A synchronous pi-calculus

Roberto M. Amadio

▶ **To cite this version:**

Roberto M. Amadio. A synchronous pi-calculus. Information and Computation, 2007, 205 (9), pp.1470-1490. hal-00078319v3

HAL Id: hal-00078319
https://hal.science/hal-00078319v3

Submitted on 9 Feb 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A synchronous π-calculus

Roberto M. Amadio*
Université Paris 7†

9th February 2007

hal-00078319, version 3 - 9 Feb 2007

Abstract

The SL synchronous programming model is a relaxation of the ESTEREL synchronous model where the reaction to the absence of a signal within an instant can only happen at the next instant. In previous work, we have revisited the SL synchronous programming model. In particular, we have discussed an alternative design of the model, introduced a CPS translation to a tail recursive form, and proposed a notion of bisimulation equivalence. In the present work, we extend the tail recursive model with first-order data types obtaining a non-deterministic synchronous model whose complexity is comparable to the one of the π-calculus. We show that our approach to bisimulation equivalence can cope with this extension and in particular that labelled bisimulation can be characterised as a contextual bisimulation.

*Partially supported by ANR-06-SETI-010-02.
†Laboratoire *Preuves, Programmes et Systèmes*, UMR-CNRS 7126.

1 Introduction

Concurrent and/or distributed systems are usually classified according to two main parameters (see, *e.g.*, [19]): the *relative speed* of the processes (or threads, or components, or nodes) and their *interaction mechanism*. With respect to the first parameter one refers to synchronous, asynchronous, partially synchronous,... systems. In particular, in *synchronous systems*, there is a notion of *instant* (or phase, or pulse, or round) and at each instant each process performs some actions and synchronizes with all other processes. One may say that all processes proceed at the same speed and it is in this specific sense that we will refer to *synchrony* in this work.

With respect to the second parameter, one considers shared memory, message passing, signals, broadcast,... Concerning the message passing interaction mechanism, one distinguishes various situations according to whether the communication channel includes a bounded or unbounded and an ordered or unordered buffer. In particular the situation where the buffer has 0 capacity corresponds to a *rendez-vous communication* mechanism which is also called *synchronous communication* in that it forces a synchronisation.

The notion of synchrony (in the sense adopted in this work) is a valuable logical concept that simplifies the design and analysis of systems. One may verify this claim by consulting standard textbooks in concurrent/distributed algorithms such as [20, 34] and comparing the algorithms for basic problems such as leader election, minimum spanning tree, consensus,... in the synchronous and asynchronous case. In [20, 34], the formalisation of the so called *synchronous network model* is quite simple. One assumes a fixed network topology and describes the behaviour of each process essentially as an infinite state Moore machine [18]: at each instant, each process, depending on its current state, emits a message on each outgoing edge, then it receives a messages from each incoming edge, and computes its state for the next instant.

In this paper, we are looking at the synchronous model from the point of view of *process calculi*. This means in particular, that we are looking for a notion of equivalence of synchronous systems with good compositionality properties. The works on SCCS [24] and Meije [5] are an early attempt at providing a process calculus representation of the synchronous model. SCCS and Meije are built over the same action structure: essentially, the free abelian group generated by a set of *particulate* actions. The models differ in the choice of the combinators: SCCS starts with a *synchronous* parallel composition and then adds operators to *desynchronise* processes while Meije starts with an *asynchronous* parallel composition and then adds operators that allow to synchronise processes. As a matter of fact, the SCCS and Meije operators are inter-definable so that the calculi can be regarded as two presentations of the same model.

SCCS/Meije is a simple model with nice mathematical properties but it has failed so far to turn into a model for a realistic synchronous programming language. For this reason, we will not take the SCCS/Meije model as a starting point, but the *synchronous language* SL introduced in [12]. Threads in the SL model interact through *signals* as opposed to channels. A cooperative scheduling (as opposed to pre-emptive, see [28]) is sometimes considered, though this is not quite a compulsory choice and it is not followed here. This

style of synchronous and possibly cooperative programming has been advocated as a more effective approach to the development of applications such as event-driven controllers, data flow architectures, graphical user interfaces, simulations, web services, multi-player games (we refer to [2] for a discussion of the applications and implementation techniques).

The SL model can be regarded as a relaxation of the ESTEREL model [8] where the reaction to the *absence* of a signal within an instant can only happen at the next instant. This design choice avoids some paradoxical situations and simplifies the implementation of the model. Unlike the SCCS/Meije model, the SL model has gradually evolved into a general purpose programming language for concurrent applications and has been embedded in various programming environments such as C, JAVA, SCHEME, and CAML (see [11, 30, 33, 21]). For instance, the Reactive ML language [21] includes a large fragment of the CAML language plus primitives to generate signals and synchronise on them. We should also mention that related ideas have been developed by Saraswat et al. [32] in the area of constraint programming.

The Meije and the ESTEREL/SL models were developed in Sophia-Antipolis in the same research team, but, as of today, there seems to be no strong positive or negative result on the possibility of representing one of the models into the other. Still there are a number of features that plead in favour of the ESTEREL/SL model. First, the shift from channel based to signal based communication allows to preserve (to some extent) the *determinacy* of the computation while allowing for multi-point interaction. Second, pure signals, *i.e.*, signals carrying no values, as opposed to pure channels, allow for a representation of data in binary rather than unary notation. Third, there is a natural generalisation of the calculus to include general data types. Fourth, the length of an instant is *programmable* rather than being given *in extenso* as a finite word of so called *particulate actions*. Fifth, efficient implementations of the model have been developed.

In the early 80's, the development of the SCCS/Meije model relied on the *same* mathematical framework (labelled transition system and bisimulation) that was used for the development of the CCS model. However, the following years have witnessed the development of two quite distinct research directions concerned with asynchronous and synchronous programming, respectively. Nowadays, the π-calculus [26] and its relatives can be regarded as typical abstract models of asynchronous concurrent programming while various languages such as LUSTRE [14], ESTEREL [8], and SL [12] carry the flag of synchronous programming.

We remark that while the π-calculus has inherited many of the techniques developed for CCS, the semantic theory of the SL model remains largely underdeveloped. In recent work [1], we have revisited the SL synchronous programming model. In particular, we have discussed an alternative design of the model, introduced a CPS translation to a tail recursive form, and proposed a novel notion of bisimulation equivalence with good compositionality properties. The original SL language as well as the revised one assume that signals are *pure* in the sense that they carry no value. Then computations are naturally deterministic and bisimulation equivalence collapses with trace equivalence. However, practical programming languages that have been developed on top of the model include data types beyond *pure signals* and this extension makes the computation *non-deterministic* unless significant restrictions are imposed. For instance, in the Reactive ML language we have

already quoted, signals carry values and the emission of two distinct values on the same signal may produce a non-deterministic behaviour.

