\![P]\!]^p_\Gamma \qquad [\![\overline{a}.P]\!]^s_\Gamma = (snd_a \otimes id_\Gamma) \circ [\![P]\!]^p_\Gamma$$

Note the conditional rule for the mapping of $[\![M]\!]^p_\Gamma$. This is required by the use of $\mathtt{0}$ 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 $M \equiv \mathtt{0}$.

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 [10]).

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.

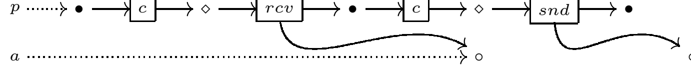


Fig. 3. Encoding for both $[\![(\nu b)a.\overline{b}.0]\!]^{p}_{\{a\}}$ and $[\![a.(\nu b)\overline{b}.0]\!]^{p}_{\{a\}}$.

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., $[\![0]\!]^{p}_{\Gamma}$.

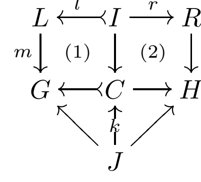
Of course, two recursive expressions may be mapped to isomorphic graphs with interfaces, even if they are not structurally congruent, nor can be unfolded to the same expression. Nevertheless, the extended encoding is clearly still sound.

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 $L \xleftarrow{l} I \xrightarrow{r} R$ 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.*

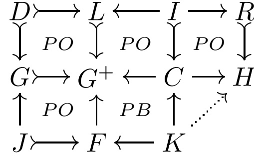
Definition 13 (derivation of graphs with interfaces). *Let $J \to G$ and $J \to H$ be two graphs with interfaces. Given a production $p : L \xleftarrow{l} I \xrightarrow{r} R$, a* match *of p in G is a morphism $m : L \to G$. A* direct derivation *from $J \to G$ to $J \to H$ via p and m is a diagram 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 $k : J \to C$ such that the left triangle commutes is unique, whenever it exists. If such a morphism does not exists, 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. On the contrary, 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 [8] in order to derive an LTS from direct derivations, using BCs as labels. The following definition is lifted from [2], 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 $p : L \xleftarrow{l} I \xrightarrow{r} R$, 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 $J \rightarrowtail F \leftarrow K$* 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 $J \to G \xrightarrow{J \rightarrowtail F \leftarrow K} K \to H$, 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 [10], 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 the upper right corner of Fig. 4 (the remaining graphs in this figure are explained later). The first rule models a τ-transition, whereas the second models synchronisation. Note that, in order to disable reduction inside prefixes, we enrich our encoding, attaching an edge *go* on the root 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 roots $\{p\} \cup \Gamma$.

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 $\texttt{fn}(P) \subseteq \Gamma$. If $P \to Q$, then $\mathcal{R}_{CCS}$ entails a direct derivation $[\![P]\!]^g_\Gamma \Longrightarrow G$ via an injective match, such that $reach(G) = [\![Q]\!]^g_\Gamma$. Viceversa, 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 $P \to Q$ 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 labeled 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 $a.P + \overline{a}.Q \to P \mid Q$.

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 $\to_C$ whose transitions have a direct interpretation as process transitions. The main theorem of the section states that $\to_C$ induces on (the encoding of) processes the standard strong bisimilarity.

