A calculus of branching processes

Thomas Ehrhard, Jean Krivine, Ying Jiang

A Calculus of Branching Processes

Thomas Ehrhard[a], Jean Krivine[a], Ying Jiang[b]

[a] *CNRS, Paris Diderot Univ. IRIF, 8, place Aurélie Nemours 75013 Paris, France*
[b] *Institute of Software, Chinese Academy of Sciences, 4 South Fourth Street, Zhong Guan Cun, Beijing 100190, P.R. China*

Abstract

CCS-like calculi can be viewed as an extension of classical automata with communication primitives. We are interested here to follow this principle, applied to tree-automata. It naturally yields a calculus of branching processes (CBP), where the continuations of communications are allowed to branch according to the arity of the communication channel. After introducing the calculus with a reduction semantics we show that CBP can be "implemented" by a fully compositional LTS semantics. We argue that CBP offers an interesting tradeoff between calculi with a fixed communication topology à la CCS and calculi with dynamic connectivity such as the π-calculus.

1. Introduction

The Calculus of Communicating Systems (CCS) [15] and Communicating Sequential Processes (CSP) [14] are expressive formalisms that are widely used to model concurrent systems, in which processes synchronize by handshake. One can view concurrent runs in these calculi as computations of classical automata or Petri Nets in which transitions have to synchronize on complementary symbols. The starting point of the formalism we introduce in this paper is the following: what happens if instead of synchronizing classical transition systems, we were to allow more expressive automata to synchronize?

We introduce a Calculus of Branching Processes (CBP), that is obtained by adding synchronization to tree-automata [9]. We obtain a process calculus whose continuation scheme is branching according to the arity of the communication symbols. A CBP process of the form $f(p, q)$ has communication symbol f and continuations p and q. Branching communication structure can only be achieved if the computations of p and q are not allowed to interact anymore. This leads naturally to a definition of a communication graph that connects processes and specifies which symbols are able to interact. In classical process algebra, the communication graph is left implicit since any two complementary symbols may synchronize. This explains, for instance, the difficulty to make concrete implementations of the π-calculus [16]: in the absence of a network topology, one has to conceive of astute mechanisms to implement communications [18]. This has led to the definition of process calculi in which routing of communication is either trivially directed

Email addresses: ehrhard@irif.fr (Thomas Ehrhard), jkrivine@irif.fr (Jean Krivine), jy@ios.ac.cn (Ying Jiang)

[11, 12] or explicit [13, 17, 10]. An interesting aspect of CBP is that it comes with a structural operational semantics that makes explicit how the communication graph evolves over time.

Outline. In Section 2 we introduce the specification of CBP's operational semantics by means of (communication) graph rewriting rules. The syntax of processes involves two main constructs: parallel composition, as in CCS, and prefixing. As already suggested, this latter construct differs from CCS prefixing in that there can be more than one continuation. The number of continuations is determined by the label of the prefix which is a function symbol. For the sake of simplicity, we assume in the technical developments that this arity is always 2 (more general arities can easily be encoded in this setting).

The operational semantics of this calculus is mainly defined as a rewriting system. Our main design choice is summarized in Figure 1: when two processes $p = f(p_1, p_2)$ and $q = \bar{f}(q_1, q_2)$ synchronize, the resulting configuration stipulates that p_i can only communicate with q_i (for $i = 1, 2$) and with the environment of p (and symmetrically for q). We can motivate this choice as follows:

- p_1 can synchronize only with q_1, and not with q_2 (and similarly for p_2), to have CBP behave as a conservative extension of tree automata, as explained in Section 3.1.
- p_1 cannot synchronize with p_2 in order to obtain branching continuations in the style of process threads.
- Last, the fact that the continuations of a prefixed process p inherit the synchronization capabilities of p is inspired by the operational semantics of CCS and has the major effect of allowing the continuations of p to communicate by means of a common, shared, context (the context X in Fig. 1).

These restrictions on possible synchronizations are implemented by a graphical structure: a *process graph* Γ is an undirected simple graph, the vertices of which are labeled by processes. The edges of the process graph correspond to communication capabilities: two processes can synchronize if they are located on opposite ends of an edge of this graph. A process graph rewriting event corresponds to a synchronization that updates the process graph according to the three principles listed above and summarized in Figure 1. The corresponding rewriting relation on process graphs is presented in Section 2.4.

Expressivity of CBP is discussed in Section 3. First we show, in Section 3.1, that CBP is a conservative extension of tree automata, in the sense that the recognition of a tree a by an automaton T can be expressed by the fact that a process representing the "parallel composition" of T and (the dual of) a reduces to an *idle* process. We then give an example, in Section 3.2, that illustrates the branching behavior of process continuations that are still able to interact by means of a shared environment. This is obtained by modeling a reader and a writer thread that have access to a common register process. Finally we explore, in Section 3.3, the *graph self-assembly* capabilities of CBP and we illustrate specifically that hexagones naturally arise from CBP rewriting, allowing for instance to implement a ternary version of dining philosophers. We consider this example as a local interaction based implementation of a distributed consensus, since the main feature of this sequence of reductions, presented in Figure 3, is that no

interaction between the philosophers is possible before the final hexagonal configuration is reached.

The compositional semantics of CBP is introduced in Section 4. As usual, rewriting semantics is not compositional (the semantics of a process cannot be described in terms of the semantics of its subprocesses). Following a standard approach initiated by Milner with CCS, we present a *labeled transition system* for CBP. Unlike CCS where labels are simple letters or τ-symbols (for internal synchronizations), our labels involve also *addresses* in *process trees*, which are the syntactic objects our LTS deals with. These trees can be understood as process graphs enriched with further informations on the sequential construction steps of the underlying graph structure.

Thanks to the presence of addresses in our process tree LTS, we are able to prove in Section 5 that the two operational semantics presented in Section 2 and 4 are indeed equivalent through a translation from process trees to process graphs.

2. The calculus CBP and its reduction semantics

In this section we introduce the calculus CBP and its reduction semantics, which is a conservative extension of CCS [15]. This semantics describes the evolution of processes that are distributed on the vertices of a graph whose edges enable message exchange: two processes that are connected may interact if they communicate on complementary function symbols. Importantly, processes that are not connected by an edge cannot interact[1].

CBP has essentially one reduction rule, which is informally described in Figure 1. The remainder of the present section is dedicated to the formal description of the reduction semantics, which should be understood as the *specification* of a distributed computation. Looking at Figure 1 the reader may wonder whether the new wiring of the right hand side of the reduction may be obtained by means of local rules since there are, *a priori*, many edges that may appear after a reduction (in particular for the full CBP calculus whose reduction scheme is shown on the right of Fig. 1). A solution to this problem is presented in Section 4, where a fully compositional semantics for CBP is introduced.

2.1. Syntax

For simplicity we restrict ourselves to binary CBP, where all function symbols have arity 2. Full CBP is obtained by relaxing this condition.

A *process* is a term built over the following grammar:

$$\begin{array}{lcll} s & ::= & f(p,p) \mid \mathbf{0} \mid D(\tilde{f}) \mid s+s & \text{(sums)} \\ p & ::= & s \mid (p \parallel p) \mid p \setminus \mathcal{H} & \text{(processes)} \end{array}$$

with $f \in \mathcal{F}$ a function symbol, $\mathcal{H} \subseteq \mathcal{F}$ and $D(\tilde{f})$ a (guarded) recursive definition with parameters $\tilde{f}$. We assume a definition set of the form $\mathsf{Def} =_{\mathsf{def}} \bigcup_i \{D_i(\tilde{f}_i) := s_i\}$, where $\tilde{f}_i$ are function symbols occurring free in s_i (see fn function defined in Section 2.3).

[1] Classical CCS terms may be denoted by processes distributed on the nodes of a clique, in this context.

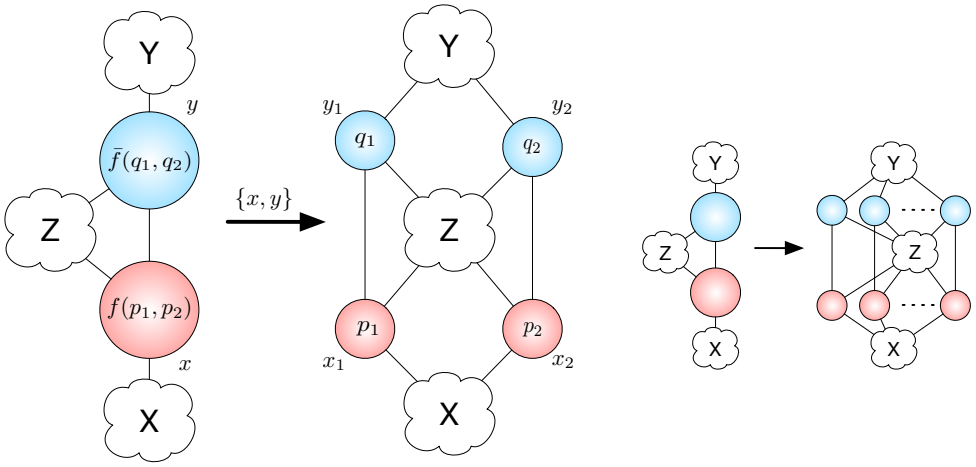


Figure 1: Binary CBP's reduction rule (left): the nodes x and y are ready to synchronize on symbols f and $\bar{f}$ (of arity 2), and are connected by an edge. X (resp. Y) denotes nodes in the context that are connected to x but not y (resp. y but not x). Z denotes nodes of the context of the reduction that are connected to both x and y. The reduction replaces nodes x and y by their continuations x_1, x_2 and y_1, y_2. New edges connect nodes in X (resp. Y) and Z to the continuations of x (resp. y). On the right, the reduction scheme for n-ary functions in the full CBP.

We let $\mathcal{P} = \{p, q, \dots\}$ denote the set of process terms. We also consider the involution $\bar{\cdot} : \mathcal{F} \to \mathcal{F}$ that will serve to denote dual function symbols[2].

Let $\mathcal{V} =_{\mathsf{def}} \{x, y, z, \dots\}$ be a set of vertices. A *process graph* $\Gamma = (V, E, \pi : V \to \mathcal{P})$ is a tuple where $V \subseteq \mathcal{V}$ are the vertices of Γ, $E \subseteq \{\{x, y\} \mid x, y \in V\}$ are its edges, and $\pi : V \to \mathcal{P}$ is a labeling function that maps each vertex to a process. We use $\mathcal{G} =_{\mathsf{def}} \{\Gamma, \Delta, \dots\}$ to denote the set of process graphs.

2.2. Update

Computations in CBP progress in the style of classical process algebra: complementary communication symbols are consumed conjunctly and their continuation is spawned. In the context of binary CBP, each function symbol comes with two continuation processes. Unlike the CCS term $a.(p \parallel q)$ that has a unique continuation $p \parallel q$, the CBP term $a(p, q)$ has two continuations that may not exchange messages in the future.[3] This is captured by a special form of substitution (described below) that substitutes a node by two continuation nodes, while taking care of the new edges that should appear.

