Math. Struct. in Comp. Science (2007), *vol.* 17, *pp.* 407–437.
doi:10.1017/S096012950700610X Printed in the United Kingdom

Graph rewriting for the π-calculus[†]

FABIO GADDUCCI

Dipartimento di Informatica, Università di Pisa,
largo Pontecorvo 3c, I-56127 Pisa, Italy
Email: gadducci@di.unipi.it.

Received 4 November 2005; revised 8 September 2006

We propose a graphical implementation for (possibly recursive) processes of the π-calculus, encoding each process into a graph. Our implementation is sound and complete with respect to the structural congruence for the calculus: two processes are equivalent if and only if they are mapped into graphs with the same normal form. Most importantly, the encoding allows the use of standard graph rewriting mechanisms for modelling the reduction semantics of the calculus.

1. Introduction

Historically, the theory of graph rewriting has its roots in the late 1960s as the conceptual extension of the theory of formal languages dealing with structures that are more general than strings. The extension was motivated by a wide range of interests, from pattern recognition to data type specification. Nowadays, the emphasis has shifted from the generative aspects of the formalism and moved toward what could be called the 'state transformation' view: a graph is considered as a data structure on which a set of rewriting rules may implement local changes. Hence, the reduction mechanism itself expresses a basic computational paradigm, where graphs describe the states of an abstract machine and rewrites express its possible evolutions. The interest is confirmed by the large diffusion of visual specification languages, such as the standard UML, and the use of graphical tools for their manipulation.

To some extent, this is also the intuition behind the introduction of process algebras, such as Milner's CCS (Milner 1989): they represent specification languages for concurrent systems, which are considered as structured entities interacting *via* some synchronisation mechanism. A (possibly distributed) system is just a term over a signature, under the hypothesis that each operator represents a basic feature of the system. The reduction mechanism (accounting for the interaction between distinct components of a system) is usually described operationally, according to the so-called SOS-style (Plotkin 1981), where the rewriting steps are inductively defined by a set of inference rules, which are driven by the structure of terms. Novel extensions of the process algebras paradigm involved calculi with higher-order features, such as process mobility. Here systems are terms, which

[†] Research partly supported by the EU within the project HPRN-CT-2002-00275 SEGRAVIS (*Syntactic and Semantic Integration of Visual Modelling Techniques*); and within the FETPI Global Computing, project IST-2004-16004 SENSORIA (*Software Engineering for Service-Oriented Overlay Computers*).

carry a set of associated *names*, and usually provided with a *structural* congruence, which expresses basic observational properties; the reduction mechanism may also change the topology of a system, which formally amounts to a change in the associated set of names.

Recent years have seen many proposals concerning the use of graph rewriting techniques for the simulation of reduction in process algebras, in particular for their mobile extensions. Typically, the use of graphs allows us to avoid the problems associated with the implementation of reduction over the structural equivalence, such as, for example, the α-conversion of bound names. Most of these proposals follow the same pattern. First, a suitable graphical syntax is introduced, and its operators used to implement processes. Then *ad hoc* graph rewriting techniques are usually developed to simulate the reduction semantics. The resulting graphical structures are normally inherently hierarchical (that is to say, roughly speaking, each node/edge is itself a structured entity, and possibly a graph). From a practical point of view, this is unfortunate, since the restriction to standard graphs would allow the reuse of existing theoretical techniques and practical tools.

Building on previous work on the syntactical presentation of graphs and graph rewriting (Corradini and Gadducci 1999a; Corradini and Gadducci 1999b; Gadducci *et al.* 1999) and using formalisms adopted in the algebraic specification community for modelling flow graphs (Căzănescu and Ştefănescu 1992), in this paper we propose an encoding of (possibly recursive) processes of the π-calculus into (typed) graphs, and prove its soundness and completeness with respect to the original reduction semantics. The use of unstructured (that is, non-hierarchical) graphs allows the reuse of standard graph rewriting theory and tools for the simulation of the reduction semantics of the calculus, such as the double-pushout (DPO) approach and the associated concurrent semantics (which allows the simultaneous execution of independent reductions, and thus implicitly defines a non-deterministic concurrent semantics (Baldan *et al.* 1999)).

Our proposal is expressive enough to encode the full calculus, which includes non-deterministic choices. In general, we consider our use of unstructured graphs to be an advance on most of the other implementations of the π-calculus based on graph rewriting, such as König (1999) and Montanari *et al.* (1999), and those based on more general graphical formalisms, such as Milner's *bigraphs* (Milner 2001).

Our representation is reminiscent of the work on the CHARM (Corradini *et al.* 1994) and on *process graphs* (Yoshida 1994). In the former, graphs are represented algebraically, with a term structure that is analogous to the normal form presentation of structurally congruent π-processes originally proposed by Milner (and rightly so, since they are both inspired by the seminal work of Berry and Boudol on the CHAM (Berry and Boudol 1992)). In the latter, an embedding of processes into non-hierarchical graphs is proposed, albeit in a context more reminiscent of interaction nets than of standard graph rewriting. The same considerations hold for *reaction graphs*, underlying the work on χ-calculus (Fu 1999), which in turn shares many assumptions with the *fusion calculus*. It is noteworthy that *solo diagrams* (Laneve *et al.* 2001), the graphical formalism associated with the fusion calculus, uses hyper-graphs techniques that are similar to ours: the main difference concerns the treatment of recursive processes, as well as our interest in also capturing a syntactic operator for non-deterministic choice.

The paper has the following structure. Section 2 introduces the finite fragment of the calculus, its syntax and reduction semantics. Section 3 recalls some basic definitions concerning (typed) graphs, and, in particular, introduces an extension, graphs *with interfaces*, and two operations on them, *sequential* and *parallel composition*. Section 4 briefly recalls some standard theory and tools of the DPO approach to graph transformation. These operators on graphs are needed in Section 5, where a graphical encoding of finite processes is introduced. Our presentation is purely set-theoretical, but its algebraic description, using the already mentioned results on the syntactical presentation of graphs (as surveyed in Bruni *et al.* (2002) and Corradini and Gadducci (1999a)), can be obtained along the lines of Gadducci and Montanari (2002). The graphical encoding for processes is further analysed, and a set of graph rewriting rules for recovering a normal form for each graphical encoding of a process is given in Section 6. Then, Section 7 presents the main result of the paper, namely, that reduction semantics can be simulated using the graph reduction mechanism on the set of graphs obtained by the encoding. Then, Section 8 illustrates some ongoing applications of the graphical mechanism devised in the paper. Finally, Section 9 extends the encoding to recursive processes.

This article is a considerably revised and improved version of Gadducci (2003).

2. The finite fragment of the π-calculus

This section gives a brief introduction to the finite fragment of the π-calculus, its structural equivalence and the associated reduction semantics.

Definition 2.1 (Processes). Let $\mathcal{N}$ be a set of *names*, ranged over by $x, y, w, \ldots$; and let $\Delta = \{x(y), \overline{x}y \mid x, y \in \mathcal{N}\}$ be the set of *prefix operators*, ranged over by δ. A *process* P is a term generated by the (mutually recursive) syntax

$$P ::= M, \quad (\nu x)P, \quad P_1 \mid P_2$$

$$M ::= 0, \quad \delta.P, \quad M_1 + M_2 .$$

We let $P, Q, R, \ldots$ range over the set *Proc* of processes, and $M, N, O \ldots$ range over the set *Sum* of *summations*.

We assume the standard definitions for the set of free names of a process P, denoted $fn(P)$, and for α-convertibility, with respect to the *restriction* $(\nu y)P$ and *input* operators $x(y).P$: in both cases the name y is bound in P and can be freely α-converted. Using these definitions, the behaviour of a process P is described as a relation over *abstract processes*, that is, a relation obtained by closing a set of basic rules under structural congruence.

Definition 2.2 (Reduction semantics). The *reduction relation* for processes is the relation $R_\pi \subseteq Proc \times Proc$, which is closed under the congruence $\equiv$ induced by the set of axioms

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

$$P \mid Q = Q \mid P \qquad P \mid (Q \mid R) = (P \mid Q) \mid R \qquad P \mid 0 = P$$

$$M + N = N + M \qquad M + (N + O) = (M + N) + O \qquad M + 0 = M$$

$$(\nu x)(\nu y)P = (\nu y)(\nu x)P \qquad (\nu x)(P \mid Q) = P \mid (\nu x)Q \text{ for } x \notin fn(P) \qquad (\nu x)0 = 0$$

Fig. 1. The set of structural axioms.

in Figure 1, inductively generated by the following set of axioms and inference rules:

$$\frac{}{x(y).P + M \mid \overline{x}w.Q + N \to P\{^{w}/_{y}\} \mid Q}$$

$$\frac{P \to Q}{(\nu x)P \to (\nu x)Q} \qquad \frac{P \to Q}{P \mid R \to Q \mid R}$$

where $P \to Q$ means that $\langle P, Q \rangle \in R_\pi$.

The first rule denotes the communication between two processes, which may occur within a non-deterministic context. The process $\overline{x}w.Q$ is ready to communicate the (possibly global) name w along the channel x; it then synchronises with process $x(y).P$, and the local name y is thus substituted by w on the residual process P. The two latter rules simply state the closure of the reduction relation with respect to the operators of restriction and parallel composition.

Apart from the lack of the prefix operator $\tau.P$, the syntax and operational semantics for the finite fragment of the calculus just presented coincide with the initial chapter of Sangiorgi and Walker (2001) (see Definition 1.1.1, Table 1.1 and Table 1.3).

3. Graphs and their extension with interfaces

In this section we recall a few definitions concerning (typed hyper-)graphs and their extension with *interfaces* – see Bruni *et al.* (2002) and Corradini and Gadducci (1999a) for a more detailed introduction.

Definition 3.1 (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. A (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* (Corradini *et al.* 1996), that is, graphs labelled over a structure that is itself a graph.

Definition 3.2 (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, that is, such that $t_{G_1} = t_{G_2} \circ f$.

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

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 give an inductive definition of 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 3.3 (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 $\mathbb{G}$, $\mathbb{H}$ 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 use $J \xrightarrow{j} G \xleftarrow{k} K$ to denote a graph with interfaces J and K. 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.

In order to define our encoding for processes, we introduce two operators on graphs with *discrete* interfaces, that is, such that their set of edges is empty.

Definition 3.4 (Two 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. 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 such that $t_k(y) = t_{k'}(y)$ for all $y \in N_k \cap N_{k'}$. Their *parallel composition* is the graph with discrete interfaces $\mathbb{G} \otimes \mathbb{H} = (J \uplus J') \xrightarrow{j''} V \xleftarrow{k''} (K \cup K')$ for V the disjoint union $G \uplus H$, modulo the equivalence on nodes induced by $k(y) = k'(y)$ 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 $\mathbb{H}$, additionally gluing the outputs K of $\mathbb{G}$ with the corresponding outputs K' of $\mathbb{H}$. 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

[†] While the sequential operator corresponds to categorical composition, the parallel operator only recalls the tensor product of monoidal catgeories. A more standard definition for the latter operator is given, for example, in Corradini and Gadducci (1999a). Our choice, however, allows for a compact presentation of the graphical encoding in the following sections.

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

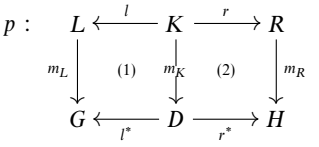

Fig. 2. A direct derivation.

expression is *well formed* if all occurrences of those operators are defined for the interfaces of their arguments, according to Definition 3.4; 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. Rewriting graphs (with interfaces)

This section recalls the basic tools of the double-pushout (DPO) approach to (typed hyper-)graph transformation, as presented in Corradini *et al.* (1997) and Drewes *et al.* (1997), and it introduces its extension to graphs with interfaces.