In the present work, we introduce a minimal extension of the tail recursive model where signals may carry first-order values including signal names. The linguistic complexity of the resulting language is comparable to the one of the π-calculus and we tentatively call it the $S\pi$-calculus (pronounced $s - pi$).[1] Our contribution is to show that the notion of bisimulation equivalence introduced in [1] is sufficiently robust to be lifted from the deterministic language with pure signals to the non-deterministic language with data types and signal name generation. The main role in this story is played by a new notion of labelled bisimulation. We show that this notion has good congruence properties and that it can be characterised via a suitable notion of contextual bisimulation in the sense of [17]. The proof of the characterisation theorem turns out to be considerably more complex than in the pure case having to cope with phenomena such as non-determinism and name extrusion.

While this approach to the semantics of concurrency has already been explored in the framework of asynchronous languages including, *e.g.*, the π-calculus [17, 3, 15], Prasad's calculus of broadcasting systems [29, 16], and the ambient calculus [23], this seems to be the first concrete application of the approach to a *synchronous* language. We expect that the resulting semantic theory for the SL model will have a positive fall-out on the development of various static analyses techniques to guarantee properties such as *determinacy* [21], *reactivity* [4], and *non-interference* [22].

In the following, we assume familiarity with the technical development of the theory of bisimulation for the π-calculus and some acquaintance with the synchronous languages of the ESTEREL family.

2 The $S\pi$-calculus

Programs $P, Q, \ldots$ in the $S\pi$-calculus are defined as follows:

$$\begin{array}{lll} P & ::= & 0 \mid A(\mathbf{e}) \mid \overline{s}e \mid s(x).P, K \mid [s_1 = s_2]P_1, P_2 \mid [u \trianglerighteq p]P_1, P_2 \mid \nu s\ P \mid P_1 \mid P_2 \\ K & ::= & A(\mathbf{r}) \end{array}$$

We use the notation $\mathbf{m}$ for a vector $m_1, \ldots, m_n$, $n \geq 0$. The informal behaviour of programs follows. 0 is the terminated thread. $A(\mathbf{e})$ is a (tail) recursive call with a vector $\mathbf{e}$ of expressions as argument. The identifier A is defined by a unique equation $A(\mathbf{x}) = P$ with the usual condition that the variables free in P are contained in $\{\mathbf{x}\}$. $\overline{s}e$ evaluates the expression e and emits its value on the signal s. A value emitted on a signal *persists* within the instant and it is *reset* at the end of each instant. $s(x).P, K$ is the *present* statement which is the fundamental operator of the SL model. If the values $v_1, \ldots, v_n$ have been emitted on the signal s in the current instant then $s(x).P, K$ evolves non-deterministically into $[v_i/x]P$ for some v_i ($[\_/\_]$ is our notation for substitution). On the

[1] S for *synchronous* as in SCCS [25] and SL [12]. Not to be confused with the so called 'synchronous' π-calculus which would be more correctly described as the π-calculus with *rendez-vous* communication nor with the SPI-calculus where the S suggests a pervasive 'spy' controlling and corrupting all communications.

other hand, if no value is emitted then the continuation K is evaluated at the end of the instant. $[s_1 = s_2]P_1, P_2$ is the usual matching function of the π-calculus that runs P_1 if $s_1 = s_2$ and P_2, otherwise. Here both s_1 and s_2 are free. $[u \trianglerighteq p]P_1, P_2$, matches u against the pattern p. We assume u is either a variable x or a value v and p has the shape $\mathsf{c}(\mathbf{p})$, where c is a constructor and $\mathbf{p}$ a vector of patterns. At run time, u is always a *value* and we run σP_1 if σ is the result of matching u against p, and P_2 otherwise. Note that as usual the variables occurring in the pattern p are bound. $\nu s\ P$ creates a new signal name s and runs P. $(P_1 \mid P_2)$ runs in parallel P_1 and P_2. The continuation K is simply a recursive call whose arguments are either expressions or values associated with signals at the end of the instant in a sense that we explain below.[2]

The definition of program relies on the following syntactic categories:

Sig	$::= s \mid t \mid \cdots$	(signal names)
Var	$::= Sig \mid x \mid y \mid z \mid \cdots$	(variables)
Cnst	$::= * \mid \mathsf{nil} \mid \mathsf{cons} \mid \mathsf{c} \mid \mathsf{d} \mid \cdots$	(constructors)
Val	$::= Sig \mid Cnst(Val, \ldots, Val)$	(values $v, v', \ldots$)
Pat	$::= Var \mid Cnst(Pat, \ldots, Pat)$	(patterns $p, p', \ldots$)
Exp	$::= Pat$	(expressions $e, e', \ldots$)
Rexp	$::= !Sig \mid Var \mid Cnst(Rexp, \ldots, Rexp)$	(exp. with dereferenciation $r, r', \ldots$)

As in the π-calculus, signal names stand both for signal constants as generated by the ν operator and signal variables as in the formal parameter of the present operator. Variables *Var* include signal names as well as variables of other types. Constructors *Cnst* include $*$, nil, and cons. We will also write $[v_1; \ldots; v_n]$ for the list of values $\mathsf{cons}(v_1, \ldots, \mathsf{cons}(v_n, \mathsf{nil}) \ldots)$, $n \geq 0$. Values *Val* are terms built out of constructors and signal names. Patterns *Pat* are terms built out of constructors and variables (including signal names). For the sake of simplicity, expressions *Exp* here happen to be the same as patterns but we could easily add first-order functional symbols defined by recursive equations. Finally, *Rexp* is composed of either expressions or the dereferenced value of a signal at the end of the instant. Intuitively, the latter corresponds to the set of values emitted on the signal during the instant. If P, p are a program and a pattern then we denote with $fn(P), fn(p)$ the set of free signal names occurring in them, respectively. We also use $FV(P), FV(p)$ to denote the set of free variables (including signal names).