We introduce the update functions that will be used to bring process graphs into a reducible form. These functions are defined respectively on V, E, π and eventually on Γ.

[2] Unlike CCS, self-dual function symbols, $\bar{f} = f$, are also allowed.

[3] However, the CCS process $a.(p \parallel q)$ can be mimicked by the CBP process $a(p \parallel q, 0)$, where p and q are in the same continuation and thus are able to interact.

For all V, E, π, with $x \in V$, $x_1, x_2 \notin V$ and $p, q \in \mathcal{P}$:

$$\begin{array}{lll} V[x_1, x_2/x] & =_{\mathsf{def}} & (V \setminus x) \uplus \{x_1, x_2\} \\ \\ (E \uplus \{\{z,t\}\})[x_1, x_2/x] & =_{\mathsf{def}} & \{\{x_1,t\},\{x_2,t\}\} \uplus (E[x_1,x_2/x]) \text{ if } x = z \\ (E \uplus \{\{z,t\}\})[x_1, x_2/x] & =_{\mathsf{def}} & \{\{z,t\}\} \uplus (E[x_1,x_2/x]) \text{ if } x \notin \{z,t\} \\ \emptyset[x_1, x_2/x] & =_{\mathsf{def}} & \emptyset \\ \\ \pi[x_1 : p, x_2 : q/x] & =_{\mathsf{def}} & [y \mapsto \pi(y)]_{y \in V \setminus \{x\}} + [x_1 \mapsto p; x_2 \mapsto q] \end{array}$$

where $\pi' = \pi + [y \mapsto p]$ is a labeling map with $\mathsf{Dom}(\pi') = \mathsf{Dom}(\pi) \uplus \{y\}$ and satisfying $\pi'(x) = \pi(x)$ for all $x \in \mathsf{Dom}(\pi)$ and $\pi'(y) = p$.

Definition 2.1 (Process graph update)**.** Let $\Gamma =_{\mathsf{def}} (V, E, \pi)$ and $\Gamma' =_{\mathsf{def}} (V', E', \pi')$. We define $\Gamma\{(x_1 : p, x_2 : q)/x \mid (y_1 : p', y_2 : q')/y\} = \Gamma'$ if:

$$\begin{array}{lcl} V' & = & V[x_1, x_2/x][y_1, y_2/y] \\ E' & = & (E \setminus \{\{x,y\}\})[x_1, x_2/x][y_1, y_2/y] \cup \{\{x_1, y_1\}, \{x_2, y_2\}\} \\ \pi' & = & \pi[x_1 : p, x_2 : q/x][y_1 : p', y_2 : q'/y] \end{array}$$

Let x be a vertex and p be a process, we use $\{x : p\}$ for the process graph (V, E, π) where $V = \{x\}$, $E = \emptyset$ and $\pi(x) = p$. We say that a process graph (V, E, π) is *idle* if $\pi(x) = \mathbf{0}$ for all $x \in V$.

2.3. Normalization

Process graph reduction can be blocked by processes that are not in a reducible form. This is due to three factors:

- Processes of the form $p \parallel q$ need to be distributed as two processes connected by an edge.
- Restriction operators need to act as a fresh function symbol generators at the level of process graphs.
- Constants need to be replaced by their definitions.

Bringing process graphs to a reducible form is the job of the normalization procedure that we introduce now. For all $p \in \mathcal{P}$, for all function symbols $f, g \in \mathcal{F}$ such that g does not occur in p, we write $p\{g/f\}$ for the substitution of symbol f by g in p.

We define the function $\mathsf{fn} : \mathcal{P} \to \mathcal{V}$, which yields the *free names* of processes, as:

$$\begin{array}{lll} \mathsf{fn}(f(p,q)) & =_{\mathsf{def}} & \{f\} \cup \mathsf{fn}(p) \cup \mathsf{fn}(q) \\ \mathsf{fn}(p+q) & =_{\mathsf{def}} & \mathsf{fn}(p) \cup \mathsf{fn}(q) \\ \mathsf{fn}(p \parallel q) & =_{\mathsf{def}} & \mathsf{fn}(p) \cup \mathsf{fn}(q) \\ \mathsf{fn}(D(\tilde{f})) & =_{\mathsf{def}} & \{f \in \tilde{f}\} \\ \mathsf{fn}(\mathbf{0}) & =_{\mathsf{def}} & \emptyset \\ \mathsf{fn}(p \setminus \mathcal{H}) & =_{\mathsf{def}} & \mathsf{fn}(p) \setminus \mathcal{H} \end{array}$$

and, for all process graph (V, E, π), we define $\mathsf{fn}(\pi) =_{\mathsf{def}} \bigcup_{x \in V} \mathsf{fn}(\pi(x))$. Process *normalization* $\Rrightarrow_{\mathsf{p}} \subseteq \mathcal{P} \times \mathcal{P}$ is:

$$\begin{array}{lll} p \parallel (q \setminus \mathcal{H}) & \Rrightarrow_{\mathsf{p}} & (p \parallel q) \setminus \mathcal{H} \text{ if } \mathcal{H} \cap \mathsf{fn}(p) = \emptyset \\ (p \setminus \mathcal{H}) \parallel q & \Rrightarrow_{\mathsf{p}} & (p \parallel q) \setminus \mathcal{H} \text{ if } \mathcal{H} \cap \mathsf{fn}(q) = \emptyset \\ p \setminus \mathcal{H} \setminus \mathcal{H}' & \Rrightarrow_{\mathsf{p}} & p \setminus (\mathcal{H} \cup \mathcal{H}') \\ D(\tilde{f}) & \Rrightarrow_{\mathsf{p}} & s\{\tilde{f}/\tilde{g}\} \text{ if } (D(\tilde{g}), s) \in \mathsf{Def} \end{array}$$

and is applied up-to α renaming of bound names. Normalization propagates at the level of process graphs. For all $x \in V$:

$$(V, E, \pi + [x \mapsto p]) \Rrightarrow_{\mathsf{g}} (V, E, \pi + [x \mapsto q]) \text{ if } p \Rrightarrow_{\mathsf{p}} q$$

$$\begin{array}{lll} (V, E, \pi + [x \mapsto p \setminus \{g\} \uplus \mathcal{H}]) & \Rrightarrow_{\mathsf{g}} & (V, E, \pi + [x \mapsto (p\{f/g\}) \setminus \mathcal{H}]) \\ & & \text{if } f \notin \mathsf{fn}(\pi) \cup \mathsf{fn}(p) \cup \mathcal{H} \end{array}$$

$$(V, E, \pi + [x \mapsto p \parallel q]) \Rrightarrow_{\mathsf{g}} (V[x_1, x_2/x], E[x_1, x_2/x], \pi[x_1 : p, x_2 : q/x])$$

where $x_i, y_i \notin V$.

Definition 2.2 (Normalized process graph)**.** A process graph (V, E, π) is *normalized* if all its nodes are either idle or labeled by a function symbol, i.e. for all $x \in V$, $\pi(x) = \mathbf{0}$ or $\pi(x) = f(p, q)$ for some $f \in \mathcal{F}$ and $p, q \in \mathcal{P}$.

2.4. Process graph rewriting

We are now in position to formally describe the reduction given in Fig. 1. Summation contexts are terms with exactly one hole of the form:

$$S[\bullet] =_{\mathsf{def}} s + \bullet \mid \bullet + s \mid \bullet$$

The process graph reduction semantics is the least relation satisfying derivations (1) and (2), for all $\Gamma =_{\mathsf{def}} (V, E, \pi)$.

$$\frac{\{x, y\} \in E \quad \{x_0, x_1, y_0, y_1\} \cap V = \emptyset \quad \pi(x) = S[f(p_0, p_1)] \quad \pi(y) = S'[\bar{f}(q_0, q_1)]}{\Gamma \to_{\{x,y\}} \Gamma\{(x_0 : p_0, x_1 : p_1)/x \mid (y_1 : q_0, y_1 : q_1)/y\}} \quad (1)$$

Reduction of process graphs is performed up-to normalization steps:

$$\frac{\Gamma \Rrightarrow_{\mathsf{g}} \Delta \to_{\{x,y\}} \Delta' \Rrightarrow_{\mathsf{g}} \Gamma'}{\Gamma \to_{\{x,y\}} \Gamma'} \quad (2)$$

We write also $\Gamma \to \Gamma'$ when $\Gamma \to_{\{x,y\}} \Gamma'$ for some x, y. We use $\to^*$ for the reflexive-transitive closure of this relation.

3. Expressivity

In this section we explore the expressivity of CBP. Although the formal treatment is restricted to binary CBP, in writing examples we will feel free to use the full calculus CBP. These examples can easily be encoded using the strict binary CBP, but this would be to the cost of readability.

3.1. Encoding tree automata

One major motivation of this work was to extend tree automata (considered from a top-down point of view) to an interactive setting, just as CCS can be understood as a natural interactive extension of ordinary word automata in the following sense. Given an automaton A, it is possible to define a process P_A such that, for any word $w = a_1 \dots a_k$, the automaton A recognizes w iff the CCS process $w \parallel P_A$ reduces to $\mathbf{0}$ (where we trivially see w as the process $a_1.\cdots.a_k.\mathbf{0}$), using the same method as in the definition of CCS testing semantics in [7] (where one would probably have rather written $\bar{w} \parallel P_A$ where $\bar{w} = \bar{a_1}.\cdots.\bar{a_k}.\mathbf{0}$, but this is a pure matter of conventional choice in the definition of P_A). The object of this section is to prove a completely similar statement for tree automata and CBP.

A tree automaton is a tuple $T = (Q, \Theta, \circ)$ where Q is a finite set of *states*, $\circ \in Q$ is a *final state* and Θ is a finite set of transitions. An element of Θ is a triple $(q, f, (q_1, q_2))$ where $f \in \mathcal{F}$ and $q, q_1, q_2 \in Q$. For the sake of simplicity, and without loss of expressiveness, we assume that no element of Θ has the form $(\circ, f, (q_1, q_2))$ (no transitions from the final state).

Let $*$ be a special symbol representing the empty tree. Trees are defined as follows:

$$a ::= * \mid f(a, a)$$

where $f \in \mathcal{F}$. We use $\mathbb{T}$ to denote the set of trees.

Given an automaton $T = (Q, \Theta, \circ)$ and an element q of Q, we define a set of trees $\mathcal{L}(T, q) = L_q$ where $(L_q)_{q \in Q}$ is the smallest element of $(2^{\mathbb{T}})^Q$ such that

- $* \in L_\circ$
- if $a_i \in L_{q_i}$ for $i = 1, 2$ and $(q, f, (q_1, q_2)) \in \Theta$, then $f(a_1, a_2) \in L_q$.

The set $\mathcal{L}(T, q)$ is the *tree language recognized by T at state q*.

Given a tree a, we can define a process $\widehat{a}$ in the following straightforward way: $\widehat{*} = \mathbf{0}$ and $\widehat{f(a_1, a_2)} = f(\widehat{a_1}, \widehat{a_2})$.