Definition 4.1 (Graph production). A T-typed *graph production* is a pair of arrows $\langle l : K \to L, r : K \to R \rangle$ in T-**Graph** such that l is mono. A T-typed *graph transformation system* (GTS) $\mathscr{G}$ 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.

A production $\pi(p)$, often called a *rewriting rule*, is usually denoted by a *span* $L \xleftarrow{l} K \xrightarrow{r} R$, and is often simply represented by the name p.

Definition 4.2 (Derivation). Let $p : L \xleftarrow{l} K \xrightarrow{r} R$ be a T-typed production. A *match* of p in a T-typed graph G is a morphism $m_L : L \to G$. A *direct derivation* from G to H via production p at a match m_L is a diagram as depicted in Figure 2, where (1) and (2) are pushout squares in T-**Graph**. In this case we write $p/m : G \Longrightarrow H$, for m the triple $\langle m_L, m_K, m_R \rangle$, or just simply $G \Longrightarrow H$.

We use $\Pi : G \Longrightarrow^* H$ to denote a sequence of direct derivations, which are constrained in the expected way, and $\Longrightarrow_{\mathscr{G}}$ for the reduction relation associated with a GTS $\mathscr{G}$.

Operationally, applying a production p to a graph G consists of three steps. First, the match $m_L : L \to G$ is chosen, providing an occurrence of L in G. Then, all the items of G matched by $L - l(K)$ are removed, leading to the context graph D. If D is well defined, and the resulting square is indeed a pushout, the items of $R - r(K)$ are added to D, further coalescing those nodes and edges identified by r, obtaining the derived graph H.

4.1. *Track function and interface morphisms*

We now turn our attention to the well-known notion of track function, which is a partial function identifying the items before and after a derivation.

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

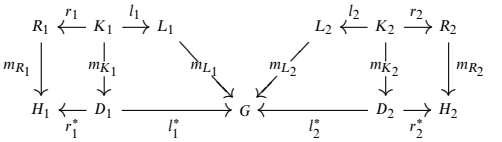

Fig. 3. Possible critical pair for direct derivations p_1/m_1 and p_2/m_2.

Definition 4.3 (Track function). Let p be a production and $p/m : G \Longrightarrow H$ be a direct derivation, as in Figure 2. The *track function* $tr(p/m)$ associated with the derivation is the partial function $r^* \circ (l^*)^{-1} : G \to H$.

We use the notion of trace to lift derivations to graphs with interfaces.

Definition 4.4 (Graph with interfaces derivation). Let $\mathbb{G} = R \overset{r}{\Rightarrow} G \overset{v}{\Leftarrow} V$ and $\mathbb{H} = R \overset{r'}{\Rightarrow} H \overset{v'}{\Leftarrow} V$ be graphs with the same interfaces, and let $p/m : G \Longrightarrow H$ be a direct derivation such that the trace function $tr(p/m)$ is total on $v(V)$ and $r(R)$. We say $p/m : \mathbb{G} \Longrightarrow \mathbb{H}$ is a direct derivation of graphs with interfaces if $r' = tr(p/m) \circ r$, that is, if r' is obtained by the composition of r with the track function $tr(p/m)$ (and similarly for v').

Hence, a derivation between graphs with interfaces is a direct derivation between the underlying graphs such that inputs and outputs are preserved.

We also use the track function to define the notion of confluence for GTS's. To this end, we consider its obvious extension $tr(\Pi)$ to a derivation Π, possibly of length zero.

Definition 4.5 (Strong (local) confluence). Let $\mathscr{G}$ be a GTS. The associated reduction relation $\Longrightarrow_{\mathscr{G}}$ is *strongly confluent* if for any two direct derivations $p_1/m_1 : G \Longrightarrow H_1$ and $p_2/m_2 : G \Longrightarrow H_2$ as in Figure 3, there exist two derivations $\Pi_1 : H_1 \Longrightarrow^* H$ and $\Pi_2 : H_2 \Longrightarrow^* H$ such that $tr(\Pi_1) \circ tr(p_1/m_1) = tr(\Pi_2) \circ tr(p_2/m_2)$.

Confluence is thus implied by the standard notion of *parallel independence*, as we will make explicit in Appendix A. Note, however, that the notion is stronger than the corresponding property in, for example, term rewriting, since the preservation of the track function implies not only that the two derivations reach the same graph, but that the items of the starting graph are preserved. In particular, this also implies that the interface morphisms are preserved.

5. From processes to graphs with interfaces

The aim of this section is to present a methodology for encoding processes into graphs.

The first step in our simulation of the π-calculus is to search for a suitable type graph, and then to exploit the composition operators defined previously to obtain an inductive encoding.

The type graph is defined in Figure 4. Note that all edges have at most one node in the source, which is connected by an incoming tentacle; on the other hand, the nodes in the target list are always enumerated in a clockwise direction, starting from the single

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

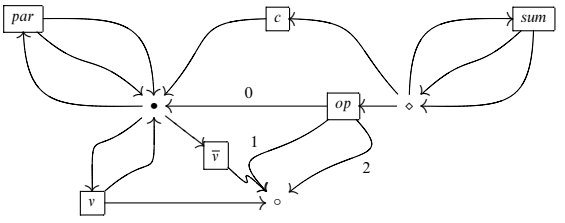

Fig. 4. The type graph (for $op \in \{in, out\}$).

incoming tentacle, unless otherwise specified by an enumerating label. For example, the edge v has the node $\bullet$ as source, and the node list $\langle \bullet, \circ \rangle$ as target. The edge op actually stands as a concise representation for two edges, namely *in* and *out*, with the same source and target: more precisely, they have the node $\diamond$ as source and the node list $\langle \bullet, \circ, \circ \rangle$ as target, as specified by the enumerating labels 0, 1 and 2.

The type graph is used to model processes syntactically, and our encoding corresponds to the usual construction of the tree associated with a term of an algebra: names are interpreted as variables, so they are mapped to leaves of the tree and can be shared safely. Intuitively, a tree with a node of type $\bullet$ or $\diamond$ as root corresponds to a process or summation, respectively, whilst each node of type $\circ$ basically represents a name. The set of edges contains an element for each operator of the calculus: it also includes the edge c for 'coercing' the occurrence of a summation within a process context (a standard device from algebraic specifications, see, for example, Goguen and Meseguer (1992)); and the edge $\overline{v}$, a restriction without continuation, which will be needed later.

A further step is the characterisation 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. Thus, we consider names $p, s \notin \mathcal{N}$: a first set actually includes an element for each edge of the type graph, and is presented in Figure 5 (except for the constant *sum*, which corresponds to the non-deterministic choice and is similar to *par*). Note also that *par* is a graph with interfaces $(\{p\}, \{p\} \uplus \{p\})$ and the disjoint union is conventionally represented by the pair $\{p_1, p_2\}$.

An additional set of constants is presented in Figure 6: these do not correspond to any operator of the signature, but represent some 'house-keeping' operations, which will be needed for our formal presentation of the encoding.

The constant new_x is the unique graph with interfaces $(\varnothing, \{x\})$ and with a bijection as output morphism, and similar characterisations hold for the other constants. Now, for a set Γ of names, we use id_Γ and new_Γ as shorthands for $\bigotimes_{o \in \Gamma} id_o$ and $\bigotimes_{o \in \Gamma} new_o$,

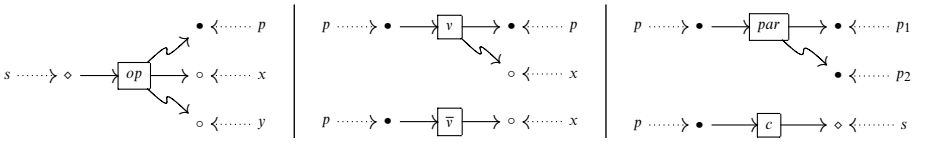

Fig. 5. Graphs $op_{x,y}$ (with $op \in \{in, out\}$); v_x and $\overline{v}_x$; and *par* and c (top to bottom and left to right).

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

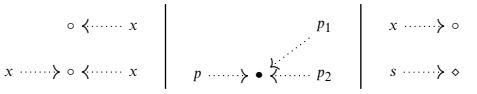

Fig. 6. Graphs new_x and id_x; Δ_p; and 0_x and 0_s (top to bottom and left to right).

$$
\begin{array}{lcll}
[\![M]\!]^p_\Gamma & = & c \circ [\![M]\!]^s_\Gamma & \\
[\![(vy)P]\!]^p_\Gamma & = & v_w \circ ([\![P\{^w/_y\}]\!]^p_{\{w\}\uplus\Gamma} \otimes id_w) \circ (0_w \otimes id_\Gamma) & \text{for } w \notin \Gamma \\
[\![P \mid Q]\!]^p_\Gamma & = & par \circ ([\![P]\!]^p_\Gamma \otimes [\![Q]\!]^p_\Gamma) & \\
[\![0]\!]^s_\Gamma & = & 0_s \otimes new_\Gamma & \\
[\![\overline{x}y.P]\!]^s_\Gamma & = & out_{x,y} \circ ([\![P]\!]^p_\Gamma \otimes id_{\{x,y\}}) & \\
[\![x(y).P]\!]^s_\Gamma & = & in_{x,w} \circ ([\![P\{^w/_y\}]\!]^p_{\{w\}\uplus\Gamma} \otimes id_{\{x,w\}}) \circ (0_w \otimes id_\Gamma) & \text{for } w \notin \Gamma \\
[\![M + N]\!]^s_\Gamma & = & sum \circ ([\![M]\!]^s_\Gamma \otimes [\![N]\!]^s_\Gamma) &
\end{array}
$$

Fig. 7. The encodings for processes.

respectively: they are well defined, as the ⊗ operator is associative. Finally, the encoding of processes into trees, which map each finite process into a graph expression, is introduced in the following definition.

Definition 5.1 (Encoding for processes). Let P be a process and Γ be a set of names such that $fn(P) \subseteq \Gamma$. The (mutually recursive) *encodings* $[\![P]\!]^p_\Gamma$ and $[\![M]\!]^s_\Gamma$, which map a process P into a graph, are defined by structural induction according to the rules in Figure 7 (where we assume the standard definition for name substitution).

As we have already observed, the encoding above merely puts the standard construction of a tree from a term into a graph expression, while solving sensible issues like α-conversion of bound names, which are denoted by ∘ nodes that are not in the image of the variable morphism. For example, the tree associated with the graph expression $[\![x(z).\overline{z}w.0 \mid \overline{x}x.0]\!]^p_{\{x,w\}}$ is represented in Figure 8. In the following, for any process P, its *tree encoding* is the graph $[\![P]\!]^p_{fn(P)}$, simply denoted as $[\![P]\!]^p$.

The mapping is well defined in the sense that the value of the resulting graph expression is independent of the choice of the name w in the rules for restriction and input prefix; moreover, given a process P and a set of names Γ such that $fn(P) \subseteq \Gamma$, the encoding $[\![P]\!]^p_\Gamma$ is a graph with interfaces $(\{p\}, \Gamma)$.

The mapping is not surjective, since there are graphs with interfaces $(\{p\}, \Gamma)$ that are not in the image of the encoding. As an example, consider the graph in Figure 9, which represents a name-sharing situation that is not allowed in the process construction, where the name y, which should be local to the process (possibly) occurring below the input prefix $x(y)$, is instead shared and made visible globally.