7.1 Reducing the borrowing

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

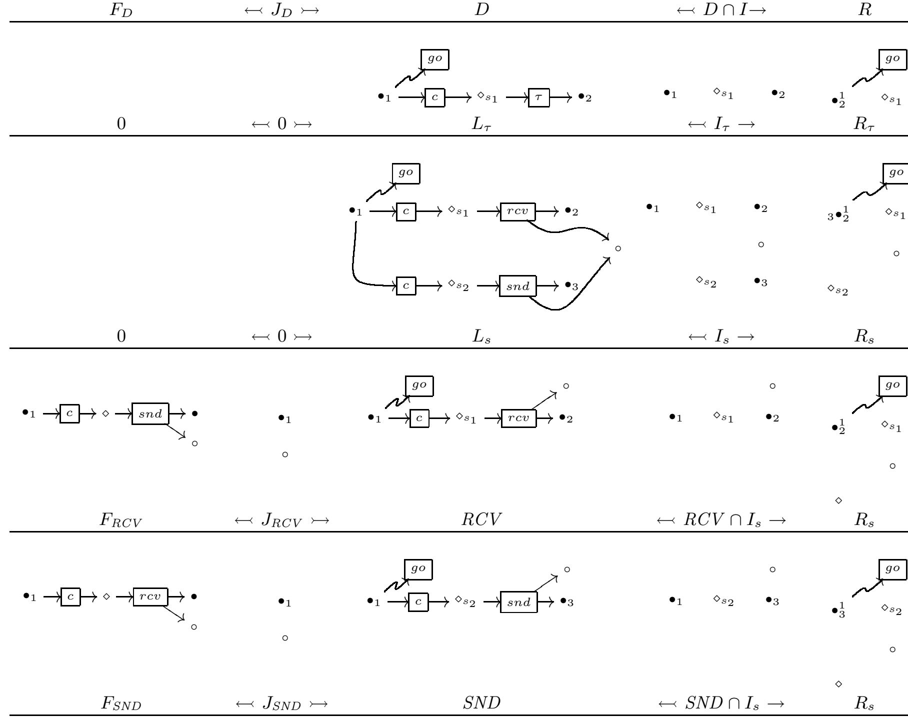


Fig. 4. Productions *synch*: $L_s \leftarrowtail I_s \rightarrow R_s$ and τ: $L_\tau \leftarrowtail I_\tau \rightarrow R_\tau$ (upper right area). The table illustrates the derivation rules for the concise LTS (0 is the empty graph).

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 pasted to G through J. Thus, consider a node n of D corresponding to 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. 4— 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 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.

The notion of boundary nodes is formally captured by the categorical notion of *initial pushout* (as recalled in [5]). Since our category has initial pushouts, the previous discussion is formalized by the proposition below.

Proposition 3. *Let* $p : L \xleftarrow{l} I \xrightarrow{r} R$ *be a production and* $d : D \rightarrowtail L$ *a mono such that diagram (i) in Fig. 5 is the initial pushout of* d*. If a graph* $J \rightarrow G$ *can perform a* BC *rewriting step* via p *and* d *then there exist a mono* $D \rightarrowtail G$ *and a morphism* $J_D \rightarrow J$ *such that diagram (ii) in Fig. 5 commutes.*

This proposition allows 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 $\diamond$) 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 $\bullet$ 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 [8]—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 labeled *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.

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. 4) as subgraphs of L_s, and only the graph L_τ as subgraph of L_τ.*

7.2 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.

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. 5, diagram (i) is the initial pushout of d and diagram (iii) is a pullback; and $J \rightarrowtail G$ a graph with interfaces. Then $J \rightarrowtail G \xrightarrow{J \rightarrowtail F \leftarrow K} K \rightarrow H$* 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 (iv) in Fig. 5 commutes and F and H are constructed as illustrated there.*

$$
\begin{array}{ccc} J_D & \longrightarrow & F_D \\ \downarrow & IPO & \downarrow \\ D & \rightarrowtail & L \end{array}
\qquad
\begin{array}{ccc} J_D & \longrightarrow & D \\ \downarrow & = & \downarrow \\ J & \longrightarrow & G \end{array}
\qquad
\begin{array}{ccc} D & \leftarrowtail & D\cap I \\ \downarrow & PB & \downarrow \\ L & \leftarrowtail & I \end{array}
\qquad
\begin{array}{ccccccccc} F_D & \leftarrowtail & J_D & \rightarrowtail & D & \leftarrowtail & D\cap I & \longrightarrow & R \\ \downarrow & PO & \downarrow & = & \downarrow & PO & \downarrow & PO & \downarrow \\ F & \leftarrowtail & J & \rightarrowtail & G & \leftarrowtail & V & \longrightarrow & H \end{array}
$$

(i) (ii) (iii) (iv)

Fig. 5. Diagrams used in the propositions of Section 7.