Now we explain how to associate a process with a tree automaton, so assume $T = (Q, \Theta, \circ)$ to be a given tree automaton. Let $\tilde{f}$ be a repetition-free list of all names which occur in Θ. With each $q \in Q$ we associate injectively a process constant D_q. We define a set Def_T by

$$\mathsf{Def}_T = \{D_q(\tilde{f}) := \sum_{(q,f,(q_1,q_2)) \in \Theta} \bar{f}(D_{q_1}(\tilde{f}), D_{q_2}(\tilde{f})) \mid q \in Q \setminus \{\circ\}\} \cup \{(D_\circ(\tilde{f}) := \mathbf{0}\}\,.$$

Theorem 3.1. *Let $q \in Q$ and let a be a tree. Then $a \in \mathcal{L}(T, q)$ iff $\{x : (\widehat{a} \parallel D_q(\tilde{f}))\} \to^* \Gamma_0$ where Γ_0 is an idle process graph.*

Proof. By induction on the tree a. Observe that

$$\{x : (\widehat{a} \parallel D_q(\tilde{f}))\} \Rrightarrow_{\mathsf{g}} \Gamma = (\{y, z\}, \{\{y, z\}\}, [y \mapsto \widehat{a}, z \mapsto D_q(\tilde{f})]).$$

Assume first that $a = *$. If $a \in \mathcal{L}(T, q)$ then we have $q = \circ$ so that Γ is an idle process graph. Conversely, assume that $\Gamma \to^* \Gamma_0$ where Γ_0 is idle. Since $\widehat{a} = \mathbf{0}$, no $f/\bar{f}$ reduction can occur in Γ and hence Γ itself must be idle, meaning that $q = \circ$ and hence $a \in \mathcal{L}(T, q)$.

Assume now that $a = f(a_1, a_2)$. If $a \in \mathcal{L}(T, q)$ then there must exist $q_1, q_2 \in Q$ such that $(q, f, (q_1, q_2)) \in \Theta$ with $a_i \in \mathcal{L}(T, q_i)$ for $i = 1, 2$. Setting

$$\Gamma^i = (\{y_i, z_i\}, \{\{y_i, z_i\}\}, [y_i \mapsto \widehat{a_i}, z_i \mapsto D_{q_i}(\tilde{f})])$$

for $i = 1, 2$ (with pairwise distinct vertices z_is and y_is), we know by inductive hypothesis that $\Gamma^i \to^* \Gamma^i_0$ where the Γ^i_0s are idle. Since $(q, f, (q_1, q_2)) \in \Theta$ we have $D_q(\tilde{f}) := \bar{f}(D_{q_1}(\tilde{f}), D_{q_2}(\tilde{f})) + s \in \mathsf{Def}_T$ for some (possibly null) sum s. On the other hand $\widehat{a} = f(\widehat{a_1}, \widehat{a_2})$. Therefore $\Gamma \to_{\{y,z\}} \Gamma' = (\{y_1, y_2, z_1, z_2\}, \{\{y_1, z_1\}, \{y_2, z_2\}\}, \pi)$ where $\pi(y_i) = \widehat{a_i}$ and $\pi(z_i) = D_{q_i}(\tilde{f})$ for $i = 1, 2$ (up to some $\Rrightarrow_{\mathsf{g}}$ steps). Since Γ^i reduces to an idle process graph for $i = 1, 2$, it follows that Γ' also reduces to an idle process graph.

If now $\Gamma \to^* \Gamma^0$ where Γ^0 is idle, since $\widehat{a} = f(\widehat{a_1}, \widehat{a_2})$, q cannot be the final state, and $D_q(\tilde{f})$ must $\Rrightarrow_{\mathsf{g}}$ normalize (by updating the value of this constant D_q using Def_T) to a sum of the form $\bar{f}(p_1, p_2) + s$ where the p_is are processes and s is a (possibly nil) sum. In view of the definition of Def_T, this means that, for some $q_1, q_2 \in Q$, one has $(q, f, (q_1, q_2)) \in \Theta$ and $p_i = D_{q_i}(\tilde{f})$ for $i = 1, 2$ and moreover the reduction starting from Γ is of the form

$$\begin{array}{l}\Gamma \Rrightarrow_{\mathsf{g}} (\{y, z\}, \{\{y, z\}\}, [y \mapsto f(\widehat{a_1}, \widehat{a_2}), z \mapsto \bar{f}(D_{q_1}(\tilde{f}), D_{q_2}(\tilde{f})) + s]) \\ \to_{\{y,z\}} \\ (\{y_1, y_2, z_1, z_2\}, \{\{y_1, z_1\}, \{y_2, z_2\}\}, [y_i \mapsto \widehat{a_i}, z_i \mapsto D_{q_i}(\tilde{f})]_{i \in 1,2})\end{array}$$

and, calling Γ' this latter graph process, we know that $\Gamma' \to^* \Gamma^0$. In view of the communications allowed by the graph structure of Γ', this implies that Γ^i reduces to an idle process graph for $i = 1, 2$, where $\Gamma^i = (\{y_i, z_i\}, \{\{y_i, z_i\}\}, [y_i \mapsto \widehat{a_i}, z_i \mapsto D_{q_i}(\tilde{f})])$. Therefore, by inductive hypothesis, $a_i \in \mathcal{L}(T, q_i)$ and hence $a \in \mathcal{L}(T, q)$ since $(q, f, (q_1, q_2)) \in \Theta$. □

3.2. Threads and shared memory

We illustrate here the fact that CBP naturally models thread-like synchronization, using shared memory, and process-like communication by means of remote procedure calls. We first model a simple one bit register[4]:

$$\begin{array}{lll}\mathsf{Reg}_0 & =_{\mathsf{def}} & r_0(\mathsf{Reg}_0) + w_0(\mathsf{Reg}_0) + w_1(\mathsf{Reg}_1) \\ \mathsf{Reg}_1 & =_{\mathsf{def}} & r_1(\mathsf{Reg}_1) + w_0(\mathsf{Reg}_0) + w_1(\mathsf{Reg}_1)\end{array}$$

Now a thread that attempts to read or write $b \in \{0, 1\}$ from the register is simply

$$\begin{array}{lll}\mathsf{Read}_b & =_{\mathsf{def}} & \bar{r}_b(\mathbf{0}) \\ \mathsf{Write}_b & =_{\mathsf{def}} & \bar{w}_b(\mathbf{0})\end{array}$$

Eventually we consider a server that may (indefinitely) spawn pairs of threads upon calling the function symbol sp:

$$\mathsf{Spawn} =_{\mathsf{def}} sp(\mathbf{0}, \mathbf{0}, \mathsf{Spawn})$$

[4]Note that the encoding of the one-bit register is done in a fragment of CBP that coincides with CCS (unary CBP).

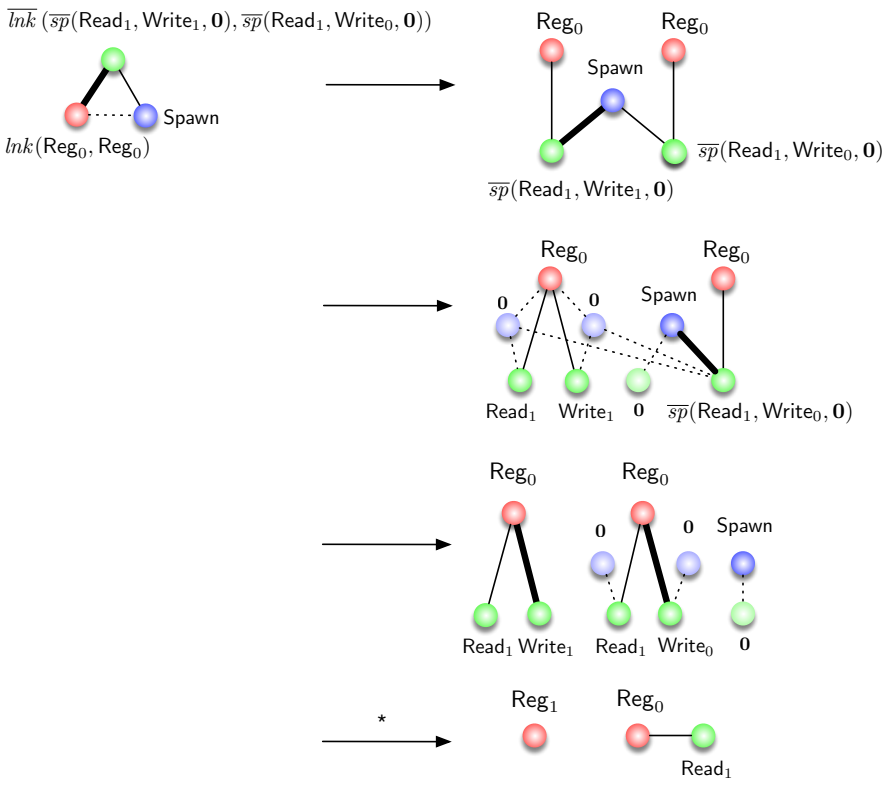


Figure 2: A computation of the threads processes (thick edges denote reduction redexes). Edges that connect two processes that have no complementary function symbols are dotted. Idle nodes are garbage collected.

Now consider the following process:

$$lnk(\mathsf{Reg}_0,\mathsf{Reg}_0) \mid \overline{lnk}\,(\overline{sp}(\mathsf{Read}_1,\mathsf{Write}_1,\mathbf{0}),\overline{sp}(\mathsf{Read}_1,\mathsf{Write}_0,\mathbf{0})) \mid \mathsf{Spawn}$$

its computation, which is depicted in Figure 2, proceeds as follows: two pairs of reader-writer threads are packed on the green node, guarded by two function symbols $\overline{lnk}$ and $\overline{sp}$. The lnk function is called at step 1, and links both packs of threads to different registers (both initialized to 0). The first and second pairs of threads are then spawned using the function sp at steps 2 and 3. Because the two combinations of threads do not share the same register, and the second register does not hold the value expected by the associated reader thread, the second combination of threads is not able to terminate successfully.

3.3. Graph self-assembly

Consider a specification process graph Γ, which describes how a computation should be distributed, and in particular how processes that are located on the vertices of Γ should be topologically separated. The self-assembly challenge is to conceive of a process p (the

implementation) that eventually produces the specification Γ, up-to garbage collection of idle nodes, as depicted in Figure 2. We finally show that CBP is able to realize non trivial assembly patterns.

Definition 3.1 (Convergence)**.** A process graph Γ *weakly converges* to Γ' if there exists a trace of the form $\Gamma \to^* \Gamma'$. We say that Γ *strongly converges* to Γ' if all traces $\Gamma \to^* \Gamma''$ either factor through Γ', or Γ'' strongly converges to Γ'.

Definition 3.2 (Homomorphism)**.** Let $\Gamma =_{\mathsf{def}} (V, E, \pi)$ and $\Gamma' =_{\mathsf{def}} (V', E', \pi')$. A process graph *homomorphism* $\phi : \Gamma \to \Gamma'$ is an injective map $\phi : V \hookrightarrow V'$ that is:

- Edge preserving : $\{x, y\} \in E$ implies $\{\phi(x), \phi(y)\} \in E'$.
- Label preserving: $\pi'\phi(x) = \pi(x)$, for all $x \in V$.