6. Process equivalence from graph rewrites

The previous section introduced a graphical encoding for processes. The aim of this section is to show that an alternative encoding of a process can be obtained from its tree

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

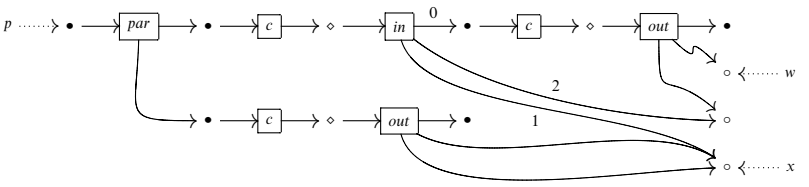

Fig. 8. The tree encoding for $x(z).\overline{z}w.0 \mid \overline{x}x.0$.

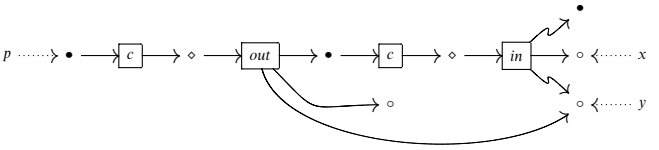

Fig. 9. A graph with a forbidden name-sharing situation.

encoding *via* a set of transformation rules, which is tailored to the need to recast process equivalence in terms of graph isomorphism.

So, we introduce the GTS R^n_π, which contains the five rules depicted in Figures 10, 11 and 12. We will now describe the two rules for removing the occurrences of the parallel and choice operators, while coalescing their continuations, represented in Figure 10. Consider the leftmost: its three components are represented by three graphs, separated by a vertical line. The span of the graph morphisms is not presented explicitly: since the morphism on the left-hand side has to be mono and the right-hand side only contains a node, the span is uniquely chosen (and the morphism on the right-hand side coalesces the three nodes).

Another rule is needed to detach restriction, by substituting the occurrence of the ν operator with $\overline{\nu}$: it is shown in Figure 11.

The final two rules are needed to remove the useless occurrences of the restriction and coercion operators, and are shown in Figure 12. Note that the occurrence of, for example, a restriction operator is useless if the name it binds does not occur in the process: the rule implements this exactly since it cannot be applied unless the node representing the name is isolated.

Proposition 6.1 (Confluent encoding). The reduction relation $\Longrightarrow_{\mathscr{R}^n_\pi}$ induced by the GTS $\mathscr{R}^n_\pi$ is strongly confluent.

The proof is straightforward since the rules are pairwise parallel independent (see Appendix A). First, they share no edge. Moreover, they usually preserve nodes, the only exception being the rule for removing restriction and coercion. However, the rule for removing, for example, the coercion operator (left-hand side of Figure 12) can never form a critical pair with the rule for removing the sum operators (left-hand side of Figure 10), since in order for the former to be applied, the unique node $\diamond$ must not be connected to any other edge (according to the so-called *dangling condition*, see, for example, Corradini *et al.* (1997)).

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

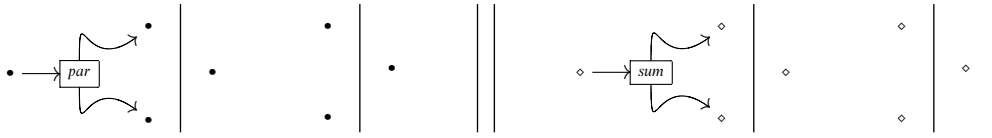

Fig. 10. The rules for removing the parallel (left-hand side) and sum (right-hand side) operators.

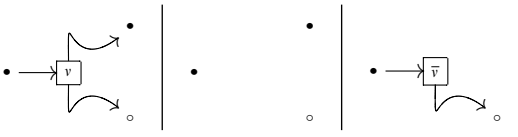

Fig. 11. The rule for substituting the restriction operator.

Fig. 12. The rules for removing the restriction (left-hand side) and coercion (right-hand side) operators.

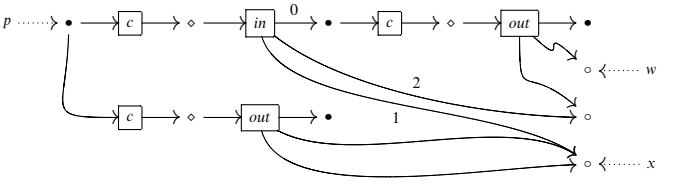

Fig. 13. Normal form of the tree encoding for $x(z).\bar{z}w.0 \mid \bar{x}x.0$.

We use $nfp(\mathbb{G})$ to denote the normal form associated with a graph with interfaces $\mathbb{G}$, which is unique up-to isomorphism.

Theorem 6.1 (Encoding preserves congruence). Let P, Q be processes and Γ be a set of names such that $fn(P) \cup fn(Q) \subseteq \Gamma$. Then, $P \equiv Q$ (that is, the two processes are equated by the set of axioms in Figure 1) if and only if $nfp([\![P]\!]^p_\Gamma) = nfp([\![Q]\!]^p_\Gamma)$.

The proof can be found in Appendix B.

Consider the process $P = x(z).\bar{z}w.0 \mid \bar{x}x.0$ again. Its tree encoding was presented in Figure 8 (left-hand side), and Figure 13 depicts its normal form, which is obtained by removing the only *par* edge and coalescing three nodes via an application of the rule in Figure 10. Note that its normal form is, for example, the same as $P \mid 0$, which is obtained by an additional application of the rule for removing the parallel operator together with an application of the rule in Figure 12 (right-hand side) for removing the useless coercion operator.

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

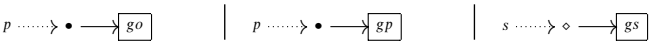

Fig. 14. Graphs *go*, *gp* and *gs* (from left to right).

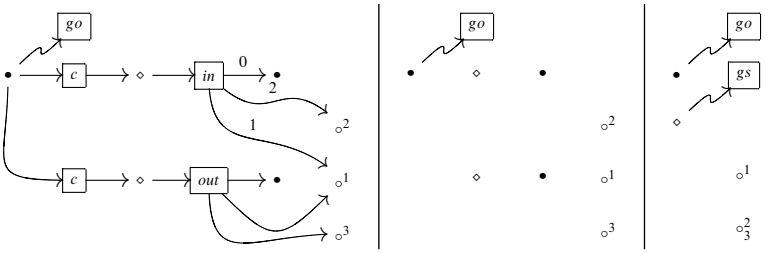

Fig. 15. The rule p_π for synchronisation in $\mathscr{R}^a_\pi$.

7. Reductions versus graph rewrites

In this section we introduce the GTS $\mathscr{R}_\pi$, and show how it simulates the reduction semantics for processes, and the removal of those sub-processes discarded after a reduction step.

First, we will enrich the signature with three additional edges: Figure 14 gives a direct presentation of the associated constants needed in the graph expressions.

The edge *go* is just a syntactical device for detecting the 'entry' point of the computation, thus avoiding the need to perform reductions below the outermost prefix operators; whilst the edges *gp* and *gs* are needed for detecting those parts of the graphical encoding (rooted in by either a process node or a sum node, hence the acronyms for *g*arbage *p*rocesses and *g*arbage *s*ummations) that must be discarded after the reduction has taken place, thus implementing a *garbage collection* phase.

Now, the GTS R^a_π for simulating the reduction steps contains just one rule, shown in Figure 15. The action of the rule on those nodes representing names is described by using an explicit label, so the $\circ$ nodes identified by 2 and 3 are coalesced by the rule. The node identifiers are of course arbitrary: they correspond to the actual elements of the set of nodes and are just used to characterise the span of functions.

It seems noteworthy that just one rule is needed to recast the reduction semantics for the π-calculus. First, the structural rules are taken care of by the fact that graph morphisms allow the embedding of a graph within a larger graph, thus simulating the closure of reduction with respect to contexts. Second, no distinct instance of the rule is needed: a particular instance of the process $x(y).P + M \mid \overline{x}w.Q + R$ is chosen to represent the synchronisation, while graph isomorphism takes care of the closure with respect to structural congruence, and the interfaces of the renaming of free names.

Finally, note that, even if the search for a match can be considered a global operation, rule application itself is a local operation, coalescing a few nodes and removing four edges.

Some components of a process may be discarded during the reduction. This forces us to perform some additional steps for garbage collection, basically restricting to the graph reachable from the source node of the unique *go* edge. This is implemented by

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

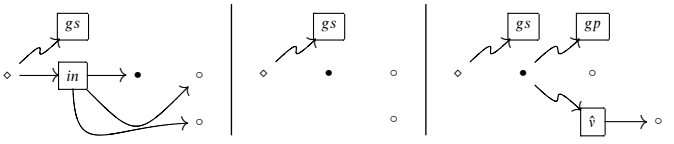

Fig. 16. The rule for the garbage collection of input in $\mathscr{R}_\pi^g$.

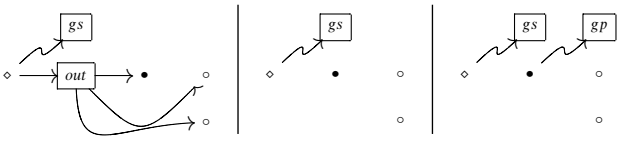

Fig. 17. The rule for the garbage collection of output in $\mathscr{R}_\pi^g$.

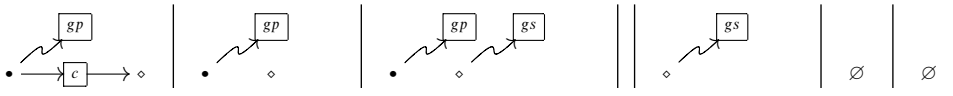

Fig. 18. The rules for the garbage collection of coercion and of summation nodes in $\mathscr{R}_\pi^g$.

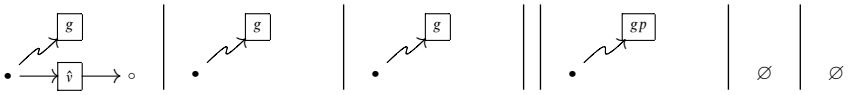

Fig. 19. The rules for the garbage collection of restriction and of process nodes in $\mathscr{R}_\pi^g$ (for $g \in \{g0, g3\}$).

adding explicit rules for removing useless edges. The GTS $\mathscr{R}_\pi^g$ thus contains seven rules: in particular, the rules for the garbage collection of to-be-removed prefixes are depicted in Figure 16 and Figure 17; the rule for coercion is depicted in Figure 18 (left-hand side), and the similar rules for restriction are presented in Figure 19 (left-hand side).

Proposition 7.1 (Confluent garbage collection). The reduction relation $\Longrightarrow_{\mathscr{R}_\pi^g}$ induced by the GTS $\mathscr{R}_\pi^g$ is strongly confluent.

As with Proposition 6.1, the proof is straightforward since the rules are pairwise parallel independent. All the possibly shared edges are preserved, except for the rules collecting sum and process nodes: once again, the dangling condition assures us that, for example, the rules for removing a prefix (Figures 16 and 17) may never form a critical pair with the rule for removing a summation node (right-hand side of Figure 18).

The intuition is that after each reduction step, the garbage collection is performed by first removing all the input and output edges; then, those ○ nodes representing bound names that are not referred to anymore by the process are also removed; and, finally, the

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

isolated sum and process nodes are discarded. Note the introduction of a restriction edge after the removal of an input edge: since each node is bound at most once, this ensures that the node is removed with the rule for the restriction operator after all the edges using that node (that is, all the input and output edges using the associated name) are discarded.

Let $nfg(\mathbb{G})$ denote the normal form associated with a graph with interfaces $\mathbb{G}$, which is unique up-to isomorphism. Moreover, let the symbol $[\![P]\!]_\Gamma$ denote the graph with interfaces $\Delta_p \circ ([\![P]\!]^p_\Gamma \otimes go)$, which can be further simplified to $[\![P]\!]$ whenever $\Gamma = fn(P)$. We now state the main theorems of the paper, which concern the soundness and completeness of our encoding with respect to the reduction semantics.