2.1 Typing

Types include the basic type 1 inhabited by the constant $*$ and, assuming t is a type, the type $sig(t)$ of signals carrying values of type t, and the type $list(t)$ of lists of values of type t with constructors nil and cons. 1 and $list(t)$ are examples of *inductive types*. More inductive types (booleans, numbers, trees,...) can be added along with more constructors. We

[2]The reader may have noticed that we prefer the term *program* to the term *process*. By this choice, we want to stress that the parallel threads that compose a program are tightly coupled and are executed and observed as a whole.

assume that variables (including signals), constructor symbols, and thread identifiers come with their (first-order) types. For instance, a constructor c may have a type $(t_1, t_2) \to t$ meaning that it waits two arguments of type t_1 and t_2 respectively and returns a value of type t. It is then straightforward to define when a program is well-typed and verify that this property is preserved by the following reduction semantics. We just notice that if a signal name s has type $sig(t)$ then its dereferenced value $!s$ should have type $list(t)$. In the following, we will tacitly assume that we are handling well typed programs, expressions, substitutions,...

2.2 Matching

As already mentioned, the $S\pi$-calculus includes two distinct matching constructions: one operating over signal names works as in the π-calculus and the other operating over values of inductive type actually computes a matching substitution $match(v, p)$ which is defined as follows:[3]

$$match(v,p) = \begin{cases} \sigma & \text{if } dom(\sigma) = FV(p), \quad \sigma(p) = v \\ \uparrow & \text{otherwise} \end{cases}$$

To appreciate the difference, assume $s \neq s'$ and consider $P = [s = s']P_1, P_2$ and $P' = [[s] \trianglerighteq [s']]P_1, P_2$. In the first case, P reduces to P_2 while in the second case, P' reduces to $[s/s']P_1$. Indeed, in the first case s' is a constant while in the second case it is a bound variable.

2.3 Informal reduction semantics

Assume $v_1 \neq v_2$ are two distinct values and consider the following program in $S\pi$:

$$P = \nu\ s_1, s_2\ (\quad \overline{s_1}v_1 \quad | \quad \overline{s_1}v_2 \quad | \quad s_1(x).\ (s_1(y).\ (s_2(z).\ A(x,y)\ \underline{,B(!s_1))}\quad \underline{,0)}\quad \underline{,0}\quad)$$

If we forget about the underlined parts and we regard s_1, s_2 as *channel names* then P could also be viewed as a π-calculus process. In this case, P would reduce to

$$P_1 = \nu s_1, s_2\ (s_2(z).A(\sigma(x), \sigma(y))$$

where σ is a substitution such that $\sigma(x), \sigma(y) \in \{v_1, v_2\}$ and $\sigma(x) \neq \sigma(y)$. In $S\pi$, *signals persist within the instant* and P reduces to

$$P_2 = \nu s_1, s_2\ (\overline{s_1}v_1 \mid \overline{s_1}v_2 \mid (s_2(z).A(\sigma(x), \sigma(y)), B(!s_1)))$$

where $\sigma(x), \sigma(y) \in \{v_1, v_2\}$.

One can easily formalise this behaviour by assuming a standard structural equivalence, by introducing the usual rules for matching and for unfolding recursive definitions (cf. rules $=_1^{sig}$, $=_2^{sig}$, $=_1^{ind}$, $=_2^{ind}$, and *rec* in the following Table 1), and by adding the rule:

$$\overline{s}v \mid s(x).P, K \to \overline{s}v \mid [v/x]P$$

[3]Without loss of expressive power, one could assume that in the second matching instruction the pattern p contains exactly one constructor symbol and that all the variables occurring in it are distinct.

What happens next? In the π-calculus, P_1 is *deadlocked* and no further computation is possible. In the $S\pi$-calculus, the fact that no further computation is possible in P_2 is detected and marks the *end of the current instant*. Then an additional computation represented by the relation $\mapsto$ moves P_2 to the following instant:

$$P_2 \mapsto P_2' = \nu s_1, s_2\ B(\ell)$$

where $\ell \in \{[v_1; v_2], [v_2; v_1]\}$. Thus at the end of the instant, a dereferenced signal such as $!s_1$ becomes a list of (distinct) values emitted on s_1 during the instant and then all signals are reset.

We will further comment on the relationships between the π-calculus and the $S\pi$-calculus in section 2.6 once the formal definitions are in place. In the following section 2.4, Table 1 will formalise the reduction relation (in the special case where the transition is labelled with the action τ) while Table 2 will describe the evaluation relation at the end of the instant.

2.4 Transitions

The behaviour of a program is specified by (i) a *labelled transition system* $\xrightarrow{\alpha}$ describing the possible interactions of the program *during an instant* and (ii) a *transition system* $\mapsto$ determining how a program evolves at the *end of each instant*.

As usual, the behaviour is defined only for programs whose only free variables are signals. The labelled transition system is similar to the one of the polyadic π-calculus modulo a different treatment of emission which we explain below. We define actions α as follows:

$$\alpha ::= \tau \mid sv \mid \nu \mathbf{t}\ \overline{s}v$$

where in the emission action the signal names $\mathbf{t}$ are distinct, occur in v, and differ from s. The functions n (names), fn (free names), and bn (bound names) are defined on actions as usual: $fn(\tau) = \emptyset$, $fn(sv) = \{s\} \cup fn(v)$, $fn(\nu \mathbf{t}\ \overline{s}v) = (\{s\} \cup fn(v)) \backslash \{\mathbf{t}\}$; $bn(\tau) = bn(sv) = \emptyset$, $bn(\nu \mathbf{t}\ \overline{s}v) = \{\mathbf{t}\}$; $n(\alpha) = fn(\alpha) \cup bn(\alpha)$. The related labelled transition system is defined in table 1 where rules apply only to programs whose only free variables are signal names and with standard conventions on the renaming of bound names. As usual, the symmetric rule for (*par*) and (*synch*) are omitted. The rules are those of the polyadic π-calculus but for the following points. (1) In the rule (*out*), the emission is *persistent*. (2) In the rule (*in*), the continuation carries the memory that the environment has emitted $\overline{s}v$. For example, this guarantees, that in the program $s(x).(s(y).P, 0), 0$, if the environment provides a value $\overline{s}v$ for the first input then that value persists and is available for the second input too. (3) The rules $(=_1^{ind})$ and $(=_2^{ind})$ handle the pattern matching. We write $P \xrightarrow{\alpha} \cdot$ for $\exists\, P'\ P \xrightarrow{\alpha} P'$. We will also write $P \stackrel{\tau}{\Rightarrow} P'$ for $P(\xrightarrow{\tau})^* P'$ and $P \stackrel{\alpha}{\Rightarrow} P'$ with $\alpha \neq \tau$ for $P(\stackrel{\tau}{\Rightarrow})\ (\xrightarrow{\alpha})(\stackrel{\tau}{\Rightarrow})P'$.