We say that a homomorphism $\phi : \Gamma \to \Gamma'$ is a *quasi-iso* if it is, in addition:

- Edge reflecting: $\{\phi(x), \phi(y)\} \in E'$ implies $\{x, y\} \in E$
- Full: for all $x' \in V' \setminus \phi(V)$, $\pi'(x') = \mathbf{0}$.

We write $\Gamma \preceq_\phi \Gamma'$ whenever there is a quasi-iso ϕ from Γ to Γ'.

Proposition 3.2. *The relation* $\approx =_{\mathsf{def}} \{(\Gamma, \Gamma') \mid \exists \phi : \Gamma \preceq_\phi \Gamma'\}$ *is a bisimulation, i.e it is a symmetric relation that satisfies:*

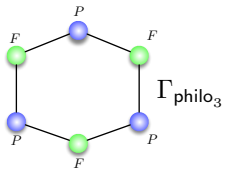
$$\Gamma \approx \Gamma' \implies \forall \Gamma \to \Gamma_0, \exists \Gamma' \to \Gamma_1$$

such that $\Gamma_0 \approx \Gamma_1$.

Definition 3.3 (Assembly)**.** A process p (strongly or weakly) *assembles* into Γ_{spec}, with $\Gamma_{\mathsf{spec}} =_{\mathsf{def}} (V_{\mathsf{s}}, E_{\mathsf{s}}, \pi_{\mathsf{s}})$, if $\{x : p\}$ (strongly or weakly) converges to a graph $\Gamma = (V, E, \pi)$ such that $\Gamma_{\mathsf{spec}} \preceq_\phi \Gamma$.

For simplicity, in the following example we consider only function symbols that are self dual, i.e $f = \bar{f}$ for all f. Consider the following process definitions and the process graph specification of the 3 dining philosophers:

$$\begin{array}{lll} \mathsf{P} & =_{\mathsf{def}} & take(take(\mathsf{Eat}) + put(\mathsf{P})) \\ \mathsf{Eat} & =_{\mathsf{def}} & put(put(\mathsf{P})) \\ \mathsf{F} & =_{\mathsf{def}} & take(put(\mathsf{F})) \end{array}$$



$\Gamma_{\mathsf{philo}_3}$

Consider the process:

$$p =_{\mathsf{def}} g(h(P, P), i(\mathbf{0}, w(P))) \parallel f(i(F, \mathbf{0}), w(h(F, F))) \parallel f(\mathbf{0}, g(\mathbf{0}, \mathbf{0}))$$

Figure 3 illustrates the proof of p's assembly into $\Gamma_{\mathsf{philo}_3}$. We obtain the desired hexagon by transforming a triangle graph, which is the normalization of the process graph $\{x : p\}$, following the reductions of Fig. 3. Importantly the reduction $\Gamma_3 \to \Gamma_4$ is enforced by symbol w which is positioned to prevent the reduction of symbol h before symbol i that would entail weak convergence only. Without symbol w, from Γ_2, one may reduce

symbol h to reach a state where two P processes have access to a single fork each. Now a philosopher may enter into infinite interactions with the fork by taking it and putting it back without letting the whole process ever reach the specification graph. Strong convergence requires that process p is able to assemble into the 3 dining philosophers in an atomic fashion: no philosopher may start taking forks before *all* philosophers are able to do so.

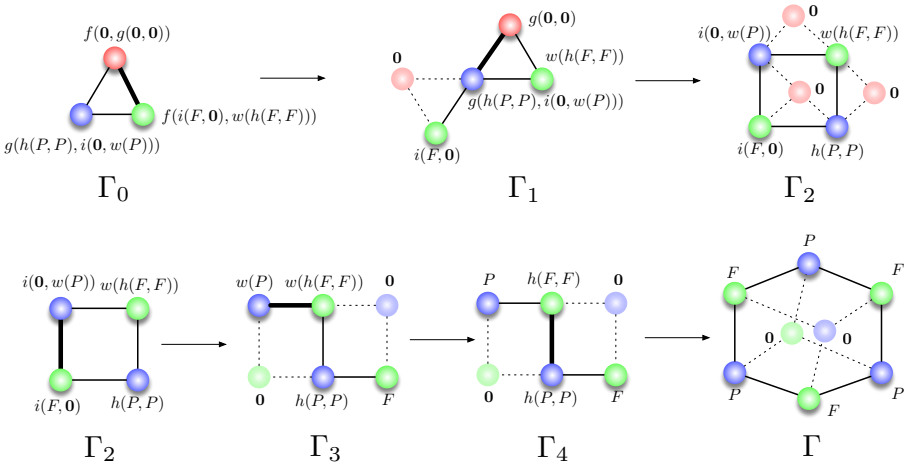


Figure 3: Proof of the assembly of p into the 3-dining philosophers. Nodes labeled with a nil process as well as edges connecting to them are faded out. Reductions are applied *modulo* quasi-iso.

4. Compositional semantics

In this section, we introduce a labeled transition system (LTS) for CBP. It relies on a localization of top-level function symbols using a form of locality [5, 6] or proof terms [3, 4]. The main idea behind the LTS is to "freeze" a process graph Γ into a tree t whose leaves correspond to nodes of Γ, and whose remaining nodes contain information allowing one to decide whether the leaves of the sub-tree are connected. The correspondence between CBP trees and CBP graphs is depicted in Figure 4. We formally address this correspondence in Section 5.

4.1. Syntax

Along with CBP processes, defined in Section 2:

$$\begin{array}{rcl} s & ::= & f(p,p) \mid \mathbf{0} \mid D(\tilde{f}) \mid s + s \\ p & ::= & s \mid (p \parallel p) \mid p \setminus \mathcal{H} \end{array}$$

we introduce CBP *process trees*, which provide an algebraic representation of graphs

$$t ::= p \mid (t \oplus_A t) \mid t \setminus \mathcal{H}$$

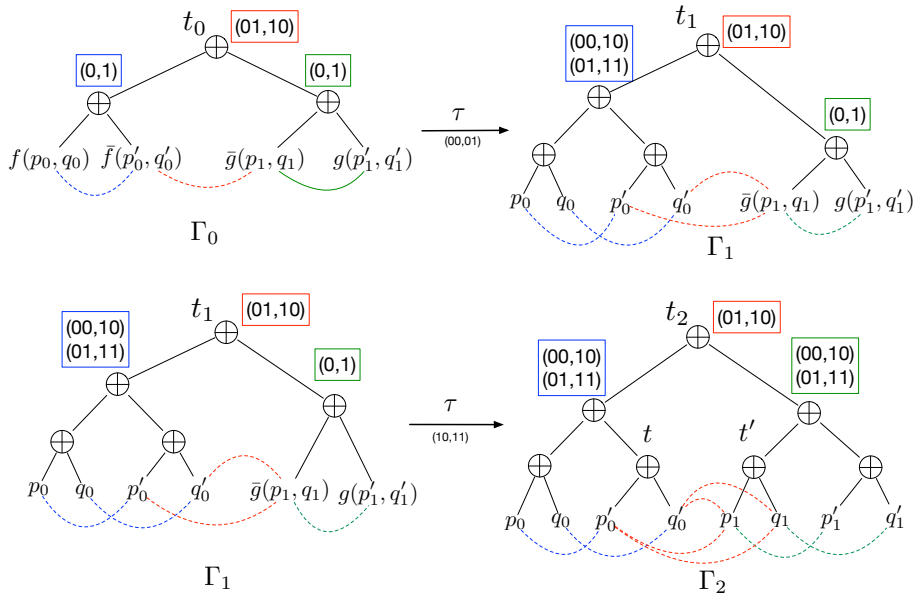


Figure 4: Representation of CBP graphs (Γ_0, Γ_1 and Γ_2) as trees (t_0, t_1 and t_2). Each leaf of the process tree t_i contains a different vertex of the process graph Γ_i (for $i \in \{0,1,2\}$). Roots of (sub)trees are labeled by a list of pairs of addresses (in the example: the blue, red and green rectangles, subtrees labeled by the empty list are pictured unlabeled). Addresses in these pairs are relative to the position of the root containing the list: 0 stands for the left subtree and 1 stands for the right subtree. The process graph Γ_i denoted by the process tree t_i is obtained by adding edges connecting two leaves (dotted in the picture) as indicated by the lists of address pairs in the root above them. For instance, the red router of process tree t_2, associates address 01 to address 10, which point respectively to the subtrees $t = p'_0 \oplus q'_0$ and $t' = p_1 \oplus q_1$. This implies that all leaves of t and t' should be pairwise connected in the corresponding process graph Γ_2. Note that the leaves labeled by p_1 and q_1 in Γ_2 are not connected since the subtree t' is labeled by the empty list.

where $A \in 2^{(\{0,1\}^+ \times \{0,1\}^+)}$. We use $\mathcal{T}$ to denote the set of process trees generated by non terminal symbol t. As presented in Fig 4, operators $\oplus_A$, that we call *routers*, act as "regulators" for communication. In the tree $t_0 \oplus_A t_1$, a function symbol originating from the subtree t_0 will only be able to synchronize with a dual symbol in t_1 if the table A allows them to do so. Router table A is updated upon synchronization between t_0 and t_1 as shown in the next section.

In the meantime we introduce an important function that allows one to unfold a leaf labeled with $p \parallel q$ into a tree. To do so, process trees are equipped with a *distribution* function, $\mathsf{dist} : \mathcal{P} \to \mathcal{T}$ and that is defined as:

$$\begin{array}{lll} \mathsf{dist}(s) & =_{\mathsf{def}} & s \\ \mathsf{dist}(p \parallel p') & =_{\mathsf{def}} & p \oplus_{\{(0,1)\}} p' \\ \mathsf{dist}(p \setminus \mathcal{H}) & =_{\mathsf{def}} & p \setminus \mathcal{H} \end{array}$$

4.2. Router update

We consider *addresses* (meta-variables $i, j, k, \ldots$), which are words over the alphabet $\{0,1\}$. For all addresses i, j, we write $i \leq j$ if $j = i.k$ for some address k (j is more

precise than i).

For all $X \in 2^{\{0,1\}^*}$ we define $\uparrow X =_{\mathsf{def}} \{i \in \{0,1\}^* \mid \exists j \in X : j \leq i\}$. Similarly, we use $\downarrow X$ to denote the set of addresses that are less precise than those in X.

For all $A \in 2^{(\{0,1\}^+ \times \{0,1\}^+)}$, the set of *edges denoted by* A is

$$\mathcal{E}(A) =_{\mathsf{def}} \{(i,j) \mid \exists (i',j') \in A : i \in \uparrow\{i'\} \wedge j \in \uparrow\{j'\}\}$$

After a synchronization has occurred, the table A needs to be updated to take into account the change in the topology of permissible communications, as specified in Fig. 1. We turn now to the formal description of this router update mechanism. For all $I \in 2^{\{0,1\}^*}$ and $b \in \{0,1\}$, we use the meta operation $b \cdot I$ defined as $b \cdot I =_{\mathsf{def}} \{b.i \mid i \in I\}$.