Theorem 7.1 (Encoding preserves reductions). Let P, Q be processes and Γ be a set of names such that $fn(P) \subseteq \Gamma$. If $P \to Q$, then $\mathscr{R}^a_\pi$ entails a direct derivation $nfp([\![P]\!]_\Gamma) \Longrightarrow \mathbb{G}$ via an edge-preserving match, and $nfg(\mathbb{G}) = nfp([\![Q]\!]_\Gamma)$.

Intuitively, a reduction step is simulated by applying a rule on an enabled event, that is, by finding a match covering a sub-graph with the *go* operator on top. Then, the garbage collection phase removes those items corresponding to the sub-processes that are discarded after the resolution of non-deterministic choices.

Theorem 7.2 (Encoding reflects reductions). Let P be a process and Γ be a set of names such that $fn(P) \subseteq \Gamma$. If $\mathscr{R}^a_\pi$ entails a direct derivation $nfp([\![P]\!]_\Gamma) \Longrightarrow \mathbb{G}$ via an edge-preserving match, then there exists a process Q such that $P \to Q$ and $nfg(\mathbb{G}) = nfp([\![Q]\!]_\Gamma)$.

As noted earlier, the correspondence holds since the presence of the *go* operator forces the match to be applied only to operators on top, thus forbidding the occurrence of a reduction inside the outermost prefix operators.

The restriction to edge-preserving matches is necessary in order to ensure that the two edges of the rule labelled by c can never be merged together. Intuitively, allowing for their coalescing would correspond to the synchronisation of two summations, that is, as allowing for a reduction $x(y).P + \overline{x}w.Q \to P\{^w/_y\} \mid Q$. Instead, it is necessary to allow for the coalescing of nodes, in order to recover those synchronisations with output prefix $\overline{x}x$, as in Figure 20 below.

The proofs can be found in Appendices C and D, respectively.

Example 7.1 (Rule application). Let P be the process $x(z).\overline{z}w.0 \mid \overline{x}x.0$. The associated tree encoding $[\![P]\!]^p$ is depicted in Figure 8, and the graph with interfaces $nfp([\![P]\!]^p)$ is represented in Figure 13. The graph with interfaces $nfp([\![P]\!])$ is concisely represented on the left of Figure 20: those nodes in the image of either the input or the output morphism are so denoted by the label (either p or one of the free names $\{x, w\}$), while the use of integers as labels on the remaining nodes is used to denote the track function associated with the derivation.

The application of a rewriting step, resulting in the graph on the right, simulates the transition $x(z).\overline{z}w.0 \mid \overline{x}x.0 \to \overline{x}w.0$. In fact, after removing the $\diamond$ node labelled by 1 (and 2), and the connected *gs* edge, the resulting graph with interfaces is $nfg([\![\overline{x}w.0]\!])$.

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

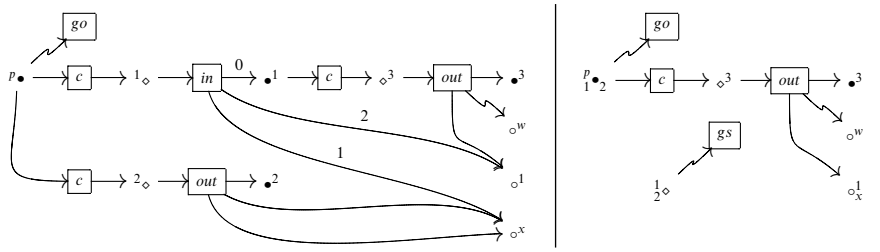

Fig. 20. A rewriting step, simulating the transition $x(z).\bar{z}w.0 \mid \bar{x}x.0 \rightarrow \bar{x}w.0$.

8. Some remarks on the use of the encoding

In this section we collect together some of the applications that have been devised so far by exploiting the technique for simulating process reduction, by means of a graphical encoding, that we have proposed and instantiated in this paper for the π-calculus.

8.1. *An implementation technique*

The graphical technique proposed in this paper proceeds in two steps.

— First, the tree encoding for processes is considered (Section 5), and a GTS $\mathscr{R}_\pi^n$ for recovering a normal form $nfp([\![P]\!]_\Gamma^p)$ for the graphical encoding of a process P is given (Section 6).
— Then, rules for simulating process reduction, as well as garbage collection, are considered (Section 7).

The normal form $nfp([\![P]\!])$ plays a pivotal role with respect to process reduction, and the theorems of Section 7 reflect this fact: indeed, in previous papers this normal form was characterised explicitly by means of a suitable encoding (Gadducci and Montanari 2001; Gadducci 2003). Nevertheless, the tree encoding suggests an alternative approach to the simulation of the reduction semantics, as made explicit by the proposition below.

Proposition 8.1 (Confluent implementation). Let $\mathscr{R}_\pi^i$ be the GTS obtained as the union of the GTS's $\mathscr{R}_\pi^n$ and $\mathscr{R}_\pi^g$. Then, the reduction relation $\Longrightarrow_{\mathscr{R}_\pi^i}$ induced by the GTS $\mathscr{R}_\pi^i$ is strongly confluent.

This result is easily proved. In fact, all the rules of this new GTS are pairwise parallel independent, with the exception of the rules for removing coercion and restriction: Figure 12 (left) and (right) versus Figure 19 (left) and Figure 18 (left), respectively. However, consider the removal of the coercion operator: the application of the rule of Figure 12 can be simulated by applying in sequence, from left to right, the rules of Figure 18, so the possible critical pair arising from the rules removing coercion is eliminated. A similar consideration also holds for the removal of the restriction operator, and thus strong confluence immediately follows.

Thus, the GTS $\mathscr{R}_\pi$, obtained as the union of $\mathscr{R}_\pi^i$ and $\mathscr{R}_\pi^a$, can be considered as a graphical implementation of the reduction semantics for the π-calculus, turning the congruence closure into a set of rewriting rules: the GTS $\mathscr{R}_\pi$ allows at the same time the normalisation

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

of a process and the execution of a reduction step, as well as the garbage collection of its discarded components.

Proposition 8.2 (Compatible reductions). Let P be a process, $[\![P]\!] \Longrightarrow^* \mathbb{G}$ be a derivation in $\mathscr{R}_\pi$ and p be a rule in $\mathscr{R}^i_\pi$ such that there exist two direct derivations $p/m_1 : \mathbb{G} \Longrightarrow \mathbb{H}_1$ and $p_\pi/m_2 : \mathbb{G} \Longrightarrow \mathbb{H}_2$ for p_π the rule in $\mathscr{R}^a_\pi$. Then there exist two direct derivations $p_\pi/m_3 : \mathbb{H}_1 \Longrightarrow \mathbb{H}$ and $p/m_4 : \mathbb{H}_2 \Longrightarrow \mathbb{H}$ such that $tr(p_\pi/m_3) \circ tr(p_1/m_1) = tr(p/m_4) \circ tr(p_\pi/m_2)$.

Consider a graph $\mathbb{G}$ that is reachable from the encoding of a process. The key remark is that there can be no occurrence in $\mathbb{G}$ of either a *gs* or a *gp* edge in the sub-graph that is rooted in the source node of the unique occurrence of the *go* edge. Hence, any two direct derivations as above starting from $\mathbb{G}$ have to be parallel independent.

A corresponding compatibility result cannot be recovered for the syntactical representation of processes: indeed, the ACI (associativity, commutativity and identity) axioms of the two binary operators make it difficult to obtain a rewriting system performing process reduction and implementing in a confluent way the laws of structural congruence.

8.2. *On concurrent reductions*

The role of the *interface graph* K in a rewriting rule (see Figure 2) is to characterise in the graph to be rewritten those items that are read but not consumed by a direct derivation. Such a distinction is important when considering *concurrent* derivations. Concurrent derivations are defined as equivalence classes of concrete derivations up to so-called *shift equivalence* (see, for example, Corradini *et al.* (1997)) identifying (as for the analogous and better-known *permutation equivalence* of the λ-calculus) those derivations that differ only in the scheduling of independent steps. Roughly speaking, the equivalence states the interchangeability of two direct derivations $p_1/m_1 : \mathbb{G} \Longrightarrow \mathbb{H}$ and $p_2/m_2 : \mathbb{H} \Longrightarrow \mathbb{I}$ if they act either on disjoint parts of $\mathbb{G}$, or on parts that are in the image of the interface graphs.

Thus, our encoding may be exploited for the definition of a concurrent reduction semantics of the π-calculus. The presence of the operator *go* on the interface graph allows for the simultaneous execution of more than one reduction, since its sharing implies that more than one pair of (distinct) input and output resources may synchronise. The removal of the two edges labelled by *c* ensures that simultaneous reductions must not include resources that occur inside a subgraph whose root is a summation node.

The idea of exploiting the concurrency features offered by the graphical encoding of processes (of the ambient calculus) was originally suggested in Gadducci and Montanari (2001). However, most of the classical results in the literature, such as the characterisation of shift-equivalent derivations by means of *graph processes*, were restricted to GTS's with linear rules, that is, such that both morphisms of the span are injective. An extension of the classical concurrent semantics for DPO, in order to include the possibility of non-linear rules, has been considered in Gadducci and Montanari (2005), with an application to the *solo calculus*. More recently, a new formalism, *graph with equivalences*, has been proposed as a means for recovering an event structures semantics for those GTS's that are well suited to modelling process calculi (Baldan *et al.* 2006).

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

8.3. *Verifying spatial logics*

A recent series of papers advocated *spatial logics* as a suitable formalism for expressing behavioural and spatial properties of system specifications, which are often given as processes of a calculus. Besides the temporal modalities of the Hennessy–Milner tradition, these logics include operators for reasoning about the structural properties of a system. For example, the connective `void` represents the (processes structurally congruent to the) empty system, and the formula $\phi_1|\phi_2$ is satisfied by those processes that can be decomposed into two parallel components satisfying ϕ_1 and ϕ_2, respectively. Moreover, these logics come equipped with mechanisms for reasoning about the names occurring in a system.

There are several approaches to the verification of spatial properties for logics for either process calculi (see, for example, Caires (2004) and Caires and Cardelli (2003), and the references therein) or for other data structures. The results for the graphical encoding presented in this paper have been exploited to introduce a novel approach to the verification of (finite) spatial formulae (Caires 2004) for π-calculus specifications. First, an algorithm for model checking spatial formulae based on the analysis of the (normal form of the) graphical representation of processes was presented (Gadducci and Lluch Lafuente 2006). An encoding of (possibly recursive) formulae in a spatial logic for processes into formulae in a temporal graph logic (a modal extension of Courcelle's monadic second-order logic (Courcelle 1997)) was then considered: the encoding is correct, that is, a process verifies a spatial formula exactly when its graphical representation verifies the translated formula (Gadducci and Lluch Lafuente 2007).

8.4. *Distilling labelled transition systems*

Reduction semantics has the advantage of conveying the semantics of calculi with relatively few compact rules. Its main drawback is poor compositionality, in the sense that the dynamic behaviour of arbitrary stand-alone terms (like $x(y).P$) can be interpreted only by inserting them in the appropriate context (that is, $[\_] \mid \overline{x}w.Q$) where a reduction may take place. In other words, reduction semantics is often less suitable when specific behaviours other than confluence (such as termination or reachability) are of interest. In fact, simply using the reduction relation to define equivalences between components (for example, in terms of *bisimulation*) fails to give a compositional framework, and in order to recover a suitable notion of equivalence, it is often necessary to verify the behaviour of single components under any viable execution context.

A standard way out of the impasse, by reducing the complexity of such analyses, is to express the behaviour of a computational device using a *labelled transition system* (LTS). Provided the label associated with 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.