A program is *suspended*, *i.e.*, it reaches the *end of an instant*, when the labelled transition system cannot produce further (internal) τ transitions.

Definition 1 *We write $P \downarrow$ if $\neg(P \xrightarrow{\tau} \cdot)$ and say that the program P is suspended.*

$$(out)\quad \frac{}{\overline{s}v \xrightarrow{\overline{s}v} \overline{s}v} \qquad (in)\quad \frac{}{s(x).P, K \xrightarrow{sv} [v/x]P \mid \overline{s}v}$$

$$(par)\quad \frac{P_1 \xrightarrow{\alpha} P_1' \quad bn(\alpha) \cap fn(P_2) = \emptyset}{P_1 \mid P_2 \xrightarrow{\alpha} P_1' \mid P_2} \qquad (synch)\quad \frac{P_1 \xrightarrow{\nu \mathbf{t}\ \overline{s}v} P_1' \quad P_2 \xrightarrow{sv} P_2' \quad \{\mathbf{t}\} \cap fn(P_2) = \emptyset}{P_1 \mid P_2 \xrightarrow{\tau} \nu \mathbf{t}\ (P_1' \mid P_2')}$$

$$(\nu)\quad \frac{P \xrightarrow{\alpha} P' \quad t \notin n(\alpha)}{\nu t\ P \xrightarrow{\alpha} \nu t\ P'} \qquad (\nu_{ex})\quad \frac{P \xrightarrow{\nu \mathbf{t}\ \overline{s}v} P' \quad t' \neq s \quad t' \in n(v) \backslash \{\mathbf{t}\}}{\nu t'\ P \xrightarrow{(\nu t', \mathbf{t})\overline{s}v} P'}$$

$$(=_1^{sig})\quad \frac{}{[s = s]P_1, P_2 \xrightarrow{\tau} P_1} \qquad (=_2^{sig})\quad \frac{s_1 \neq s_2}{[s_1 = s_2]P_1, P_2 \xrightarrow{\tau} P_2}$$

$$(=_1^{ind})\quad \frac{match(v, p) = \sigma}{[v \trianglerighteq p]P_1, P_2 \xrightarrow{\tau} \sigma P_1} \qquad (=_2^{ind})\quad \frac{match(v, p) = \uparrow}{[v \trianglerighteq p]P_1, P_2 \xrightarrow{\tau} P_2}$$

$$(rec)\quad \frac{A(\mathbf{x}) = P}{A(\mathbf{v}) \xrightarrow{\tau} [\mathbf{v}/\mathbf{x}]P}$$

Table 1: Labelled transition system during an instant

$$(0)\ \frac{}{0 \overset{\emptyset,V}{\longmapsto} 0} \qquad (out)\ \frac{v \text{ occurs in } V(s)}{\overline{s}v \overset{[\{v\}/s],V}{\longmapsto} 0} \qquad (in)\ \frac{s \notin dom(V)}{s(x).P, K \overset{\emptyset,V}{\longmapsto} V(K)}$$

$$(par)\ \frac{P_i \overset{E_i,V}{\longmapsto} P_i' \quad i = 1,2}{(P_1 \mid P_2) \overset{E_1 \cup E_2,V}{\longmapsto} (P_1' \mid P_2')} \qquad (\nu)\ \frac{P \overset{E,V'}{\longmapsto} P' \quad V'(s) \,\|\!\!-E(s) \quad V[[]/s] = V'[[]/s]}{\nu s\ P \overset{E[\emptyset/s],V}{\longmapsto} \nu s\ P'}$$

$$\frac{P \overset{E,V}{\longmapsto} P' \quad V \,\|\!\!-E}{P \mapsto P'}$$

Table 2: Transition system at the end of the instant

When the program P is suspended, an additional computation is carried on to move to the next instant. This computation is described by the transition system $\mapsto$. First of all, we have to compute the set of values emitted on every signal. To this end, we introduce some notation.

Let E vary over functions from signal names to finite sets of values. Denote with $\emptyset$ the function that associates the empty set with every signal name, with $[M/s]$ the function that associates the set M with the signal name s and the empty set with all the other signal names, and with $\cup$ the union of functions defined pointwise.

We represent a set of values as a list of the values contained in the set. More precisely, we write $v \,\|\!\!-M$ and say that v *represents* M if $M = \{v_1, \ldots, v_n\}$ and $v = [v_{\pi(1)}; \ldots; v_{\pi(n)}]$ for some permutation π over $\{1, \ldots, n\}$. Suppose V is a function from signal names to lists of values. We write $V \,\|\!\!-E$ if $V(s) \,\|\!\!-E(s)$ for every signal name s. We also write $dom(V)$ for $\{s \mid V(s) \neq []\}$. If K is a continuation, *i.e.*, a recursive call $A(\mathbf{r})$, then $V(K)$ is obtained from K by replacing each occurrence $!s$ of a dereferenced signal with the associated value $V(s)$. We denote with $V[\ell/s]$ the function that behaves as V except on s where $V[\ell/s](s) = \ell$.

To define the transition $\mapsto$ at the end of the instant, we rely on an auxiliary judgement $P \overset{E,V}{\longmapsto} P'$. Intuitively, this judgement states that: (1) P is suspended, (2) P emits exactly the values specified by E, and (3) the behaviour of P in the following instant is P' and depends on V. The transition system presented in table 2 formalizes this intuition. For instance, one can show that:

$$\nu s_1\ (s_1(x).0, A(!s_2) \mid \overline{s_2}v_3) \mid (\overline{s_2}v_2 \mid \overline{s_1}v_1) \overset{E,V}{\longmapsto} \nu s_1\ (A(V(s_2)) \mid 0) \mid (0 \mid 0)$$

where $E = [\{v_1\}/s_1, \{v_2, v_3\}/s_2]$ and, *e.g.*, $V = [[v_1]/s_1, [v_3; v_2]/s_2]$.