For all $b \in \{0,1\}$ we write $\neg b$ to denote $(1-b)$. We define $\bar{\cdot} : \{0,1\}^* \to 2^{\{0,1\}^*}$ as:

$$\begin{array}{rcl} \overline{\epsilon} & =_{\mathsf{def}} & \emptyset \\ \overline{b.i} & =_{\mathsf{def}} & \{\neg b\} \cup b \cdot \bar{i} \end{array}$$

For all $i' \leq i$ we define $i' \restriction i$ as:

$$\begin{array}{rcl} \epsilon \restriction i & =_{\mathsf{def}} & \bar{i} \\ b.i' \restriction b.i & =_{\mathsf{def}} & b \cdot (i' \restriction i) \end{array}$$

We leave $i \restriction j$ undefined if $i \not\leq j$.

Lemma 4.1. *For all $i \leq j$, $\uparrow(i \restriction j) = \uparrow\{i\} \setminus \uparrow\{j\}$.*

For instance, $10 \restriction 1011 =_{\mathsf{def}} 10 \cdot \{0, 10\}$, which is $\{100, 1010\}$, and $\uparrow\{100, 1010\}$ contains all addresses above 10 that are not above 1011.

For all $i \in \{0,1\}^*$ and all $I \subseteq \downarrow\{i\}$, we write $I \restriction i$ for $\bigcup_{i' \in I} i' \restriction i$. Now, consider a set $A \in 2^{(\{0,1\}^* \times \{0,1\}^*)}$ and two addresses $i, j \in \{0,1\}^*$ and let

$$\begin{array}{rcl} I_A & =_{\mathsf{def}} & \{i' \in \{0,1\}^* \mid \exists k : (i',k) \in A \wedge i' \leq i\} \\ J_A & =_{\mathsf{def}} & \{j' \in \{0,1\}^* \mid \exists k : (k,j') \in A \wedge j' \leq j\} \end{array}$$

One defines:

$$\delta_{i,j}(A) =_{\mathsf{def}} A \setminus (\downarrow\{i\} \times \downarrow\{j\}) \cup (I_A \times (J_A \restriction j) \cup (I_A \restriction i) \times J_A)$$

Theorem 4.2. *The effect of $\delta_{i,j}(A)$ is to prevent all synchronizations between localities that are more precise than (i,j) and to preserve all other previously possible communications, i.e.:*

$$\mathcal{E}(\delta_{i,j}(A)) = \mathcal{E}(A) \setminus \mathcal{E}(\{i,j\})$$

Definition 4.1 (Router update)**.** For all $i, j \in \{0,1\}^*$, the router *update* is:

$$\mathsf{upd}(A,i,j) =_{\mathsf{def}} \delta_{i,j}(A) \cup \{(i.0, j.0), (i.1, j.1)\}$$

Corollary. *For all $A \in 2^{\{0,1\}^* \times \{0,1\}^*}$ and $i, j \in \{0,1\}^*$:*

$$\mathcal{E}(\mathsf{upd}(A,i,j)) = (\mathcal{E}(A) \setminus \mathcal{E}(\{i,j\})) \cup \mathcal{E}(\{(i.0, j.0), (i.1, j.1)\})$$

4.3. labeled transition system

Let $\alpha \in \mathcal{F} \cup \{\tau\}$ and $\theta \in \{0,1\}^* + (\{0,1\}^* \times \{0,1\}^*)$. For all (i,j), let $b \cdot (i,j) =_{\mathsf{def}} (b.i, b.j)$. The compositional semantics of CBP is induced by the following derivations:

$$\frac{}{f(p_1,p_2) \xrightarrow{\epsilon:f} p_1 \oplus_\emptyset p_2} \ (\mathsf{act})$$

$$(\mathsf{sum}-\mathsf{left}) \ \frac{s_1 \xrightarrow{\epsilon:f} t}{s_1 + s_2 \xrightarrow{\epsilon:f} t} \qquad \frac{s_2 \xrightarrow{\epsilon:f} t}{s_1 + s_2 \xrightarrow{\epsilon:f} t} \ (\mathsf{sum}-\mathsf{right})$$

$$\frac{s \xrightarrow{\epsilon:f} t \quad (D(\tilde{h}) := s\{\tilde{h}/\tilde{g}\}) \in \mathsf{Def}}{D(\tilde{g}) \xrightarrow{\epsilon:f} t} \ (\mathsf{link})$$

$$\frac{t_0 \xrightarrow{i:f} t_0' \quad t_1 \xrightarrow{j:\bar{f}} t_1' \quad (0.i, 1.j) \in \mathcal{E}(A) \quad B = \mathsf{upd}(A, 0.i, 1.j)}{t_0 \oplus_A t_1 \xrightarrow{(0.i,1.j):\tau} t_0' \oplus_B t_1'} \ (\mathsf{synch})$$

$$(\mathsf{res}) \ \frac{t \xrightarrow{\theta:\alpha} t' \quad \alpha \notin \mathcal{H} \cup \overline{\mathcal{H}}}{t \setminus \mathcal{H} \xrightarrow{\theta:\alpha} t' \setminus \mathcal{H}} \qquad \frac{\mathsf{dist}(p) \xrightarrow{\theta:\alpha} t'}{p \xrightarrow{\theta:\alpha} t'} \ (\mathsf{dist})$$

$$\frac{t \xrightarrow{\theta:\alpha} t'}{t \oplus_A t'' \xrightarrow{0\cdot\theta:\alpha} t' \oplus_A t''} \ (\mathsf{par}-\mathsf{left}) \qquad \frac{t \xrightarrow{\theta:\alpha} t'}{t'' \oplus_A t \xrightarrow{1\cdot\theta:\alpha} t'' \oplus_A t'} \ (\mathsf{par}-\mathsf{right})$$

5. Correctness of the labeled transition system

This section is dedicated to the proof of correctness of the LTS semantics of CBP introduced in Section 4. In Section 5.1 we show that any rigid process tree t can "thaw" into a process graph Γ. This "thawing" relation is defined by induction on the structure of t and produces eventually a bijective map that associates leaves of t to nodes of Γ. Theorem 5.1 then formalizes the correctness of the LTS as a bisimulation between the τ-transitions of CBP trees and the reductions that can be triggered from their thawed versions. The (main parts of the) proof of the theorem is split into Section 5.2, where we show that reductions can simulate the LTS, and Section 5.3 where the LTS is shown to simulate the reductions.

5.1. Thawing trees into graphs

For all $t \in \mathcal{T}$ define $|t| \subseteq 2^{\{0,1\}^*}$ as:

$$|p| =_{\mathsf{def}} \{\epsilon\} \qquad |t \oplus_A t'| =_{\mathsf{def}} 0 \cdot |t| \cup 1 \cdot |t'|$$

We define the *thaw* relation $t \rightsquigarrow_\phi (V, E, \pi)$, where ϕ is a bijective map $\phi : |t| \to V$, as the least relation satisfying:

$$\frac{}{p \rightsquigarrow_{[\epsilon \mapsto x]} \{x : p\}} \tag{3}$$

$$\frac{t \setminus \mathcal{H} \rightsquigarrow_\phi \Gamma \quad \mathsf{fresh}(g)}{t \setminus (f \uplus \mathcal{H}) \rightsquigarrow_\phi (V, E, \pi\{g/f\})} \tag{4}$$

$$\frac{t_0 \rightsquigarrow_{\phi_0} (V_0, E_0, \pi_0) \quad t_1 \rightsquigarrow_{\phi_1} (V_1, E_1, \pi_1) \quad \forall i \in |t_0| \cup |t_1|, \forall b \in \{0,1\} : \phi(b.i) = \phi_b(i)}{t_0 \oplus_A t_1 \rightsquigarrow_{\phi} (V_0 \uplus V_1, E_0 \uplus E_1 \uplus E_A, \pi_0 + \pi_1)} \quad (5)$$

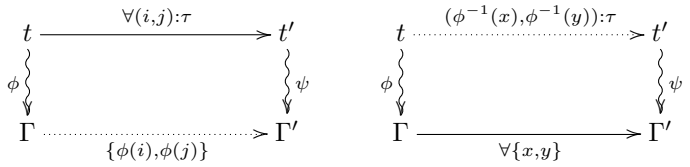
with:

$$E_A =_{\mathsf{def}} \{\{x, y\} \in V_0 \times V_1 \mid \big(0 \cdot (\phi_0^{-1}(x)), 1 \cdot (\phi_1^{-1}(y))\big) \in \mathcal{E}(A)\}$$

Theorem 5.1. *The relation:*

$$\sim =_{\mathsf{def}} \{(t, \Gamma) \mid \exists \phi : t \rightsquigarrow_{\phi} \Gamma\}$$

is a bisimulation between process graphs reductions, and τ-transitions of process trees, i.e. *for all ϕ, there exists a ψ that satisfies the following diagrams:*



5.2. Process graph reductions simulate τ-transitions

Structural congruence for processes $\equiv_{\mathsf{p}} \subseteq \mathcal{P} \times \mathcal{P}$ is defined as $\equiv_{\mathsf{p}} =_{\mathsf{def}} \left(\Rrightarrow_{\mathsf{p}} \cup \Rrightarrow_{\mathsf{p}}^{-1} \cup \doteq_{\mathsf{p}}\right)$ where:

$$\begin{array}{lll} s + s' & \doteq_{\mathsf{p}} & s' + s \\ (s + s') + s'' & \doteq_{\mathsf{p}} & s + (s' + s'') \\ s + \mathbf{0} & \doteq_{\mathsf{p}} & s \end{array}$$

and structural congruence for process graphs is $\equiv_{\mathsf{g}} =_{\mathsf{def}} \left(\Rrightarrow_{\mathsf{g}} \cup \Rrightarrow_{\mathsf{g}}^{-1} \cup \doteq_{\mathsf{g}}\right)$ with:

$$\begin{array}{lll} (V, E, \pi) & \doteq_{\mathsf{g}} & (V, E, \pi\{g/f\}) \text{ if } g \notin \mathsf{fn}(\pi) \\ (V \cup \{x\}, E, \pi + [x \mapsto p]) & \doteq_{\mathsf{g}} & (V \cup \{x\}, E, \pi + [x \mapsto q]) \text{ if } p \equiv_{\mathsf{p}} q \end{array}$$

Lemma 5.2. *$\equiv_{\mathsf{g}}$ is a bisimulation for process graph reduction semantics.*

The above lemma insures that it is safe to consider process graphs up-to consistent substitution of function symbols. There is an equivalent proposition for process trees:

Definition 5.1. Consider $\Rrightarrow_{\alpha} \subseteq \mathcal{T} \times \mathcal{T}$ that is the least relation satisfying:

$$\frac{t_0 \Rrightarrow_{\alpha} t_0' \quad t_1 \Rrightarrow_{\alpha} t_1'}{t_0 \oplus_A t_1 \Rrightarrow_{\alpha} t_0' \oplus_A t_1'} \qquad \frac{t' = t\{g/f\} \quad \mathsf{fresh}(g)}{t \setminus (\{f\} \uplus \mathcal{H}) \Rrightarrow_{\alpha} t' \setminus \mathcal{H}}$$

Lemma 5.3. *The relation $\Rrightarrow_{\alpha}$ is a bisimulation for CBP's labeled transition system.*

This lemma allows us to consider terms of CBP with restrictions occurring exclusively inside recursive definition bodies and to safely erase them upon definition unfolding. Note that this procedure is not needed for the operational semantics but simplifies greatly the proofs.