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

The graphical techniques proposed in the paper have been successfully employed to synthesise suitable LTS's for process calculi. In fact, graphs with interfaces are amenable to the synthesis mechanism based on *borrowed contexts* (BC's), which were proposed by Ehrig and König (Ehrig and König 2004), and which are in turn an instance of *relative pushouts*, which were originally introduced by Milner and Leifer (Leifer and Milner 2000). 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. The approach has been initially tested against the π-calculus (Gadducci and Montanari 2005); and it has been fully exploited for the simpler CCS to prove that the bisimilarity on (encodings of possibly recursive) processes obtained via BC's coincides with the standard strong bisimilarity for that calculus (Bonchi *et al.* 2006).

9. Dealing with recursion

The introduction of a *replication* operator, denoted by !, is now almost considered the standard approach for dealing with infinite processes. Intuitively, the behaviour of a process $!P$ coincides with the behaviour of a (possibly) unbound number of instances of P itself, that is, $P \mid P \mid \ldots$. Unfortunately, it seems hard to describe the behaviour of processes including replication within our graphical encoding, since this operator, even in the input-guarded $!x(y).P$ version, implicitly represents a global operation involving the duplication of necessarily unspecified sub-processes, so it is hard to model *via* graph rewriting, which is an eminently local process.

Nevertheless, it should be obvious that our framework allows for the modelling of *recursive* processes, that is, of processes defined using constant invocation, which is equivalent to the use of replication (as originally stated in Milner (1993, page 212)). Each process is compiled into a GTS, and new rules are added for the simulation of the *unfolding* steps in addition to the two rules synthesised by Figure 15.

9.1. *On recursive processes*

In this section we present recursive processes and their reduction semantics – see, with minor variations, Sangiorgi and Walker (2001, Section 3.2).

Definition 9.1 (Recursive process expressions). Let $\mathcal{N}$ be a set of *names* ranged over by $x, y, w, \ldots$; let $\Delta = \{x(y), \overline{x}y \mid x, y \in \mathcal{N}\}$ be the set of *prefix operators* ranged over by δ; and let $\mathcal{I}$ be a set of *process identifier* ranged over by $A, B, C, \ldots$.

A *(recursive) process expression* P is a term generated by the (mutually recursive) syntax

$$P ::= M, \quad (\nu x)P, \quad P_1 \mid P_2, \quad A\langle x_1, \ldots, x_n\rangle$$

$$M ::= 0, \quad \delta.P, \quad M_1 + M_2 .$$

We let $P, Q, R, \ldots$ range over the set *RProc* of process expressions, and $M, N, O, \ldots$ range over the set *RSum* of summation expressions.

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

We assume the standard definitions for the set of free names of a process expression (just stating that $fn(A\langle x_1,\ldots,x_n\rangle) = \{x_1,\ldots,x_n\}$), and for (capture avoiding) substitution and α-conversion.

Definition 9.2 (Recursive processes). Let $\mathcal{N}$ be a set of names ranged over by $x, y, w, \ldots$, and $\mathcal{I}$ be a set of process identifiers ranged over by $A, B, C, \ldots$.

A *recursive* process ϕ is a finite set of equations (at most one for each process identifier A) of the following kind

$$A(x_1,\ldots,x_n) =_\phi P_A$$

where the x_i are distinct names and P_A is a process expression with

$$fn(P_A) \subseteq \{x_1,\ldots,x_n\}.$$

Intuitively, an equation corresponds to a procedure definition (introducing formal parameters within round brackets), and each identifier in a process expression represents a procedure invocation (actual parameters within angle brackets).

As with finite processes, we may define the behaviour of a process ϕ as a relation over abstract process expressions, though the reduction relation is now parametric with respect to the recursive process.

Definition 9.3 (Recursive reduction semantics). The reduction relation for a recursive process ϕ is the relation $R_\pi^\phi \subseteq RProc \times RProc$, which is closed under the congruence $\equiv$ induced by the set of axioms in Figure 1, that is inductively generated by the following set of axioms and inference rules:

$$\overline{x(y).P + M \mid \bar{x}w.Q + N \rightarrow_\phi P\{^w/_y\} \mid Q}$$

$$\frac{P \rightarrow_\phi Q}{(\nu x)P \rightarrow_\phi (\nu x)Q} \qquad \frac{P \rightarrow_\phi Q}{P \mid R \rightarrow_\phi Q \mid R}$$

$$\frac{A(x_1,\ldots,x_n) =_\phi P_A}{A\langle y_1,\ldots,y_n\rangle \rightarrow_\phi P_A\{^{y_1}/_{x_1},\ldots,^{y_n}/_{x_n}\}}$$

where $P \rightarrow_\phi Q$ means that $\langle P, Q\rangle \in R_\pi^\phi$.

9.2. *From recursive processes to graphs*

In the definition of reductions semantics given in the previous section, a different reduction system was associated with each recursive process. It then seems natural that, in order to encode recursive processes into graphs, we associate a distinct graph transformation system with each process to simulate its behaviour.

We now extend the type graph by adding an edge A_n for each equation identifier A and natural number n with the node • as source and n outgoing tentacles reaching the node ○. The resulting type graph is, of course, infinite, but the set of edges with different types occurring in each recursive process is actually finite.

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

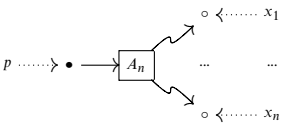

Fig. 21. The graph with interfaces $A_{x_1,\ldots,x_n}$.

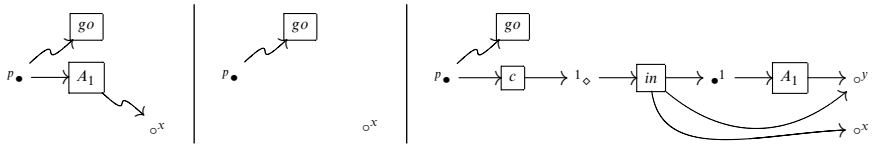

Fig. 22. The rule p_A^ϕ for $A(x) =_\phi x(y).A\langle y\rangle$.

We must also extend the class of graphs to be used as constants in order to simulate process expressions by graph expressions. More precisely, we need graphs like the one depicted in Figure 21, which, intuitively, represent $A\langle x_1,\ldots,x_n\rangle$.

Remember that new_Γ is a shorthand for $\bigotimes_{o\in\Gamma} new_o$ for a set of names Γ. The encoding of process expressions into graphs is then introduced in the following definition.

Definition 9.4 (Encoding for process expressions). Let P be a process expression and Γ be a set of names such that $fn(P) \subseteq \Gamma$. The *graph encoding* $[\![P]\!]^p_\Gamma$, which maps a process expression P into a graph, is given by extending the encoding of Definition 5.1 with the following rule:

$$[\![A\langle x_1,\ldots,x_n\rangle]\!]^p_\Gamma \;=\; A_{x_1,\ldots,x_n} \otimes new_\Gamma\,.$$

The mapping is well defined, and, given a set of names Γ, the encoding $[\![P]\!]^p_\Gamma$ of a process expression P is a graph (actually, a tree) with interfaces $(\{p\},\Gamma)$. It is still sound and complete, as stated by the proposition below, which extends Proposition 6.1.

Proposition 9.1. Let P, Q be process expressions and Γ be a set of names, such that $fn(P) \cup fn(Q) \subseteq \Gamma$. Then, $P \equiv Q$ if and only if $nfp([\![P]\!]^p_\Gamma) = nfp([\![Q]\!]^p_\Gamma)$.

The result follows from Theorem 6.1: a process identifier behaves as an additional constant since it is ininfluent with respect to the laws of structural congruence.

9.3. *From recursive processes to graph transformation systems*

We introduce for each recursive process ϕ the GTS $\mathscr{R}^\phi_\pi$, which extends $\mathscr{R}^a_\pi$, described in Section 7, by adding an unfolding rule p^ϕ_A for each equation $A(x_1,\ldots,x_n) =_\phi P_A$ in ϕ. Intuitively, the production is given by restricting the span

$$A_{x_1,\ldots,x_n} \longleftarrow go \otimes new_\Gamma \longrightarrow nfp([\![P_A]\!]_\Gamma)$$

for $\Gamma = \{x_1,\ldots,x_n\}$ to the underlying graphs. Consider, for example, a process ϕ containing the equation $A(x) =_\phi x(y).A\langle y\rangle$: the unfolding rule p^ϕ_A is shown in Figure 22.

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

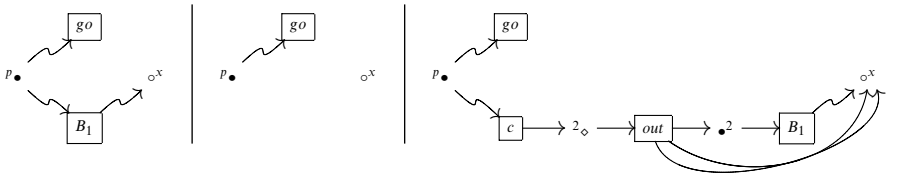

Fig. 23. The rule p_B^ϕ for $B(x) =_\phi \overline{x}x.B\langle x\rangle$.

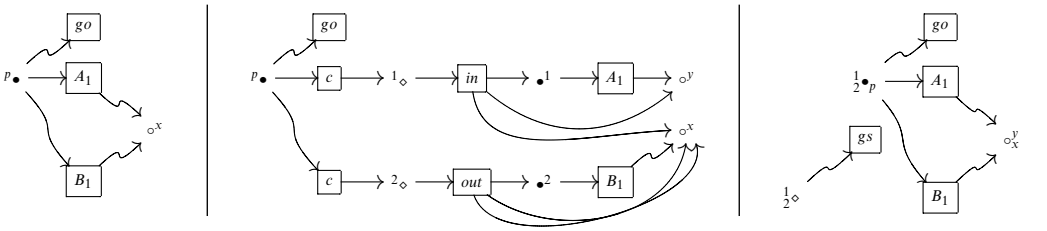

Fig. 24. Unfolding $[\![A\langle x\rangle \mid B\langle x\rangle]\!]$ to $[\![x(y).A\langle y\rangle \mid \overline{x}x.B\langle x\rangle]\!]$, then synchronising.

Now, we add to $\mathscr{R}_\pi^g$ a new rule for each process identifier, which is used to remove it whenever a *gp* edge is attached to the source of the identifier: the resulting GTS, which is still denoted $\mathscr{R}_\pi^g$, is also strongly confluent.

We can then extend our soundness and completeness results to recursive processes.

Proposition 9.2 (Encoding the reductions). Let ϕ be a recursive process, P be a process expression and Γ be a set of names such that $\mathit{fn}(P) \subseteq \Gamma$.

Let Q be a process expression. If $P \to_\phi Q$, then $\mathscr{R}_\pi^\phi$ entails a direct derivation $\mathit{nfp}([\![P]\!]_\Gamma) \Longrightarrow \mathbb{G}$ via an edge-preserving match, and $\mathit{nfg}(\mathbb{G}) = \mathit{nfp}([\![Q]\!]_\Gamma)$.

For the other direction, if $\mathscr{R}_\pi^\phi$ entails a direct derivation $\mathit{nfp}([\![P]\!]_\Gamma) \Longrightarrow \mathbb{G}$ via an edge-preserving match, there exists a process expression Q such that $P \to_\phi Q$ and $\mathit{nfg}(\mathbb{G}) = \mathit{nfp}([\![Q]\!]_\Gamma)$.

As with Proposition 9.1, the result is a consequence of Theorems 7.1 and 7.2.