2.5 Derived operators

We introduce some derived operators and some abbreviations. The calculi with *pure signals* considered in [12, 2, 1] can be recovered by assuming that all signals have type $Sig(1)$. In

this case, we will simply write $\overline{s}$ for $\overline{s}*$ and $s.P, K$ for $s(x).P, K$ where $x \notin FV(P)$. We denote with Ω a *looping process* defined, *e.g.*, by $\Omega = A()$ where $A() = A()$. We abbreviate $s(x).P, 0$ with $s(x).P$. We can derive an *internal choice operator* by defining,

$$P_1 \oplus P_2 = \nu s\ (s(x)[x \trianglerighteq \mathsf{0}]P_1, P_2 \mid \overline{s}\mathsf{0} \mid \overline{s}\mathsf{1})$$

where, *e.g.*, we set $\mathsf{0} = []$ and $\mathsf{1} = [*]$. The pause operation suspends the execution till the end of the instant. It is defined by:

$$\mathsf{pause}.K = \nu s\ s.0, K$$

where: $s \notin fn(K)$. We can also simulate an operator $\mathsf{await}\ s(x).P$ that waits for a value on a signal s for arbitrarily many instants by defining:

$$\mathsf{await}\ s(x).P = s(x).P, A(\mathbf{x})$$

where $\{\mathbf{x}\} = \{s\} \cup (FV(P) \backslash \{x\})$ and $A(\mathbf{x}) = s(x).P, A(\mathbf{x})$.

It is also interesting to program a *generalised matching operator* $[x = \nu\mathbf{s}\ v]_X P$ that given a value x, checks whether x has the shape $\nu\mathbf{s}\ v$ where the freshness of the signal names $\mathbf{s}$ is relative to a finite set X of signal names, *i.e.*, no name in $\mathbf{s}$ belongs to X. If this is the case, we run P and otherwise we do nothing. Assuming, $\{\mathbf{s}\} \subseteq fn(v)$, $fn(v) \backslash \{\mathbf{s}\} \subseteq X$, $\{\mathbf{s}\} \cap X = \emptyset$, and $X = \emptyset$ whenever $\{\mathbf{s}\} = \emptyset$, there are three cases to consider:

1. $v = s$ is a signal name and $\mathbf{s}$ is empty. Then $[x = s]_X P$ is coded as $[x = s]P, 0$.

2. $v = s$ is a signal name and $\mathbf{s} = s$. Then $[x = \nu s\ s]_X P$ is coded as $[x \notin X]P$ where if $X = \{s_1, \ldots, s_n\}$ then $[x \notin X]P$ is coded as $[x = s_1]0, (\cdots, [x = s_n]0, P \cdots)$.

3. $v = \mathsf{c}(p_1, \ldots, p_n)$. Let $\{\mathbf{s}'\} = fn(v) \backslash \{\mathbf{s}\}$ be the set of signal names which are free in $\nu\mathbf{s}\ v$. We associate with the vector of signal names $\mathbf{s}'$ a vector of fresh signal names $\mathbf{s}''$. Let $v'' = [\mathbf{s}''/\mathbf{s}']v$. Then $[x = \nu\mathbf{s}\ v]_X P$ is coded as:

   $$[x \trianglerighteq v''][\mathbf{s}'' = \mathbf{s}'][\{\mathbf{s}\} \cap X = \emptyset][\mathbf{s}\ \text{distinct}]P$$

   where: (1) $[s_1'', \ldots, s_m'' = s_1', \ldots, s_m']Q$ is an abbreviation for $[s_1'' = s_1'] \ldots ([s_m'' = s_m']Q, 0) \ldots, 0$, (2) $[\{\mathbf{s}\} \cap X = \emptyset]$ is expressed by requiring that every signal name in $\{\mathbf{s}\}$ does not belong to X, and (3) $[\mathbf{s}\ \text{distinct}]$ is expressed by requiring that the signal names in $\mathbf{s}$ are pairwise different. For example, to express

   $$[x = \nu s_1, s_2\ \mathsf{c}(s_1, \mathsf{c}(s_3', s_2, s_1), s_3')]_{\{s_3', s_4'\}} P$$

   we write $[x \trianglerighteq \mathsf{c}(s_1, \mathsf{c}(s_3'', s_2, s_1), s_3'')][s_3'' = s_3'][s_1 \notin \{s_3', s_4'\}][s_2 \notin \{s_3', s_4'\}][s_1 \neq s_2]P$. Note that the introduction of the auxiliary signal names $\mathbf{s}''$ is required because in the pattern considered the signal names are interpreted as variables and not as constants. Also, note that the names s_1, s_2, and s_3'' are bound in P.

2.6 Comparison with the π-calculus

In order to make a comparison easier, the syntax of the $S\pi$-calculus is similar to the one of the π-calculus. However there are some important semantic differences to keep in mind.

Deadlock vs. End of instant. What happens when all threads are either terminated or waiting for an event that cannot occur? In the π-calculus, the computation stops. In the $S\pi$-calculus (and more generally, in the SL model), this situation is detected and marks the end of the current instant. Then suspended threads are reinitialised, signals are reset, and the computation moves to the following instant.

Channels vs. Signals. In the π-calculus, a message is consumed by its recipient. In the $S\pi$-calculus, a value emitted along the signal persists within an instant and it is reset at the end of it. We note that in the semantics the only relevant information is whether a given value was emitted or not, *e.g.*, we do not distinguish the situation where the same value is emitted once or twice within an instant.

Data types. The (polyadic) π-calculus has *tuples* as basic data type, while the $S\pi$-calculus has *lists*. The reason for including lists rather than tuples in the *basic* calculus is that at the end of the instant we transform a set of values into a suitable data structure (in our case a list) that represents the set and that can be processed as a whole in the following instant. Note in particular, that the list associated with a signal is empty if and only if no value was emitted on the signal during the instant. This allows to detect the *absence* of a signal at the end of the instant.

Determinism vs. Non-determinism. In the $S\pi$-calculus there are two sources of non-determinism. (1) Several values emitted on the same signal compete to be received during the instant, *e.g.*, $\overline{s}\mathsf{0} \mid \overline{s}\mathsf{1} \mid s(x).P$ may evolve into either $\overline{s}\mathsf{0} \mid \overline{s}\mathsf{1} \mid [\mathsf{0}/x]P$ or $\overline{s}\mathsf{0} \mid \overline{s}\mathsf{1} \mid [\mathsf{1}/x]P$. (2) At the end of the instant, values emitted on a signal are collected in an order that cannot be predicted, *e.g.*, $\nu s', s''\ (\overline{s}s' \mid \overline{s}s'' \mid \mathsf{pause}.A(!s, s', s''))$ may evolve into either $A([s'; s''], s', s'')$ or $A([s''; s'], s', s'')$. Accordingly, one may consider two restrictions to make the computation *deterministic*. (i) If a signal can be read *during* an instant then at most one value can be emitted on that signal during an instant.[4] (ii) If a signal can only be read *at the end* of the instant then the processing of the associated list of values is *independent* of its order.[5]

2.7 Comparison with CBS and the timed π-calculus

In the *calculus of broadcasting systems* (CBS, [29]), threads interact through a *unique* broadcast channel. The execution mechanism *guarantees* that at each step one process sends a message while all the other processes either receive the message or ignore it. There

[4]For instance, the calculus with pure signals satisfies this condition.