Definition 5.2. Let t a process tree. For all $i \in \downarrow |t|$, we define $t@i$ by induction on i:

$$\begin{array}{lll} t@\epsilon & =_{\mathsf{def}} & t \\ (t \oplus_A t')@0.i & =_{\mathsf{def}} & t@i \\ (t \oplus_A t')@1.i & =_{\mathsf{def}} & t'@i \\ p@b.i & & \text{undefined for all } b \in \{0,1\} \end{array}$$

Lemma 5.4. *For all transition* $t \xrightarrow{i:f} t'$*, there exist* $p_0, p_1 \in \mathcal{P}$ *such that* $t@i \equiv_{\mathsf{p}} f(p_0, p_1) + s$ *for some (possibly null)* s*.*

Lemma 5.5. *Let* $t \rightsquigarrow_\phi \Gamma =_{\mathsf{def}} (V, E, \pi)$*. For all* $i \in |t|$*, if* $t@i = p$ *then* $\pi \circ \phi(i) = p$*.*

Lemma 5.6. *For all process tree* t *and process graph* $\Gamma =_{\mathsf{def}} (V, E, \pi)$*, the following derivation holds* $\forall b \in \{\mathtt{0}, \mathtt{1}\}$*:*

$$\frac{\left\{\begin{array}{l} t \xrightarrow{i:f} t' \\ t \rightsquigarrow_\phi \Gamma \end{array}\right. \quad x_b \notin V \quad \Gamma' \equiv_{\mathsf{g}} \Gamma \quad \left\{\begin{array}{l} \forall j \in |t'| \setminus \uparrow\{i\} : \psi(j) = \phi(j) \\ \psi(i.b) = x_b \ \ else \end{array}\right.}{t' \rightsquigarrow_\psi \Gamma' \left[x_0 : t'@i.\mathtt{0}, x_1 : t'@i.\mathtt{1}/\phi(i)\right]}$$

Theorem 5.1, left diagram. We wish to show:

$$t \xrightarrow{(i,j):\tau} t' \wedge t \rightsquigarrow_\phi \Gamma \implies \exists \Gamma', \psi : \Gamma \rightarrow_{\{\phi(i),\phi(j)\}} \Gamma' \wedge t' \rightsquigarrow_\psi \Gamma'$$

We proceed by induction on the derivation of the τ-transition.

- Base case is $t \xrightarrow{(\mathtt{0}.i,\mathtt{1}.j):\tau} t'$, for some $i, j \in \{\mathtt{0}, \mathtt{1}\}^*$, by application of rule (synch). We have:

$$\frac{t_0 \xrightarrow{i:f} t'_0 \quad t_1 \xrightarrow{j:\bar{f}} t'_1 \quad (\mathtt{0}.i, \mathtt{1}.j) \in \mathcal{E}(A) \quad B = \mathsf{upd}_{(\mathtt{0}.i,\mathtt{1}.j)}(A)}{t = t_0 \oplus_A t_1 \xrightarrow{(\mathtt{0}.i,\mathtt{1}.j):\tau} t'_0 \oplus_B t'_1 = t'} \tag{6}$$

  Since $t \rightsquigarrow_\phi \Gamma$ we can use derivation (5) to deduce:

$$t_0 \rightsquigarrow_{\phi_0} (V_0, E_0, \pi_0) \qquad t_1 \rightsquigarrow_{\phi_1} (V_1, E_1, \pi_1)$$

  for some V_i, E_i, π_i satisfying $\Gamma = (V_0 \uplus V_1, E_0 \uplus E_1 \uplus E_A, \pi)$. We apply Lemma 5.4, to the premises of derivation (6) and we obtain:

$$t_0@i \equiv_{\mathsf{p}} f(p_0, p_1) + s \qquad t_1@j \equiv_{\mathsf{p}} \bar{f}(p'_0, p'_1) + s'$$

  We apply Lemma 5.5 to deduce $\pi_0 \circ \phi_0(i) \equiv_{\mathsf{p}} f(p_0, p_1) + s$ and $\pi_1 \circ \phi_1(j) \equiv_{\mathsf{p}} \bar{f}(p'_0, p'_1) + s'$. Now according to the premises of derivation (5), $\phi(\mathtt{0}.i) = \phi_0(i)$ and $\phi(\mathtt{1}.j) = \phi_1(j)$. Hence we have:

$$\pi \circ \phi(\mathtt{0}.i) \equiv_{\mathsf{p}} f(p_0, p_1) + s \qquad \pi \circ \phi(\mathtt{1}.j) \equiv_{\mathsf{p}} \bar{f}(p'_0, p'_1) + s'$$

  Since $\phi_0(i) \in V_0$ and $\phi_1(j) \in V_1$ and $V = V_0 \uplus V_1$ we have $\phi(\mathtt{0}.i) \in V$ and $\phi(\mathtt{1}.j) \in V$. It remains to verify that $\phi(\mathtt{0}.i)$ and $\phi(\mathtt{1}.j)$ are connected in Γ. The only possibility is to verify that $\{\phi(\mathtt{0}.i), \phi(\mathtt{1}.j)\} \in E_A$. It is indeed the case if:

$$\left(\mathtt{0} \cdot \phi_0^{-1}(\phi(\mathtt{0}.i)), \mathtt{1} \cdot \phi_1^{-1}(\phi(\mathtt{1}.j))\right) \in \mathcal{E}(A)$$

  which is equivalent to:

$$\left(\mathtt{0} \cdot \phi_0^{-1}(\phi_0(i)), \mathtt{1} \cdot \phi_1^{-1}(\phi_1(j))\right) \in \mathcal{E}(A)$$

  and reduces to $(\mathtt{0}.i, \mathtt{1}.j) \in \mathcal{E}(A)$ which holds by the premises of derivation (6). We can conclude that $\Gamma \rightarrow_{\{\phi(\mathtt{0}.i),\phi(\mathtt{1}.j)\}} \Gamma'$ since all preconditions to trigger a reduction

of the process graph Γ hold. It remains to verify that $t' = t'_0 \oplus_B t'_1 \rightsquigarrow_\psi \Gamma'$ for some ψ. We apply Lemma 5.6 to the premises of

$$\frac{t_0 \xrightarrow{i:f} t'_0 \quad t_1 \xrightarrow{j:\bar{f}} t'_1}{t_0 \oplus_A t_1 \xrightarrow{0.i,1.j:\tau} t'_0 \oplus_B t'_1}$$

and derivation (5) applied to $t_0 \oplus_A t_1 \rightsquigarrow_\phi \Gamma$, and we obtain $t'_0 \rightsquigarrow_{\phi'_0} \Gamma'_0$ and $t'_1 \rightsquigarrow_{\phi'_1} \Gamma'_1$. Let $\Gamma'_b = (V'_b, E'_b, \pi'_b)$. We apply derivation (5) to deduce $t'_0 \oplus_B t'_1 \rightsquigarrow_\psi \Gamma'$ with $\Gamma' = (V'_0 \uplus V'_1, E'_0 \uplus E'_1 \uplus E_B, \pi'_0 + \pi'_1)$.

- We conclude by applying a straightforward induction for the contextual rules in the particular case of a τ-transition.

□

5.3. τ-transitions simulate process graph reductions

In order to prove the right diagram of Theorem 5.1 we need a couple of definitions and lemmas.

Definition 5.3 (dist-normal form)**.** We define the dist-normalization function $\mathsf{dnf} : \mathcal{T} \to \mathcal{T}$ as:

$$\mathsf{dnf}(t \oplus_A t') =_{\mathsf{def}} \mathsf{dnf}(t) \oplus_A \mathsf{dnf}(t') \qquad \mathsf{dnf}(t \setminus \mathcal{H}) =_{\mathsf{def}} \mathsf{dnf}(t) \setminus \mathcal{H}$$

$$\mathsf{dnf}(\mathbf{0}) =_{\mathsf{def}} \mathbf{0} \qquad \mathsf{dnf}(p) =_{\mathsf{def}} \mathsf{dist}(p)$$

Note that $\mathsf{dnf}(t)$ returns after exactly $|t|$ calls to dist.

Lemma 5.7. *Let $\mathcal{D} =_{\mathsf{def}} \{(t, t') \mid \mathsf{dnf}(t) = \mathsf{dnf}(t')\}$. The relation $\mathcal{D}$ is a bisimulation for CBP's labeled transition system.*

Lemma 5.8. *For all $\Gamma, \Gamma' \in \mathcal{G}$ and all $t \in \mathcal{T}$ satisfying $\Gamma \Rrightarrow_{\mathsf{g}} \Gamma'$ and $t \rightsquigarrow_\phi \Gamma$, there exists ψ such that $\mathsf{dnf}(t) \rightsquigarrow_\psi \Gamma'$.*

Lemma 5.9. *For all t and $\Gamma = (V, E, \pi)$, the following derivation holds:*

$$\frac{t \rightsquigarrow_\phi \Gamma \quad x \in V \quad \pi(x) \equiv_{\mathsf{p}} f(p, q) + s \quad i = \phi^{-1}(x)}{t \xrightarrow{i:f} t' \rightsquigarrow_\psi \Gamma[x_0 : p, x_1 : q/x]} \quad \textit{for some } \psi : |t'| \to V.$$

We have now everything in place to prove the final part of the theorem.

Theorem 5.1, right diagram. We need to show

$$t \rightsquigarrow_\phi \Gamma \wedge \Gamma \to_{\{x,y\}} \Gamma' \wedge i = \phi^{-1}(x) \wedge j = \phi^{-1}(y) \implies t \xrightarrow{(i,j):\tau} t' \rightsquigarrow_\psi \Gamma' \tag{7}$$

We prove this derivation by induction on the derivation of the reduction $\Gamma \to_{\{x,y\}} \Gamma'$.

- Suppose $\Gamma \to_{\{x,y\}} \Gamma'$ by application of reduction rule (1). We show statement (7) by induction on the derivation of $t \rightsquigarrow_\phi \Gamma$.
  - Base case: if $t = p$ then no reduction is possible from Γ through reduction rule (1).