Example 9.1 (Mapping a recursive process). Consider the process expression $P = A\langle x\rangle \mid B\langle x\rangle$, for the process ϕ defined by

$$A(x) =_\phi x(y).A\langle y\rangle \qquad\qquad B(x) =_\phi \overline{x}x.B\langle x\rangle.$$

The unfolding rule p_B^ϕ is presented in Figure 23.

How is the reduction $P \to_\phi P$ simulated? The graph on the left of Figure 24 represents $\mathit{nfp}([\![P]\!])$. The graph in the centre represents the situation after the (possibly simultaneous) application of both the unfolding rules p_A^ϕ and p_B^ϕ. Now, an application of the rule for synchronisation does the trick, giving the graph on the right: after the garbage collection phase, the resulting graph is isomorphic to the initial state.

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

10. Conclusions and further work

In this paper we have presented an encoding of (possibly recursive) processes of the π-calculus into (typed hyper-)graphs, and have proved that it is sound and complete with respect to the original reduction semantics.

The key technical point is the use of nodes as place-holders for names, and, in particular, the use of a graph morphism to represent the free names occurring in a process. Our solution avoids the need for any encapsulation in the encoding of processes, and it allows us to capture the full extent of the calculus, including recursion and non-deterministic choice. Moreover, the use of unstructured graphs allows the reuse of standard graph rewriting theory and tools for simulating the reduction semantics of the calculus. We consider this an advance on most of the other approaches for the graphical implementation of calculi with name mobility (such as Milner's *bigraphs* (Milner 2001)), where *ad hoc* mechanisms for graph rewriting have to be developed.

Our presentation of the finite fragment of the π-calculus is reminiscent of the encoding for mobile ambients in Gadducci and Montanari (2001). This fact strengthens our belief that any calculus with name mobility may find a presentation within our formalism along the lines of the encodings in these two papers. The calculus should contain a parallel operator that is associative, commutative and with an identity; also, its operational semantics should be reduction-like (that is, *via* unlabelled transitions), and the rules should not substitute a free name for another, so that name substitution is handled by node coalescing. The mechanism is reminiscent of *name fusion*, and hence the resemblance of our graphical encoding to solo diagrams.

Our encoding of process reduction represents a starting point, and we plan to test its efficiency by using it as a preprocessing step in one of the many existing tools for implementing graph rewriting. On the theoretical side, we have sketched out a list of ongoing applications in Section 8: in particular, we plan to continue the study of the observational equivalences induced on processes by the encoding according to the relative pushouts approach.

Appendix A. Parallel independence

This appendix introduces the classical notion of parallel independence, and states its connection with strong confluence. It is a mild generalisation of Plump (2005, Section 3.3).

Definition A.1 (Parallel independence). Let $p_1/m_1 : G \Longrightarrow H_1$ and $p_2/m_2 : G \Longrightarrow H_2$ be two direct derivations as in Figure 25. These derivations are *parallel independent* if there exists an *independence pair* amongst them, that is, two graph morphisms $i_1 : L_1 \to D_2$ and $i_2 : L_2 \to D_1$ such that $l_2^* \circ i_1 = m_{L_2}$ and $l_1^* \circ i_2 = m_{L_1}$.

Intuitively, two derivations as in Figure 25 are parallel independent if they act on disjoint items of the graph G, or at least on items that are simply read, and thus not deleted, by any of the two rule applications. The proposition below is a classical result relating parallel independence to rule sequentialisation (see, for example, Corradini *et al.* (1997)).

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

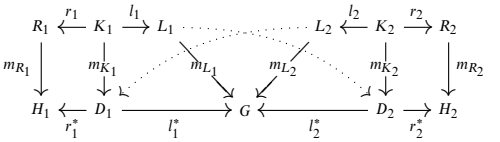

Fig. 25. Parallel independence for $p_1/m_{L_1} : G \Longrightarrow H_1$ and $p_2/m_{L_2} : G \Longrightarrow H_2$.

Proposition A.1 (Confluence from parallel independence). Let $p_1/m_1 : G \Longrightarrow H_1$ and $p_2/m_2 : G \Longrightarrow H_2$ be two direct derivations as in Figure 25 such that they are parallel independent with independence pair $i_1 : L_1 \to D_2$ and $i_2 : L_2 \to D_1$. Then there exists a graph H and two derivations $p_2/m_2^* : H_1 \Longrightarrow H$ with match $r_2^* \circ i_2$ and $p_1/m_1^* : H_2 \Longrightarrow H$ with match $r_1^* \circ i_1$ such that $tr(p_2/m_2^*) \circ tr(p_1/m_1) = tr(p_1/m_1^*) \circ tr(p_2/m_2)$.

We say that two rules are parallel independent if for any two possible matches, the corresponding direct derivations are parallel independent, and a GTS is parallel independent if its productions are pairwise parallel independent. It is then clear, thanks to the result above, that the reduction relation in a parallel independent GTS is strongly confluent.

Proposition A.2 (Confluent reductions). If $\mathscr{G}$ is a parallel independent GTS, the associated reduction relation $\Longrightarrow_{\mathscr{G}}$ is strongly confluent.

Appendix B. Process equivalence versus graph isomorphism

In this appendix we provide a proof sketch of Proposition 6.1 of Section 5, which links finite processes and graphs with interfaces. The correspondence itself is split into two parts: soundness is proved, basically, by induction on the tree encoding proposed in Definition 5.1, and completeness is shown by analysing the structure of the graphs in the image of the tree encoding, and of their normal forms.

We begin by stating a pair of useful lemmas.

Lemma B.1. Let P, Q be processes. If $P \equiv Q$, then $fn(P) = fn(Q)$.

The next lemma proves that it is enough to restrict attention to encodings with respect to the set of free names of a process.

Lemma B.2. Let P be a process and Γ be a set of names such that $fn(P) \subseteq \Gamma$. Then, $[\![P]\!]^p_\Gamma = [\![P]\!]^p \otimes new_\Gamma$.

The result above is proved by induction on the cardinality of Γ. The key point, which is proved by induction on the encoding of P, is that for each $x \in \Gamma$, the name x is mapped into an isolated node if and only if $x \notin fn(P)$.

The pivotal lemma below states that derivations are preserved under closure with respect to graph contexts, which, intuitively, are defined as graph expressions 'with a hole'.

Lemma B.3. Let $\mathbb{G}$ a graph with discrete interfaces and $C[-]$ be a graph context such that the graph expression $C[\mathbb{G}]$ is well defined. If $\mathscr{R}_\pi$ entails a direct derivation $\mathbb{G} \Longrightarrow \mathbb{H}$, then it also entails a direct derivation $C[\mathbb{G}] \Longrightarrow C[\mathbb{H}]$.

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

Sketch of proof. The proof is by induction on $C[-]$, that is, on the number of composition operators occurring in it. Let $\mathbb{G} = I \Rightarrow G \Leftarrow J$ be a graph with interfaces, $p : L \leftarrow K \rightarrow R$ be a production and $m_L : L \rightarrow G$ be a match. Clearly, this induces a match from L to the graphs underlying the compositions $\mathbb{G} \circ \mathbb{H}_1$ and $\mathbb{G} \otimes \mathbb{H}_2$ for any pair $\mathbb{H}_1$, $\mathbb{H}_2$.

Moreover, if the match induces a derivation $p/m : \mathbb{G} \Longrightarrow \mathbb{H}$, the derivation can be lifted to $\mathbb{G} \circ \mathbb{H}_1$ and $\mathbb{G} \otimes \mathbb{H}_2$. This is because a derivation must preserve the interfaces, so no composition can let a dangling condition occur: should a rule remove, for example, a node (such as the rule for the garbage collection of coercion on the right-hand side of Figure 12), then that node does not occur in the interface, hence it cannot interact with the context, and thus remains isolated after the composition. The shape of the target of the derivation can be proved, by checking the construction, to be $\mathbb{H} \circ \mathbb{H}_1$ and $\mathbb{H} \otimes \mathbb{H}_2$, respectively. □

The lemmas above are used to infer the following result.

Corollary B.1. Let P be a process and Γ be a set of names such that $fn(P) \subseteq \Gamma$. Then $nfp([\![P]\!]^p_\Gamma) = nfp([\![P]\!]^p) \otimes new_\Gamma$.

This corollary tells us that in order to prove soundness, it is enough to restrict attention to the set of names occurring free in a process.

Proposition B.1. Let P, Q be processes. If $P \equiv Q$, then $nfp([\![P]\!]^p) = nfp([\![Q]\!]^p)$.

Sketch of proof. We need to prove that the axioms in Figure 1 are preserved by the reduction in normal form of the tree encoding. First, note that α-conversion is intuitively verified since bound names are mapped into *internal* nodes (that is, nodes that are not in the image of the variable function) and these nodes are identity-less, up-to isomorphism, in graphs with interfaces.

The core of the proof then proceeds by structural induction on processes to show that whenever two processes are identified by an axiom, they can be rewritten to the same graph. For all the axioms, the pattern to be followed is the same: first, the graph expression associated with a process by the tree encoding is manipulated, thus highlighting the left-hand side of a rule; then, Lemma B.3 is applied.

As an example, consider the law $P \mid \mathtt{0} = P$. By construction, $[\![P \mid \mathtt{0}]\!]^p$ coincides with $par \circ ([\![P]\!]^p \otimes [\![\mathtt{0}]\!]^p_{fn(P)})$, and the latter part is encoded as $c \circ (\mathtt{0}_s \otimes new_{fn(P)})$. Now, the value of the latter expression is isomorphic to the value of $(c \circ \mathtt{0}_s) \otimes new_{fn(P)}$, so, thanks to Lemma B.2 and the associativity and commutativity of $\otimes$, we obtain the graph expression $par \circ ([\![P]\!]^p \otimes (c \circ \mathtt{0}_s))$. Now, $c \circ \mathtt{0}_s$ can be rewritten to $\mathtt{0}_p$ (the obvious graph with interfaces $(\{p\}, \varnothing)$) using the rule for removing coercion, hence, by Lemma B.3, we obtain the graph expression $par \circ ([\![P]\!]^p \otimes \mathtt{0}_p)$. Analogously, the removal of the *par* edge results in the expression $\Delta_p \circ ([\![P]\!]^p \otimes \mathtt{0}_p)$, whose value is isomorphic to $[\![P]\!]^p$.

The soundness of the remaining laws is proved similarly. □

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

The combination of the two results above implies that the soundness part of Theorem 6.1 holds. The completeness result is more difficult to prove, and we need to introduce two further technical lemmas. The first restates a well-known normal form for processes.

Lemma B.4 (Normal forms). Let P be a process and $\equiv$ be the congruence induced by the set of axioms in Figure 1. Then, P is equivalent, according to $\equiv$, to a process of the shape $(\nu x_1)\dots(\nu x_n)(M_1 \mid \dots \mid M_m)$ such that all x_j's are different names, $\{x_1,\dots,x_n\} \subseteq \bigcup_i \mathit{fn}(M_i)$ and each summation M_i has the shape $\delta_{i_1}.P_{i_1} + \dots + \delta_{i_k}.P_{i_k}$.

Now, thanks to the soundness of the encoding, it is enough to prove completeness for the normal forms, that is, that if two processes are mapped to graphs with interfaces with the same normal form (with respect to $\mathscr{R}^n_\pi$), then their normal forms as processes are also the same, modulo renaming of bound names and the associativity and commutativity axioms of the parallel and non-deterministic choice operators.

Lemma B.5. Let P be a process. If $\mathscr{R}^n_\pi$ entails a derivation $[\![P]\!]^p \Longrightarrow \mathbb{G}$, the graph with interfaces $\mathbb{G}$ satisfies the *single output property*: the underlying graph is a (hyper-)tree, only sharing $\circ$ leaves; the input (the node in the image of p) has no predecessor; and the outputs (the nodes in the image of $\mathit{fn}(P)$) have no successors.