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 $R \leftarrow D \cap I \rightarrowtail D \leftarrowtail J_D \rightarrowtail F_D$. 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 morphism $J_D \rightarrowtail J$ such that the central diagram commutes. In this case, the resulting label F is computed as the pushout between 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 $R \leftarrow D \cap I \rightarrowtail D \leftarrowtail J_D \rightarrowtail F_D$, that are depicted in Fig. 4. 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. 4. 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. 5 (iv) 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. 4 all the nodes of $D \cap I$ are in D, and thus we have a transition for each $D \rightarrowtail G$ and for each $J_D \to J$ 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 $J_D \to J$ 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 0 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.

The difference between $\to$ and $\to_C$ can be explained via an analogy to the CCS-like transition $a.\mathsf{0} + b.\mathsf{0} \xrightarrow{-|\bar{b}.P+M} P$. The concise LTS forgets about P and M, and the transition represented in $\to_C$ is $a.\mathsf{0} + b.\mathsf{0} \xrightarrow{-|\bar{b}.\mathsf{0}}_R \mathsf{0}$. 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.*

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 of 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 seems then 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. 4.

For instance the third rule above corresponds to the third row $D = RCV$ in Fig. 4. 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 morphism $J_{RCV} \rightarrow J$ 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 between $J_{RCV} \rightarrowtail J$ and $J_{RCV} \rightarrowtail F_{RCV}$. Since the latter morphism is fixed, $J \rightarrowtail F$ depends only $J_{RCV} \rightarrowtail J$, 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 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 0. 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 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 by Milner in [18].

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 the borrowed context technique with respect to the bigraphical approach, where reaction rules must be ground, hence infinite in number and inducing an infinitely branching LTS already for finite processes.

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 already presented in [3].

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 [19], 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 [13].

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, A. Corradini, T. Heindel, B. König, and P. Sobociński. Processes for adhesive rewriting systems. In W. Aceto and A. Ingólfsdóttir, editors, *Foundations of Software Science and Computation Structures*, volume 3921 of *Lect. Notes in Comp. Sci.* Springer, 2006.
3. P. Baldan, H. Ehrig, and B. König. Composition and decomposition of DPO transformations with borrowed contexts. This volume.
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. Technical Report TR-06-07, Dipartimento di Informatica, Università di Pisa, 2006.
6. A. Corradini and F. Gadducci. An algebraic presentation of term graphs, via gs-monoidal categories. *Applied Categorical Structures*, 7:299–331, 1999.
7. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26:241–265, 1996.
8. 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.
9. J. Engelfriet and T. Gelsema. Multisets and structural congruence of the π-calculus with replication. *Theor. Comp. Sci.*, 211:311–337, 1999.
10. 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.
11. 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.
12. 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.
13. F. Gadducci and U. Montanari. Observing reductions in nominal calculi *via* a graphical encoding of processes. In A. Middeldorp *et al.*, editor, *Processes, terms and cycles (Klop Festschrift)*, volume 3838 of *Lect. Notes in Comp. Sci.*, pages 106–126. Springer, 2005.
14. O. H. Jensen and R. Milner. Bigraphs and transitions. In G. Morriset, editor, *Principles of Programming Languages*, pages 38–49. ACM Press, 2003.
15. 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.
16. R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.
17. 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.
18. R. Milner. Pure bigraphs: Structure and dynamics. *Information and Computation*, 204:60–122, 2006.
19. 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.
20. V. Sassone and P. Sobociński. Deriving bisimulation congruences using 2-categories. *Nordic Journal of Computing*, 10:163–183, 2003.
21. V. Sassone and P. Sobociński. Reactive systems over cospans. In *Logic in Computer Science*, pages 311–320. IEEE Computer Society Press, 2005.
22. P. Sewell. From rewrite rules to bisimulation congruences. *Theor. Comp. Sci.*, 274:183–230, 2004.