– Suppose $t = t_0 \oplus_A t_1 \rightsquigarrow_\phi \Gamma$. By derivation (5) we have $t_0 \rightsquigarrow_{\phi_0} \Gamma_0$ and $t_1 \rightsquigarrow_{\phi_1} \Gamma_1$. If $x \in V_0$ and $y \in V_1$ then we can apply Lemma 5.9, rule (synch) and derivation (5) to obtain the derivation:

$$\frac{\dfrac{t_0 \rightsquigarrow_{\phi_0} \Gamma_0 \quad \pi_0(x) = f(p,q) + s}{t_0 \xrightarrow{i:f} t'_0 \rightsquigarrow_{\phi'_0} \Gamma_0[x_0 : p, x_1 : q/x]} \quad \dfrac{t_1 \rightsquigarrow_{\phi_1} \Gamma_1 \quad \pi_1(y) = \bar{f}(p',q') + s'}{t_1 \xrightarrow{j:\bar{f}} t'_1 \rightsquigarrow_{\phi'_1} \Gamma_1[y_0 : p', y_1 : q'/y]}}{t_0 \oplus_A t_1 \xrightarrow{(i,j):\tau} t'_0 \oplus_B t'_1 \rightsquigarrow_\psi \Gamma''}$$

with $\Gamma'' = (V'', E'', \pi'')$ with

$$\begin{array}{lcl} V'' & = & V_0[x_0, x_1/x] \uplus V_1[y_0, y_1/y] \\ E'' & = & E_0[x_0, x_1/x] \uplus E_1[y_0, y_1/y] \uplus E_B \\ \pi'' & = & \pi_0[x_0 : p, x_1 : q/x] + \pi_1[y_0 : p', y_1 : q'/y] \end{array}$$

We have $t' = t'_0 \oplus_B t'_1$ and by derivation (5) we have:

$$E_B =_{\mathsf{def}} \left\{\{x', y'\} \in V'_0 \times V'_1 \mid \left(0 \cdot (\phi_0'^{-1}(x')), 1 \cdot (\phi_1'^{-1}(y'))\right) \in \mathcal{E}(B)\right\}$$

with $V'_0 = V_0[x_0, x_1/x]$ and $V'_1 = V_1[y_0, y_1/y]$.
Now, CBP reduction rule (1) produces $\Gamma' =_{\mathsf{def}} (V', E', \pi')$, where according to Def. 2.1:

$$\begin{array}{lcl} V' & =_{\mathsf{def}} & V[x_0, x_1/x][y_0, y_1/y] \\ E' & =_{\mathsf{def}} & (E \setminus \{\{x, y\}\})[\tilde{x}_i/x][\tilde{y}_i/y] \cup \{\{x_0, y_0\}, \{x_1, y_1\}\} \\ \pi' & =_{\mathsf{def}} & \pi \restriction (V \setminus \{x, y\}) + [x_0 \mapsto p; x_1 \mapsto q; y_0 \mapsto p'; y_1 \mapsto q'] \end{array}$$

We have $V'' = V'$ and $\pi'' = \pi'$. It remains to verify that $E'' = E'$. We decompose $E = E_0 \uplus E_1 \uplus E_{01}$ where $E_{01} = \{\{z, t\} \in V_0 \times V_1 \mid \{z, t\} \in E\}$. Now we deduce:

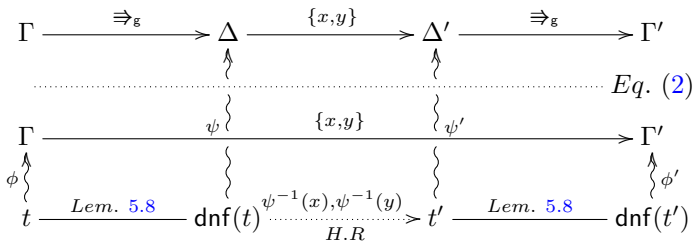
$$E \setminus \{\{x, y\}\} = (E_0 \uplus E_1 \uplus E_{01}) \setminus \{\{x, y\}\} = E_0 \uplus E_1 \uplus (E_{01} \setminus \{\{x, y\}\})$$

since $x \in V_0$ and $y \in V_1$. We have:

$$\begin{array}{l} E' = (E_0 \uplus E_1 \uplus (E_{01} \setminus \{\{x, y\}\}))[\tilde{x}_i/x][\tilde{y}_i/y] \cup \{\{x_0, y_0\}, \{x_1, y_1\}\} \\ \quad = E_0[\tilde{x}_i/x] \uplus E_1[\tilde{y}_i/y] \uplus (E_{01} \setminus \{\{x, y\}\} \cup \{\{x_0, y_0\}, \{x_1, y_1\}\}) \end{array}$$

Since $E_{01} \setminus \{\{x, y\}\} \cup \{\{x_0, y_0\}, \{x_1, y_1\}\} = E_B$ we conclude that $E' = E''$ and hence $\Gamma' = \Gamma''$.

- Suppose $\Gamma \rightarrow_{\{x,y\}} \Gamma'$ by application of the reduction rule (2). We summarize the inductive step in the diagram below.



We conclude that $t \xrightarrow{i,j:\tau} t'$, for $i = \psi^{-1}(x)$ and $j = \psi^{-1}(y)$, by Lemma 5.7. Since $\psi(i) = \phi(i)$ for all i in $|t|$, this terminates the proof. □

□

6. Conclusion

We have introduced CBP, a calculus of branching processes, which is a generalization of CCS with branching continuations. Importantly, CBP is not conceived as an *ad hoc* calculus aiming at capturing a particular feature of concurrent systems. Yet, this arguably natural generalization of CCS provides powerful means to model a combination of distribution, thanks to non trivial communication graphs, and synchronization. Several theoretical investigations are suggested by the present work:

- *A structural theory for process graphs.* As pointed out in Section 3 we lack a theory of reachable process graphs in order to establish expressivity results. Being able to characterize whether a graph Γ can be obtained by successive reductions from a clique would characterize precisely which specification graphs are amenable to assembly from a purely local initial process.

- *Self-assembly.* More generally, techniques to produce a CBP implementation that converges to a specification need to be formally established. We conjecture that unlabeled process graph reductions, where one is interested in the assembly of a particular graph, without considering the way nodes are labeled, should play an important role. Also given weak convergence between $\{x : p\}$ and Γ, it would be interesting to establish under which conditions one may automatically obtain strong convergence, by transformation of p.

- *Other classes of automata.* Following the CBP approach, one may wonder what class of concurrent systems corresponds to other types of communicating automata. In particular, possible connections with pushdown automata [2] and final state machines [8] would be worth studying.

- *Behavioral equivalences.* CBP is a localized process calculus. Locality aware bisimulations [5, 1] should naturally provide tools to compare branching processes.

Acknowledgment

This work is supported by the Chinese Academy of Science-INRIA project *Verification, Interactions and proofs*, VIP GJHZ1844.

References

[1] Bednarczyk, M.A., 1991. Hereditary History Preserving Bisimulations or What is the Power of the Future Perfect in Program Logics. Technical Report. ICS PAS.

[2] Bouajjani, A., Esparza, J., Schwoon, S., Strejček, J., 2005. Reachability analysis of multithreaded software with asynchronous communication, in: Sarukkai, S., Sen, S. (Eds.), FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, Springer Berlin Heidelberg. pp. 348–359.

[3] Boudol, G., Castellani, I., 1988. A non-interleaving semantics for CCS based on proved transitions. Fundamentae Informaticae 11, 433–452.
[4] Boudol, G., Castellani, I., 1989. Permutation of transitions: an event structure semantics for CCS and SCCS, in: REXSchool/Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, pp. 411–427.
[5] Boudol, G., Castellani, I., Hennessy, M., Kiehn, A., 1994. A theory of processes with localities. Formal Aspects of Computing 6, 165–200. doi:10.1007/BF01221098.
[6] Castellani, I., 1995. Observing distribution in processes: Static and dynamic localities. International Journal of Foundations of Computer Science 6, 353–393.
[7] De Nicola, R., C. B. Hennessy, M., 1984. Testing equivalences for processes. Theor. Comput. Sci. 34, 83–133.
[8] Deniélou, P.M., Yoshida, N., 2012. Multiparty session types meet communicating automata, in: Seidl, H. (Ed.), Programming Languages and Systems, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 194–213.
[9] Doner, J., 1970. Tree acceptors and some of their applications. Journal of Computer and System Sciences 4, 406–451.
[10] Fehnker, A., van Glabbeek, R., Höfner, P., McIver, A., Portmann, M., Tan, W.L., 2012. A process algebra for wireless mesh networks, in: Seidl, H. (Ed.), Programming Languages and Systems, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 295–315.
[11] Fournet, C., Gonthier, G., 1996. The reflexive CHAM and the join-calculus, in: Boehm, H., Jr., G.L.S. (Eds.), Conference Record of POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM Press. pp. 372–385. URL: http://dl.acm.org/citation.cfm?id=237721, doi:10.1145/237721.237805.
[12] Fournet, C., Gonthier, G., 2002. The join calculus: A language for distributed mobile programming, in: Applied Semantics, International Summer School, APPSEM 2000, Caminha, Portugal, September 9-15, 2000, Advanced Lectures, Springer-Verlag, London, UK, UK. pp. 268–332. URL: http://dl.acm.org/citation.cfm?id=647424.725795.
[13] Hennessy, M., 2007. A Distributed Pi-Calculus. Cambridge University Press. doi:10.1017/CBO9780511611063.
[14] Hoare, C.A.R., 1985. Communicating Sequential Processes. Prentice-Hall.
[15] Milner, R., 1989. Communication and Concurrency. International Series on Computer Science, Prentice Hall.
[16] Milner, R., 1999. Communicating and mobile systems: the π-calculus. Cambridge University Press, Cambridge.
[17] Montanari, U., Sammartino, M., 2015. Network-conscious π-calculus – a model of pastry. Electronic Notes in Theoretical Computer Science 312, 3 – 17. doi:https://doi.org/10.1016/j.entcs.2015.04.002.
[18] Peschanski, F., Hym, S., 2006. A stackless runtime environment for a pi-calculus, in: VEE 2006 - Proceedings of the Second International Conference on Virtual Execution Environments, pp. 57–67.

Appendix

Missing proofs of important lemmas are listed here.