Hence, the graph $\mathbb{G}$ is connected and acyclic, the input and target functions are jointly surjective and the output and source functions are jointly injective. Even if the normal forms for those graphs with interfaces in the image of the tree encoding could be described more precisely (additional constraints involve, for example, the node in the target of those edges labelled by restriction), this is enough for our purposes.

Sketch of proof. The property is clearly true for all the graph constants used in the tree encoding, and it is easy to see that it holds for the graph expressions resulting from the tree encoding: it is always preserved by the parallel composition operators, since the interfaces are discrete; and by case inspection it can be verified that it is preserved also by all the occurrences of the sequential composition operator. Hence the lemma holds since all the rules in $\mathscr{R}^n_\pi$ also preserve the property. □

Note now that the normal form of a process is essentially unique since the orderings of the restriction and parallel and sequential composition operators are irrelevant. Consequently, we will denote the process $(\nu x_1)\dots(\nu x_n)(M_1 \mid \dots \mid M_m)$ by the shorthand $(\nu\Gamma)\biguplus_{i=1}^m M_i$ for the set of names $\Gamma = \{x_1,\dots,x_n\}$.

We can now give a sketch of the completeness part. For more general completeness results on term-like presentations of graphical structures, see Bruni *et al.* (2002), Corradini and Gadducci (1999a) and Gadducci *et al.* (1999), in particular, see, for example, Corradini and Gadducci (1999a, Lemma 22).

Proposition B.2. Let P and Q be processes. If $\mathit{nfp}([\![P]\!]^p) = \mathit{nfp}([\![Q]\!]^p)$, then $P \equiv Q$.

Sketch of proof. Let P and Q be processes and $(\nu\Gamma_P)\biguplus_{i=1}^m M_i$ and $(\nu\Gamma_Q)\biguplus_{j=1}^o N_i$ be their respective normal forms. Consider the tree encodings $[\![P]\!]^p$ and $[\![Q]\!]^p$.

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

Note that the graph with interfaces v_w rewrites, according to the rule in Figure 11, to the value of the expression $\Delta_p \circ (id_p \otimes \overline{v}_w)$. By repeated applications, $[\![P]\!]^p$ rewrites to

$$[\Delta_p \circ (id_p \otimes \overline{v}_{x_1})] \circ \ldots \circ [\Delta_p \circ (id_p \otimes \overline{v}_{x_n})] \circ \mathbb{G} \circ (0_{x_n} \otimes id_{fn(P) \cup \{x_1,\ldots,x_{n-1}\}}) \circ \ldots \circ (0_{x_1} \otimes id_{fn(P)})$$

for $\mathbb{G}$ the value of $[\![\biguplus_{i=1}^m M_i]\!]_{fn(P) \cup \Gamma_P}$. First, note that the latter part of the expression has the same value of $0_{\Gamma_P} \otimes id_{fn(P)}$. So, let Δ_p^k be the unique graph with interfaces $(\{p\}, \{p_1, \ldots, p_k\})$: the former part of the expression then has the same value as $\Delta_p^{n+1} \circ (id_p \otimes \overline{v}_{\Gamma_P})$. A similar characterisation holds for the rewrites of *par* into Δ_p, so we finally obtain the graph expression

$$\Delta_p^{n+m} \circ \left(\overline{v}_{\Gamma_P} \otimes \bigotimes_{i=1}^{m} [\![M_i]\!]_{fn(P) \cup \Gamma_P} \right) \circ (0_{\Gamma_P} \otimes id_{fn(P)}) .$$

Now, since $nfp([\![P]\!]^p)$ and $nfp([\![Q]\!]^p)$ denote isomorphic graphs, they have the same interfaces, hence $fn(P) = fn(Q)$. Furthermore, there must be a bijective correspondence between the sets of edges attached to the image of the input p, so also Γ_P and Γ_Q have the same cardinality (hence, they are the same, since their identity is irrelevant) and the (normal forms of) P and Q have the same number of summations in parallel.

Hence, the result relies inductively on the proof that summations mapped to isomorphic graphs are equated by the structural congruence. The general result then holds by structural induction on the depth of the normal form of a process. □

Combined with Lemma B.3, the propositions above imply Theorem 6.1. Finally, note that Proposition 9.1, concerning processes expressions, is also a corollary, since process identifiers are just additional constants that may occur in a graph expression.

Appendix C. From reductions to rewrites

This appendix contains the proof of the first main result of the paper, Theorem 7.1, which relates process reductions to graph rewrites. By exploiting the correspondence between processes and tree encodings shown in the previous section, we could rely on the correspondence between graph rewrites and their algebraic presentation, as proved, for example, in Corradini and Gadducci (1999b, Theorem 4.9). We prefer instead to provide the sketch of a direct proof.

So, let Δ_s and $\Delta_{x,y}$ denote the constant graphs with interfaces $(\{s\}, \{s_1, s_2\})$ and $(\{x\}, \{x, y\})$, respectively, obviously corresponding to Δ_p; let new_s denote the constant graph with interfaces $(\varnothing, \{s\})$, corresponding to new_x; and let $[\![M]\!]_\Gamma^{gs}$ denote the graph expression $\Delta_s \circ (gs \otimes [\![M]\!]_\Gamma^s)$ for each summation M, enriching the tree encoding with a gs edge and obviously corresponding to $[\![P]\!]_\Gamma$.

Proposition C.1. Let R be the process $x(y).P + M \mid \overline{x}w.Q + N$. Then, $\mathscr{R}_\pi^a$ entails the derivation $nfp([\![R]\!]) \Longrightarrow nfp([\![P\{^w/_y\} \mid Q]\!]_{fn(R)}) \otimes [new_s \circ nfp([\![M + N]\!]_{fn(R)}^{gs})]$.

Sketch of proof. The proof has the following pattern: first, a graph expression corresponding to the left-hand side of the rule in $\mathscr{R}_\pi^a$ is chosen; then, a graph expression corresponding to the application of the rule to the given graph expression is computed;

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

finally, we show how the left-hand side occurs in the encoding $[\![R]\!]$, much as in the proof of Proposition B.1, and then apply Lemma B.3.

So, consider a graph expression corresponding to the left-hand side of the rule in $\mathscr{R}^a_\pi$ and such that the source of the *go* occurs in the input interface, and all the remaining nodes occur in the output interface, namely,

$$\mathbb{L} = \Delta_p \circ (go \otimes \Delta_p) \circ (\mathbb{G} \otimes \mathbb{H})$$

for

$$\mathbb{G} = c \circ \Delta_s \circ (id_s \otimes in_{x,y})$$

and

$$\mathbb{H} = c \circ \Delta_s \circ (id_s \otimes out_{x,w})\,.$$

The application of the rule with the identity match results in the value of

$$\mathbb{R} = [\Delta_p \circ (go \otimes \Delta_p)] \otimes [new_s \circ \Delta_s \circ (gs \otimes \Delta_s)] \otimes new_x \otimes (new_w \circ \Delta_{w,y})\,.$$

Consider the graphical encoding for R: it can be reduced by the *par* and *sum* rules in $\mathscr{R}^n_\pi$ into the value of the expression $\Delta_p \circ (go \otimes \Delta_p) \circ (\mathbb{G}' \otimes \mathbb{H}')$, for

$$\mathbb{G}' = c \circ \Delta_s \circ \{[\![M]\!]^s_{fn(R)} \otimes [in_{x,y} \circ ([\![P]\!]^p_{\{y\}\cup fn(R)} \otimes id_{\{x,y\}}) \circ (0_y \otimes id_{fn(R)})]\}$$

(and similarly for $\mathbb{H}'$). Manipulating $\mathbb{G}'$, we get the graph expression

$$\mathbb{G} \circ \{[\![M]\!]^s_{fn(R)} \otimes [([\![P]\!]^p_{\{y\}\cup fn(R)} \otimes id_{\{x,y\}}) \circ (0_y \otimes id_{fn(R)})]\}$$

(and similarly for $\mathbb{H}$ and $\mathbb{H}'$). The latter expression is equivalent to

$$([\![M]\!]^s_{fn(R)} \otimes [\![P]\!]^p_{\{y\}\cup fn(R)} \otimes id_{\{x,y\}}) \circ (0_y \otimes id_{fn(R)})\,.$$

We now turn our attention to the parallel composition $\mathbb{G}' \otimes \mathbb{H}'$. The resulting expression, after the manipulation of $\mathbb{G}'$ and $\mathbb{H}'$, is equivalent to the left-hand side $\mathbb{L}$ of the rule in $\mathscr{R}^a_\pi$, post-composed with

$$\mathbb{C} = ([\![M]\!]^s_{fn(R)} \otimes [\![P]\!]^p_{\{y\}\cup fn(R)} \otimes [\![N]\!]^s_{fn(R)} \otimes [\![Q]\!]^p_{fn(R)} \otimes id_{\{x,y,w\}}) \circ (0_y \otimes id_{fn(R)})\,.$$

After the reduction step, the resulting graph is the value of the expression $\mathbb{R} \circ \mathbb{C}$, thanks to Lemma B.3. This can be further manipulated into the equivalent expression

$$\begin{aligned}&\{[\Delta_p \circ (go \otimes \Delta_p)] \otimes [new_s \circ \Delta_s \circ (gs \otimes \Delta_s)]\}\\ &\quad \circ [[\![M]\!]^s_{fn(R)} \otimes [\![P]\!]^p_{\{y\}\cup fn(R)} \otimes [\![N]\!]^s_{fn(R)} \otimes [\![Q]\!]^p_{fn(R)} \otimes (new_w \circ \Delta_{y,w})]\\ &\quad \circ (0_y \otimes id_{fn(R)})\,;\end{aligned}$$

the intermediate expression is equivalent to

$$([\![M]\!]^s_{fn(R)} \otimes id_p \otimes [\![N]\!]^s_{fn(R)} \otimes [\![Q]\!]^p_{fn(R)}) \circ [id_{fn(R)} \otimes [\![P]\!]^p_{\{y\}\cup fn(R)} \otimes (new_w \circ \Delta_{y,w})]\,.$$

Finally, note that the expression

$$[id_{fn(R)} \otimes [\![P]\!]^p_{\{y\}\cup fn(R)} \otimes (new_w \circ \Delta_{y,w})] \circ (0_y \otimes id_{fn(R)})$$

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

has the same value as

$$id_{fn(R)} \otimes [\![P\{^w/_y\}]\!]^p_{fn(R)}.$$

Thus, the expression resulting from the reduction step has the same value as

$$[\Delta_p \circ (go \otimes \Delta_p) \circ ([\![P\{^w/_y\}]\!]^p_{fn(R)} \otimes [\![Q]\!]^p_{fn(R)})] \otimes [new_s \circ \Delta_s \circ (gs \otimes \Delta_s) \circ ([\![M]\!]^s_{fn(R)} \otimes [\![N]\!]^s_{fn(R)})],$$

and the result holds. □

A further lemma characterises the normal form for discarded summations.

Lemma C.1. Let M be a summation and Γ be a set of names such that $fn(M) \subseteq \Gamma$. Then $nfg(nfp([\![M]\!]^{gs}_{\Gamma}))$ has the same value as the expression $gs \otimes new_{\Gamma}$.

These results are enough to prove Theorem 7.1. In fact, the closure under structural congruence $\equiv$ is ensured by the correspondence between processes and graphs with interfaces established via the GTS $\mathscr{R}^n_\pi$; while the closure of reduction with respect to process contexts is ensured by Lemma B.3 (see also Corradini and Gadducci (1999b, Lemma 4.8)).