[5]In the languages of the ESTEREL family, sometimes one makes the hypothesis that the values collected at the end of the instant are combined by means of an associative and commutative function. While this works in certain cases, it seems hard to conceive such a function when manipulating objects such as pointers. It seems that a general notion of deterministic program should be built upon a suitable notion of program equivalence such as the one we develop here.

is a similarity between the emission of a value on a signal and the broadcast of a value in the sense that in both cases the value can be received an arbitrary number of times. On the other hand, it appears that the CBS model does not offer a *direct* representation of the notion of instant.

Berger's timed π-calculus [7] includes a primitive $\mathsf{timer}^t\ x(y).P, Q$ which means: wait for a message on x for at most t time units and if it does not come then do Q. While there is a syntactic similarity with the present statement of the SL model, we remark that the notion of *time unit* is very different from the notion of *instant* in the SL model. In the SL model, an instant lasts exactly the time needed for every process to accomplish the tasks it has scheduled for the current instant. In the timed model, a time unit lasts exactly one reduction step. As a matter of fact, the notion of 'reduction step' is based on a rather arbitrary definition and it fails to be a *robust* programming concept.

3 Labelled bisimulation and its characterisation

We introduce a new notion of labelled bisimulation, a related notion of contextual bisimulation and state our main result: the two bisimulations coincide.

Definition 2 *We write:*

$$P \Downarrow \quad \text{if } \exists\, P' \ (\ P \stackrel{\tau}{\Rightarrow} P' \text{ and } P' \downarrow\ ) \qquad \text{(weak suspension)}$$
$$P \Downarrow_L \quad \text{if } \exists\, \alpha_1, P_1 \ldots, \alpha_n, P_n \ (\ P \stackrel{\alpha_1}{\rightarrow} P_1 \cdots \stackrel{\alpha_n}{\rightarrow} P_n, \quad n \geq 0, \text{ and } P_n \downarrow\ ) \qquad \text{(L-suspension)}$$

Obviously, $P \downarrow$ implies $P \Downarrow$ which in turn implies $P \Downarrow_L$ and we will see that these implications cannot be reversed. The L-suspension predicate (L for labelled) plays an important role in the definition of labelled bisimulation which is the central concept of this paper.

Definition 3 (labelled bisimulation) *A symmetric relation $\mathcal{R}$ on programs is a labelled bisimulation if whenever $P\ \mathcal{R}\ Q$ the following holds:*

(L1) *If $P \stackrel{\tau}{\rightarrow} P'$ then $\exists\, Q'$ $(Q \stackrel{\tau}{\Rightarrow} Q'$ and $P'\ \mathcal{R}\ Q')$.*

(L2) *If $P \stackrel{\nu \mathbf{t}\ \overline{s}v}{\rightarrow} P'$, $P \Downarrow_L$, $\{\mathbf{t}\} \cap fn(Q) = \emptyset$ then $\exists\, Q'$ $(Q \stackrel{\nu \mathbf{t}\ \overline{s}v}{\Rightarrow} Q'$ and $P'\ \mathcal{R}\ Q')$.*

(L3) *If $P \stackrel{sv}{\rightarrow} P'$ then $\exists\, Q'$ $(\ (\ Q \stackrel{sv}{\Rightarrow} Q'$ and $P'\ \mathcal{R}\ Q')$ or $(\ Q \stackrel{\tau}{\Rightarrow} Q'$ and $P'\ \mathcal{R}\ (Q' \mid \overline{s}v)\ )\ )$.*

(L4) *If $S = \overline{s_1}v_1 \mid \cdots \mid \overline{s_n}v_n$, $n \geq 0$, $P' = (P \mid S) \downarrow$, and $P' \mapsto P''$ then*
$\exists\, Q', Q''$ $(\ (Q \mid S) \stackrel{\tau}{\Rightarrow} Q',\ Q' \downarrow,\ P'\ \mathcal{R}\ Q',\ Q' \mapsto Q'',$ and $P''\ \mathcal{R}\ Q''\ )$.

We denote with $\approx_L$ the largest labelled bisimulation.

In reactive synchronous programming, a program is usually supposed to read 'input' signals at the beginning of each instant and to react delivering 'output' signals at the end of each instant. In particular, a program that does not reach a suspension point cannot produce an observable output signal. For instance, if we run $\overline{s} \mid \Omega$ then the emission on

the signal s should not be observable because the program never suspends. Following this intuition, we comment on the conditions $(L1-4)$.

$(L1)$ This condition is standard in the framework of a *bisimulation* semantics. As in the asynchronous case, it exposes the branching structure of a system to the extent that it distinguishes, *e.g.*, the program $(\overline{s}_1 \oplus \overline{s}_2) \oplus \overline{s}_3$ from the program $\overline{s}_1 \oplus (\overline{s}_2 \oplus \overline{s}_3)$. We will comment on alternative approaches at the end of this section.

$(L2)$ According to the intuition sketched above, the condition $(L2)$ requires that an output of a program P is observable only if $P \Downarrow_L$, *i.e.*, only if P may potentially reach a suspension point (remember that in $S\pi$ an output *persists* within an instant). The reasons for choosing the L-suspension predicate rather than, *e.g.*, the weak suspension predicate will be clarified in section 4 and have to do with the fact that L-suspension has better properties with respect to parallel composition. We also anticipate that in the premise of condition $(L2)$, it is equivalent to require $P \Downarrow_L$ or $P' \Downarrow_L$ (cf. remark 19) and that in the conclusion the property $Q' \Downarrow_L$ can be derived (cf. proposition 11). Last but not least, we should stress that in practice we are interested in programs that *react* at each instant and for this reason, programs that do not satisfy the L-suspension predicate are usually rejected by means of static analyses. In this relevant case, the condition $(L2)$ is the usual output condition of the π-calculus.

$(L3)$ The reception of a signal is not directly observable just as the reception of a message in the π-calculus with asynchronous communication. For instance, there is no reason to distinguish $s.0, 0$ from 0. Techniques for handling this situation have already been developed in the framework of the π-calculus with *asynchronous communication* and amount to modify the input clause as in condition $(L3)$ (see [3]). It is a pleasant surprise that this idea can be transposed to the current context.