Lemma 5.4. By induction on the derivation of $t \xrightarrow{i:f} t'$

- (act) we have $t@\epsilon = f(p_0, p_1)$ which is readily of the desired form.
- (sum-left) by induction hypothesis, we have $s_1@\epsilon \equiv_{\mathsf{p}} f(p_0, p_1)+s$ so $t = (f(p_0, p_1)+s)+s_2$ and $t \equiv_{\mathsf{p}} f(p_0, p_1)+(s+s_2)$, which is in the desired form. The symmetric argument applies for (sum-right).
- (link) we have $t@\epsilon = D(\tilde{g})$. Since Def contains only guarded sums, we have $D(\tilde{g}) \equiv_{\mathsf{p}} f(p_0, p_1)+s$ and one proceeds as before.
- (dist) The interesting case is whenever $p =_{\mathsf{def}} p_0 \parallel p_1$. We have $\mathsf{dist}(p) = p_0 \oplus_{\{(0,1)\}} p_1$ and $\mathsf{dist}(p) \xrightarrow{i:f} t$.
  - Suppose $i = 0.i'$ for some i'. The only rule that has:
  $$p_0 \oplus_{\{(0,1)\}} p_1 \xrightarrow{0.i':f} t$$
  in its conclusion is $(\mathsf{par}-\mathsf{left})$. Therefore we have $p_0 \xrightarrow{i':f} t_0$ (by the premises of the rule) and $t = t_0 \oplus_{\{(0,1)\}} p_1$. By induction hypothesis we have $p_0@i' \equiv_{\mathsf{p}} f(p', q')+s$. One deduces that $\mathsf{dist}(p)@0.i' \equiv_{\mathsf{p}} f(p', q')+s$ or written otherwise $\mathsf{dist}(p)@i \equiv_{\mathsf{p}} f(p', q')+s$, which is the desired form.
  - The case $i = 1.i'$ for some i' is treated likewise using the $(\mathsf{par}-\mathsf{right})$ rule.
  - The case where $i = \epsilon$ is not applicable because no rule having $t \oplus_A t'$ on the left hand side has label with an ϵ address.
- (synch) This rule may not produce a transition of the correct form.
- (par-left,par-right) one applies induction hypothesis on the premises of the rules.

□

Lemma 5.6. We proceed by induction on the derivation of $t \xrightarrow{i:f} t'$.

- Case (act). We have $t = f(p_0, p_1)$ and $\Gamma = (\{x\}, \emptyset, [x \mapsto f(p_0, p_1)])$, and $\phi = [\epsilon \mapsto x]$, by derivation (3). Furthermore, we have $t' = p_0 \oplus_\emptyset p_1$ and for all $b \in \{0, 1\}$, by derivation (3), we have $p_b \rightsquigarrow_{\phi_b} \Gamma_b$ with $\Gamma_b = (\{x_b\}, \emptyset, [x_b \mapsto q_b])$ for some $q_b \equiv_{\mathsf{p}} p_b$, and with $\phi_b = [\epsilon \mapsto x_b]$. *W.l.o.g* we suppose $x_0 \neq x_1$. We apply derivation (5) to deduce:
  $$p_0 \oplus_\emptyset p_1 \rightsquigarrow_\psi (\{x_0, x_1\}, \emptyset, [x_0 \mapsto q_0; x_1 \mapsto q_1]) \equiv_{\mathsf{g}} \Gamma[x_0 : p_0, x_1 : p_1/x]$$
  since $q_b \equiv_{\mathsf{p}} p_b$.
- Cases (sum-left/right). We have $t = s_0+s_1 \rightsquigarrow_\phi \Gamma$ and we apply induction hypothesis to $s_0 \xrightarrow{i:f} t'$ (resp. $s_1 \xrightarrow{i:f} t'$) to obtain $t' \rightsquigarrow_\psi \Gamma[x_0, x_1/\phi(i)]$.

- Case (link). By derivation (3), we have $\Gamma = (\{x\}, \emptyset, [x \mapsto D(\tilde{g})])$ and $\phi = [\epsilon \mapsto x]$. Following the (link) rule, $D(\tilde{g}) \xrightarrow{i:f} t'$ is derivable if $(D(\tilde{g}), s) \in \mathsf{Def}$ (*modulo* proper renaming) and $s \xrightarrow{i:f} t'$ (i). Now because s is a sum, we have $s \rightsquigarrow_{\phi'} \Gamma' =_{\mathsf{def}} (\{x\}, \emptyset, [x \mapsto s'])$ (ii) with $s \equiv_p s'$. We apply induction on (i) and (ii) to deduce $t' \rightsquigarrow_{\psi} \Gamma'[x_0, x_1/x']$ and we can conclude since:

$$\Gamma =_{\mathsf{def}} (\{x\}, \emptyset, [x \mapsto D(\tilde{g})]) \equiv_{\mathsf{g}} (\{x\}, \emptyset, [x \mapsto s']) =_{\mathsf{def}} \Gamma'$$

- Case (dist). The interesting case is when $p = p_0 \parallel p_1$. By derivation (3), we have $\Gamma = (\{x\}, \emptyset, [x \mapsto p_0 \parallel p_1])$, and $\phi = [\epsilon \mapsto x]$. Furthermore we have $\mathsf{dist}(p) =_{\mathsf{def}} p_0 \oplus_{\{(0,1)\}} p_1 \xrightarrow{i:f} t'$ and $i = b.i'$ for some address i' and some $b \in \{0, 1\}$.
  Applying derivation (5) and derivation (3), we have:

$$\mathsf{dist}(p) \rightsquigarrow_{\phi'} (\{x, y\}, \{\{x, y\}\}, [x \mapsto q_0; y \mapsto q_1]) =_{\mathsf{def}} \Gamma'$$

  with $q_b \equiv_{\mathsf{p}} p_b$.

  - Suppose $i = 0.i'$. We apply inductive step to deduce:

$$t' \rightsquigarrow_{\psi} (\{x_0, x_1, y\}, \{\{x_0, y\}, \{x_1, y\}\}, [x_b \mapsto t'@i.b; y \mapsto q_1]) =_{\mathsf{def}} \Gamma''$$

    Since $\Gamma'' = \Gamma'[x_0, x_1/x]$ and $\Gamma' \equiv_{\mathsf{g}} \Gamma$ we may conclude.
  - The case $i = 1.i'$ follows a symmetric argument.

- Cases (par-left/right). Let $t = t_0 \oplus_A t_1 \xrightarrow{0.i':f} t'_0 \oplus_A t_1$. By the premises of (par-left) and derivation (5) we have $t_0 \xrightarrow{i':f} t'_0$ and $t_b \rightsquigarrow_{\phi_b} \Gamma_b$ for all $b \in \{0, 1\}$. We apply induction hypothesis to deduce:

$$t'_0 \rightsquigarrow_{\psi} \Gamma'_0[x_0 : t'_0@i'.0, x_1 : t'_0@i'.1/\phi_0(i')]$$

  with $\Gamma'_0 \equiv_{\mathsf{g}} \Gamma_0$.
  Let $p_b = t'_0@i'.b$ and $\Gamma'_0[x_0 : p_0, x_1 : p_1/\phi_0(i')] = (V'_0, E'_0, \pi'_0)$. We use the fact that $t_1 \rightsquigarrow_{\phi_1} \Gamma_1 =_{\mathsf{def}} (V_1, E_1, \pi_1)$ to obtain, thanks to derivation (5):

$$t'_0 \oplus_A t_1 \rightsquigarrow_{\psi} (V'_0 \uplus V_1, E'_0 \uplus E_1 \uplus E_A, \pi'_0 + \pi_1) =_{\mathsf{def}} \Gamma'$$

  We verify now that $\Gamma' \equiv_{\mathsf{g}} \Gamma[x_0, x_1/\phi(i)]$. Since $t = t_0 \oplus_A t_1$ we know, by derivation (5) that $\Gamma = (V_0 \uplus V_1, E_0 \uplus E_1, \pi_0)$. Since V'_0 is the set of vertices of $\Gamma'_0[x_0 : p_0, x_1 : p_1/\phi_0(i')]$ with $\Gamma'_0 \equiv_{\mathsf{g}} \Gamma_0$ we have:

$$\begin{array}{lcl} V'_0 & = & V_0[x_0, x_1/\phi_0(i')] \\ E'_0 & = & E_0[x_0, x_1/\phi_0(i')] \\ \pi'_0 & = & \pi_0[x_0 : t'_0@i'.0, x_1 : t'_0@i'.1/\phi_0(i')] \end{array}$$

  Since $\phi_0(i') = \phi(0.i') = \phi(i)$ and $t'_0@i'.b = (t'_0 \oplus_A t_1)@0.i'.b = t'@i.b$ we can conclude.

- The case (res) is straightforward.

□

Lemma 5.9. By induction on the derivation of $t \rightsquigarrow_\phi \Gamma$.

- Base case is $t = f(p,q) + s \rightsquigarrow_\phi \Gamma = (\{x\}, \emptyset, [x \mapsto f(p,q) + s])$. By rule (act), $t \xrightarrow{\epsilon:f} p \oplus_\emptyset q$. We apply derivation (3) to construct $p \rightsquigarrow_{\phi_0} (\{x_0\}, \emptyset, [x_0 \mapsto p])$ and $q \rightsquigarrow_{\phi_1} (\{x_1\}, \emptyset, [x_1 \mapsto q])$. By derivation (5) we build $p \oplus_\emptyset q \rightsquigarrow_\phi (\{x_0, x_1\}, \emptyset, [x_0 \mapsto p; x_1 \mapsto q]) =_{\mathsf{def}} \Gamma'$. We can conclude since $\Gamma' = \Gamma[x_0 : p, x_1 : q/x]$.

- In the inductive step one assumes $t = t_0 \oplus_A t_1 \rightsquigarrow_\phi \Gamma$ and for all $b \in \{0,1\}$, $t_b \rightsquigarrow_{\phi_b} \Gamma_b$. Since $V =_{\mathsf{def}} V_0 \uplus V_1$, x is either in Γ_0 or Γ_1. *W.l.o.g.*, suppose $x \in \Gamma_0$. We can derive by induction hypothesis:

$$\frac{t_0 \rightsquigarrow_{\phi_0} \Gamma_0 \quad \pi_0(x) = \pi(x) = f(p,q) + s \quad i = \phi_0^{-1}(x)}{t_0 \xrightarrow{i:f} t_0' \rightsquigarrow_{\psi_0} \Gamma_0[x_0 : p, x_1 : q/x] =_{\mathsf{def}} \Gamma_0'}$$

  By rule (par-left), we derive:

$$\frac{t_0 \xrightarrow{i:f} t_0'}{t = t_0 \oplus_A t_1 \xrightarrow{0.i:f} t_0' \oplus_A t_1 = t'}$$

  Additionally, for derivation (5) we derive:

$$\frac{t_0' \rightsquigarrow_{\psi_0} \Gamma_0' \quad t_1 \rightsquigarrow_{\phi_1} \Gamma_1}{t' \rightsquigarrow_{\psi'} (V_0' \uplus V_1, E_0' \uplus E_1 \uplus E_A, \pi_0' + \pi_1) =_{\mathsf{def}} \Gamma'}$$

  with, for all $i \in |t'|$, $\psi'(0.i) = \psi_0(i)$ and $\psi'(1.i) = \phi_1(i)$.

  It remains to verify that $\Gamma' = \Gamma[x_0 : p, x_1 : q/x]$. By definition of the substitution operation we have:

$$\begin{array}{rcl} V_0' & = & V_0[x_0, x_1/x] \\ E_0' & = & E_0[x_0, x_1/x] \\ \pi_0' & = & \pi_0[x_0 : p, x_1 : q/x] \end{array}$$

  Since $\Gamma = (V_0 \uplus V_1, E_0 \uplus E_1 \uplus E_A, \pi_0 + \pi_1)$ and $x \in V_0$ we do have $\Gamma' = \Gamma[x_0 : p, x_1 : q/x]$.

□