Appendix D. From rewrites to reductions

In this appendix we turn our attention to the proof of Theorem 7.2, which relates graph rewrites to process reductions. As with the proof of the completeness of $\mathscr{R}^n_\pi$, it exploits the normal form for processes established by Lemma B.4.

We first state a simple result concerning the application of the rule p_π in $\mathscr{R}^a_\pi$.

Lemma D.1. Let $\mathbb{G}$ be a graph with discrete interfaces and m_L be an edge-preserving match for the rule p_π in $\mathscr{R}^a_\pi$ into the graph underlying $\mathbb{G}$. Then, a direct derivation $p_\pi/m : \mathbb{G} \Longrightarrow \mathbb{H}$ for that given match always exists.

The existence of the derivation for any match m_L relies on standard results from graph rewriting theory, since the rule is left-injective, and it is a bijection on nodes, while the match is injective on edges (for example, the existence of the so-called *pushout complement* is ensured by the fact that no node is removed by the application of the rule: see, for example, Corradini *et al.* (1997, Proposition 3.3.4)).

Proposition D.1. Let R be a process. If $\mathscr{R}^a_\pi$ entails the derivation $nfp([\![R]\!]) \Longrightarrow \mathbb{G}$, there exist processes P, Q, S and summations M, N such that

$$R \equiv (\nu\Gamma_R)(x(y).P + M \mid \overline{x}w.Q + N \mid S)$$

for a set of names Γ_R of cardinality m and

$$\begin{aligned}\mathbb{G} \;=\; & \Delta_p^{m+1} \\ & \circ \{\overline{\nu}_{\Gamma_R} \otimes nfp([\![P\{^w/_y\} \mid Q \mid S]\!]_{fn(R)\cup\Gamma_R}) \otimes [mew_s \circ nfp([\![M+N]\!]^{gs}_{fn(R)\cup\Gamma_R})]\} \\ & \circ (0_{\Gamma_R} \otimes id_{fn(R)}).\end{aligned}$$

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

Sketch of proof. For any process R, the graph $\mathit{nfp}(\llbracket R \rrbracket)$ has interfaces $(\{p\}, \mathit{fn}(R))$ and exactly one occurrence of a go edge, which is outgoing from the image of p.

Since $\mathit{nfp}(\llbracket R \rrbracket)$ satisfies the single output property (see Lemma B.5), any match m_L for the rule p_π must be injective, and at most coalesces the $\circ$ nodes corresponding to the names 1 and 3 in Figure 15[†]. Furthermore, the presence of a go edge forces m_L to map the edges of the left-hand side of p_π onto two corresponding top operators in $\mathit{nfp}(\llbracket R \rrbracket)$.

So, let $(\nu\Gamma_R)\biguplus_{i=1}^{n} M_i$ be the normal form for R. The mapping m_L identifies two indexes $j,k \in 1 \ldots n$, such that $M_j = x(y).P_j + N_j$ and $M_k = \overline{x}w.P_k + N_k$ (for x the unique name such that it coincides with $m_L(1)$, and similarly for y and w). Since we may safely assume $j = 1$ and $k = 2$, we end up obtaining a labelled reduction

$$(\nu\Gamma_R)(x(y).P + N \mid \overline{x}w.Q + M \mid S) \to (\nu\Gamma_R)(P\{^{w}/_{y}\} \mid Q \mid S)\,.$$

Then, the result holds by mimicking the reasoning of Proposition C.1 above, and because, according to Lemma B.3, graph rewrites are preserved by closure with respect to graph contexts. □

Theorem 7.2 is then a corollary of Proposition D.1. The proof takes into account the characterisation of the normal form for discarded sums, as shown in Lemma C.1, and the further removal of useless restriction operators occurring among $\overline{\nu}_R$.

Proposition 9.2, which concerns the reduction of recursive processes, is proved along the lines of the two propositions in these appendixes. In fact, the key point is that strengthened versions of the various lemmas still hold, since the unfolding rules remove no nodes.

Acknowledgments

I am greatly indebted to the anonymous referees for suggesting many improvements with respect to the submitted version.

References

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

Baldan, P., Gadducci, F. and Montanari, U. (2006) Concurrent rewriting for graphs with equivalences. In: Baier, C. and Hermanns, H. (eds.) Concurrency Theory. *Springer-Verlag Lecture Notes in Computer Science* **4137** 279–294.

Berry, G. and Boudol, G. (1992) The chemical abstract machine. *Theoretical Computer Science* **96** 217–248.

Bonchi, F., Gadducci, F. and König, B. (2006) Process bisimulation via a graphical encoding. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L. and Rozenberg, G. (eds.) Graph Transformation. *Springer-Verlag Lecture Notes in Computer Science* **4187** 168–183.

[†] In fact, no coalescing of the $\circ$ node labelled 2 occurs in the tree encoding, since it corresponds to a local name, and the property is preserved under reduction.

https://doi.org/10.1017/S096012950700610X Published online by Cambridge University Press

Bruni, R., Gadducci, F. and Montanari, U. (2002) Normal forms for algebras of connections. *Theoretical Computer Science* **286** 247–292.

Caires, L. (2004) Behavioral and spatial observations in a logic for the π-calculus. In: Walukiewicz, I. (ed.) Foundations of Software Science and Computation Structures. *Springer-Verlag Lecture Notes in Computer Science* **2987** 72–87.

Caires, L. and Cardelli, L. (2003). A spatial logic for concurrency (part I). *Information and Computation* **186** 194–235.

Căzănescu, V.-E. and Ştefănescu, Gh. (1992) A general result on abstract flowchart schemes with applications to the study of accessibility, reduction and minimization. *Theoretical Computer Science* **99** 1–63.

Corradini, A. and Gadducci, F. (1999a) An algebraic presentation of term graphs, via gs-monoidal categories. *Applied Categorical Structures* **7** 299–331.

Corradini, A. and Gadducci, F. (1999b) Rewriting on cyclic structures: Equivalence between the operational and the categorical description. *Informatique Théorique et Applications/Theoretical Informatics and Applications* **33** 467–493.

Corradini, A., Montanari, U. and Rossi, F. (1994) An abstract machine for concurrent modular systems: CHARM. *Theoretical Computer Science* **122** 165–200.

Corradini, A., Montanari, U. and Rossi, F. (1996) Graph processes. *Fundamenta Informaticae* **26** 241–265.

Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R. and Löwe, M. (1997) Algebraic approaches to graph transformation I: Basic concepts and double pushout approach. In: Rozenberg, G. (ed.) *Handbook of Graph Grammars and Computing by Graph Transformation*, Volume 1, World Scientific 163–245.

Courcelle, B. (1997) The expression of graph properties and graph transformations in monadic second-order logic. In: Rozenberg, G. (ed.) *Handbook of Graph Grammars and Computing by Graph Transformation*, Volume 1, World Scientific 313–400.

Drewes, F., Habel, A. and Kreowski, H.-J. (1997) Hyperedge replacement graph grammars. In: Rozenberg, G. (ed.) *Handbook of Graph Grammars and Computing by Graph Transformation*, Volume 1, World Scientific 95–162.

Ehrig, H. and König, B. (2004) Deriving bisimulation congruences in the DPO approach to graph rewriting. In: Walukiewicz, I. (ed.) Foundations of Software Science and Computation Structures. *Springer-Verlag Lecture Notes in Computer Science* **2987** 151–166.

Fu, Y. (1999) Variations on mobile processes. *Theoretical Computer Science* **221** 327–368.

Gadducci, F. (2003) Term graph rewriting and the π-calculus. In: Ohori, A. (ed.) Programming Languages and Semantics. *Springer-Verlag Lecture Notes in Computer Science* **2895** 37–54.

Gadducci, F., Heckel, R. and Llabrés, M. (1999) A bi-categorical axiomatisation of concurrent graph rewriting. In: Hofmann, M., Pavlovič, D. and Rosolini, G. (eds.) Category Theory and Computer Science. *Electronic Notes in Theoretical Computer Science* **29**.

Gadducci, F. and Lluch Lafuente, A. (2006) Graphical verification of a spatial logic for the π-calculus. In: Heckel, R., König, B. and Rensink, A. (eds.) Graph Transformation for Verification and Concurrency. *Electronic Notes in Theoretical Computer Science* **154**.

Gadducci, F. and Lluch Lafuente, A. (2007) Graphical encoding of a spatial logic for the π-calculus. In: Montanari, U. and Mossakowski, T. (eds.) Algebra and Coalgebra in Computer Science. *Springer-Verlag Lecture Notes in Computer Science* (to appear).

Gadducci, F. and Montanari, U. (2001) A concurrent graph semantics for mobile ambients. In: Brookes, S. and Mislove, M. (eds.) Mathematical Foundations of Programming Semantics. *Electronic Notes in Theoretical Computer Science* **45**.

Gadducci, F. and Montanari, U. (2002) Comparing logics for rewriting: Rewriting logic, action calculi and tile logic. *Theoretical Computer Science* **285** 319–358.

Gadducci, F. and Montanari, U. (2005) Observing reductions in nominal calculi *via* a graphical encoding of processes. In: Middeldorp, A., van Oostrom, V., van Raamsdonk, F. and de Vrijer, R. C. (eds.) Processes, terms and cycles (Klop Festschrift). *Springer-Verlag Lecture Notes in Computer Science* **3838** 106–126.

Goguen, J. and Meseguer, J. (1992) Order sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science* **105** 217–273.

König, B. (1999) Generating type systems for process graphs. In: Baeten, J. C. M. and Mauw, S. (eds.) Concurrency Theory. *Springer-Verlag Lecture Notes in Computer Science* **1664** 352–367.

Laneve, C., Parrow, J. and Victor, B. (2001) Solo diagrams. In: Kobayashi, N. and Pierce, B. (eds.) Theoretical Aspects of Computer Science. *Springer-Verlag Lecture Notes in Computer Science* **2215** 127–144.

Leifer, J. and Milner, R. (2000) Deriving bisimulation congruences for reactive systems. In: Palamidessi, C. (ed.) Concurrency Theory. *Springer-Verlag Lecture Notes in Computer Science* **1877** 243–258.

Milner, R. (1989) *Communication and Concurrency*, Prentice Hall.

Milner, R. (1993) The polyadic π-calculus: A tutorial. In: Bauer, F. L., Brauer, W. and Schwichtenberg, H. (eds.) Logic and Algebra of Specification. *Nato ASI Series F* **94** 203–246.

Milner, R. (2001) Bigraphical reactive systems. In: Larsen, K. G. and Nielsen, M. (eds.) Concurrency Theory. *Springer-Verlag Lecture Notes in Computer Science* **2154** 16–35.

Montanari, U., Pistore, M. and Rossi, F. (1999) Modeling concurrent, mobile and coordinated systems via graph transformations: Concurrency, parallellism, and distribution. In: Ehrig, H., Kreowski, H.-J., Montanari, U. and Rozenberg, G. (eds.) *Handbook of Graph Grammars and Computing by Graph Transformation*, Volume 3, World Scientific 189–268.

Plotkin, G. (1981) A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University.

Plump, D. (2005) Confluence of graph transformation revisited. In: Middeldorp, A., van Oostrom, V., van Raamsdonk, F. and de Vrijer, R. C. (eds.) Processes, terms and cycles (Klop Festschrift). *Springer-Verlag Lecture Notes in Computer Science* **3838** 280–308.

Sangiorgi, S. and Walker, D. (2001) *The π-calculus: A Theory of Mobile Processes*, Cambridge University Press.

Yoshida, N. (1994) Graph notation for concurrent combinators. In: Ito, T. and Yonezawa, A. (eds.) Theory and Practice of Parallel Programming. *Springer-Verlag Lecture Notes in Computer Science* **907** 393–412.