$(L4)$ The condition $(L4)$ corresponds to the end of the instant and of course it does not arise in the π-calculus. The end of the instant is an observable event since, as we explained above, it is at the end of the instant that we get the results of the program for the current instant. Let us explain the role of the context $S = \overline{s_1}v_1 \mid \cdots \mid \overline{s_n}v_n$ in this condition. Consider the programs:

$$P = s_1.0, A(!s_2) \qquad Q = s_1.0, A([]) \qquad A(l) = [l \trianglerighteq []]0, \overline{s_3}$$

Then $P \downarrow$, $Q \downarrow$, $P \mapsto A([])$, and $Q \mapsto A([])$. However, if we plug P and Q in the context $[\cdot] \mid \overline{s}_2$ then the resulting programs exhibit different behaviours. In other terms, when comparing two suspended programs we should also consider the effect that emitted values may have on the computation performed at the end of the instant. We stress that the context S must preserve the suspension of the program, therefore the emissions in S are only relevant if they correspond to a signal s which is dereferenced at the end of the instant. In particular, the number of contexts S to be considered in rule $(L4)$ is finite whenever the number of distinct values that can be emitted on dereferenced signals is finite (possibly up to injective renaming).

Admittedly, the definition of labelled bisimulation is technical and following previous work [17, 3, 15], we seek its justification through suitable notions of barbed and contextual bisimulation.

Definition 4 (commitment) *We write* $P \searrow \overline{s}$ *if* $P \stackrel{\nu \mathbf{t}\ \overline{s}v}{\longrightarrow} \cdot$ *and say that* P *commits to emit on* s*.*

Definition 5 (barbed bisimulation) *A symmetric relation* $\mathcal{R}$ *on programs is a barbed bisimulation if whenever* $P\ \mathcal{R}\ Q$ *the following holds:*
$(B1)$ *If* $P \stackrel{\tau}{\longrightarrow} P'$ *then* $\exists\, Q'\ \ Q \stackrel{\tau}{\Rightarrow} Q'$ *and* $P'\ \mathcal{R}\ Q'$*.*
$(B2)$ *If* $P \searrow \overline{s}$ *and* $P \Downarrow_L$ *then* $\exists\, Q'\ \ (Q \stackrel{\tau}{\Rightarrow} Q', Q' \searrow \overline{s},$ *and* $P\ \mathcal{R}\ Q')$*.*
$(B3)$ *If* $P \downarrow$ *and* $P \mapsto P''$ *then* $\exists\, Q', Q''\ \ (Q \stackrel{\tau}{\Rightarrow} Q', Q' \downarrow, P\ \mathcal{R}\ Q', Q' \mapsto Q'',$ *and* $P''\ \mathcal{R}\ Q'')$*.*
We denote with $\approx_B$ *the largest barbed bisimulation.*

We claim that this is a 'natural' definition. Condition $(B1)$ corresponds to the usual treatment of τ moves. Condition $(B2)$ corresponds to the observation of the output commitments in the π-calculus with asynchronous communication modulo the L-suspension predicate whose role has already been discussed in presenting the condition $(L2)$. We will see that the L-suspension predicate $\Downarrow_L$ can be defined just in terms of internal reduction (remark 10). As in condition $(L2)$, the condition $Q' \Downarrow_L$ is a consequence of the definition (cf. proposition 24(2)). Finally, condition $(B3)$ corresponds to the observation of the end of the instant and it is a special case of condition $(L4)$ where the context S is empty.

Definition 6 *A static context* C *is defined as follows:*

$$C ::= [\ ] \mid C \mid P \mid \nu s\ C \tag{1}$$

A reasonable notion of program equivalence should be preserved by the static contexts, *i.e.*, by parallel composition and name generation. We define accordingly a notion of contextual bisimulation (cf. [17, 15]).

Definition 7 (contextual bisimulation) *A symmetric relation* $\mathcal{R}$ *on programs is a contextual bisimulation if it is a barbed bisimulation (conditions* $(B1-3)$*) and moreover whenever* $P\ \mathcal{R}\ Q$ *then*
$(C1)$ $C[P]\ \mathcal{R}\ C[Q]$*, for any static context* C*.*
We denote with $\approx_C$ *the largest contextual barbed bisimulation.*

Our main result shows that labelled and contextual bisimulation collapse. In particular, this implies that labelled bisimulation is preserved by the contexts C. The proof will be developed in the following sections.

Theorem 8 *Let* P, Q *be programs. Then* $P \approx_L Q$ *if and only if* $P \approx_C Q$*.*

We claim that our approach to the semantics of the $S\pi$-calculus is rather natural and mathematically robust, however we *cannot* claim that it is more *canonical* than, say, the *weak, early bisimulation semantics* of the π-calculus. We have chosen to explore a path following our mathematical taste, however, as in the π-calculus, other paths could be explored. In this respect, we will just mention three directions. First, one could remark that condition (B1) in definition 5 allows to observe the branching structure of a program and argue that only suspended programs should be observed. This would lead us towards a failure semantics/testing scenario [13, 9] (in the testing semantics, a program that cannot perform internal reductions is called *stable* and this is similar to a *suspended* program in the synchronous context). Second, one could require that program equivalence is preserved by all contexts and not just the static ones and proceed to adapt, say, the concept of *open bisimulation* [31] to the present language. Third, one could plead for reduction congruence [27] rather than for contextual bisimulation and then try to see whether the two concepts coincide following [15]. We refer to the literature for standard arguments concerning bisimulation vs. testing semantics (*e.g.*, [25]), early vs. open bisimulation (*e.g.*, [31]), and contextual vs. reduction bisimulation (*e.g.*, [15]).

4 Understanding L-suspension

In this section, we study the properties of the L-suspension predicate and justify its use in the definition of labelled bisimulation.

Proposition 9 (characterisations of L-suspension) *Let P be a program. The following are equivalent:*

(1) $P \Downarrow_L$.

(2) *There is a program Q such that* $(P \mid Q) \Downarrow$.

(3) *There is a static context C (cf. definition 6) such that* $C[P] \Downarrow_L$.

Proof. $(1 \Rightarrow 2)$ Suppose $P_0 \stackrel{\alpha_1}{\to} P_1 \cdots \stackrel{\alpha_n}{\to} P_n$ and $P_n \downarrow$. We build Q by induction on n. If $n = 0$ we can take $Q = 0$. Otherwise, suppose $n > 0$. By inductive hypothesis, there is Q_1 such that $(P_1 \mid Q_1) \Downarrow$. We proceed by case analysis on the first action α_1.

$(\alpha_1 = \tau)$ Then we can take $Q = Q_1$ and $(P_0 \mid Q) \stackrel{\tau}{\to} (P_1 \mid Q_1)$.

$(\alpha_1 = sv)$ Let $Q = (Q_1 \mid \overline{s}v)$. We have $(P_0 \mid Q) \stackrel{\tau}{\to} (P_1 \mid Q_1 \mid \overline{s}v)$. Since $P_1 \stackrel{\overline{s}v}{\to} P_1$, we observe that $(P_1 \mid Q_1) \Downarrow$ implies $(P_1 \mid Q_1 \mid \overline{s}v) \Downarrow$.

$(\alpha_1 = \nu \mathbf{t}\ \overline{s}v)$ We distinguish three subcases.

1. If $\alpha_1 = \overline{s}t$ then define $Q = s(t).Q_1$ and observe that $(P_0 \mid Q) \stackrel{\tau}{\to} (P_1 \mid Q_1)$.
2. If $\alpha_1 = \nu t\ \overline{s}t$ then define again $Q = s(t).Q_1$ and observe that (i) $(P_0 \mid Q) \stackrel{\tau}{\to} \nu t\ (P_1 \mid Q_1)$ and (ii) $(P_1 \mid Q_1) \Downarrow$ implies $\nu t\ (P_1 \mid Q_1) \Downarrow$.

3. If $\alpha_1 = \nu\mathbf{t}\ \overline{s}\mathsf{c}(\mathbf{v})$ then let $\{\mathbf{t}'\} = fn(\mathsf{c}(\mathbf{v}))\backslash\{\mathbf{t}\}$ and $\mathbf{t}''$ a tuple of fresh names (one for each name in $\mathbf{t}'$). We define $Q = s(x).[x \trianglerighteq [\mathbf{t}''/\mathbf{t}']\mathsf{c}(\mathbf{v})]Q_1, 0$ where $x, \mathbf{t}'' \notin FV(Q_1)$ and observe that: (i) $(P_0 \mid Q) \stackrel{\tau}{\Rightarrow} \nu\mathbf{t}\ (P_1 \mid Q_1)$ and (ii) $(P_1 \mid Q_1) \Downarrow$ implies $\nu\mathbf{t}\ (P_1 \mid Q_1) \Downarrow$. For instance, if $P_0 \stackrel{\nu t\ \overline{s}\mathsf{c}(t,t')}{\longrightarrow} P_1$ then we take $Q = s(x).[x \trianglerighteq \mathsf{c}(t, t'')]Q_1, 0$ with $x, t'' \notin FV(Q_1)$.

$(2 \Rightarrow 3)$ Take $C = [\ ] \mid Q$ and note that by definition $(P \mid Q) \Downarrow$ implies $(P \mid Q) \Downarrow_L$.

$(3 \Rightarrow 1)$ First, check by induction on a static context C that $P \stackrel{\tau}{\longrightarrow} \cdot$ implies $C[P] \stackrel{\tau}{\longrightarrow} \cdot$. Hence, $C[P] \downarrow$ implies $P \downarrow$. Second, show that $C[P] \stackrel{\alpha}{\longrightarrow} Q$ implies that $Q = C'[P']$ and either $P = P'$ or $P \stackrel{\alpha'}{\longrightarrow} P'$. Third, suppose $C[P] \stackrel{\alpha_1}{\longrightarrow} Q_1 \cdots \stackrel{\alpha_n}{\longrightarrow} Q_n$ with $Q_n \downarrow$. Show by induction on n that $P \Downarrow_L$. □

Remark 10 *The second characterisation, shows that the L-suspension predicate can be defined just in terms of the internal (τ) transitions and the suspension predicate. Thus it does not depend on the choice of observing certain labels.*

Proposition 11 (L-suspension and labelled equivalence) (1) *If $\neg P \Downarrow_L$ and $\neg Q \Downarrow_L$ then $P \approx_L Q$.*

(2) *If $P \approx_L Q$ and $P \Downarrow_L$ then $Q \Downarrow_L$.*

Proof. (1) First we note that $\neg P \Downarrow_L$ and $P \stackrel{\alpha}{\longrightarrow} P'$ implies $\neg P' \Downarrow_L$. Second, we check that $R = \{(P, Q) \mid \neg P \Downarrow_L \text{ and } \neg Q \Downarrow_L\}$ is a labelled bisimulation.

$(L1)$ If $P \stackrel{\tau}{\longrightarrow} P'$ then $\neg P' \Downarrow_L$. Then $Q \stackrel{\tau}{\Rightarrow} Q$ and $P'\ \mathcal{R}\ Q$.

$(L2)$ The condition holds since $\neg P \Downarrow_L$.

$(L3)$ If $P \stackrel{sv}{\longrightarrow} P'$ then $\neg P' \Downarrow_L$. Then $Q \stackrel{\tau}{\Rightarrow} Q$ and by proposition 9, $\neg Q \Downarrow_L$ implies $\neg(Q \mid \overline{s}v) \Downarrow_L$.

$(L4)$ The condition holds since $\neg(P \mid S) \downarrow$. Indeed if $(P \mid S) \downarrow$ then $(P \mid S) \Downarrow_L$ and by proposition 9, $P \Downarrow_L$ which contradicts the hypothesis.

(2) Suppose $P_0 \approx_L Q_0$ and $P_0 \Downarrow_L$. We proceed by induction on the length n of the shortest sequence of transitions to a suspended program: $P_0 \stackrel{\alpha_1}{\longrightarrow} \cdots \stackrel{\alpha_n}{\longrightarrow} P_n$ and $P_n \downarrow$. If $n = 0$ then by $(L4)$, $Q_0 \stackrel{\tau}{\Rightarrow} Q'$ and $Q' \downarrow$. Thus $Q_0 \Downarrow_L$. If $n > 0$ then we analyse the first action α_1.

$(\alpha_1 = \tau)$ By $(L1)$, $Q_0 \stackrel{\tau}{\Rightarrow} Q_1$ and $P_1 \approx_L Q_1$. By inductive hypothesis $Q_1 \Downarrow_L$ and therefore $Q_0 \Downarrow_L$.

$(\alpha_1 = \nu\mathbf{t}\ \overline{s}v)$ By $(L2)$, since $P_0 \Downarrow_L$, we have $Q_0 \stackrel{\nu\mathbf{t}\ \overline{s}v}{\Rightarrow} Q_1$ and $P_1 \approx_L Q_1$. By inductive hypothesis, $Q_1 \Downarrow_L$. Thus $Q_0 \Downarrow_L$.

$(\alpha_1 = sv)$ According to $(L3)$ we have two subcases. If $Q_0 \stackrel{sv}{\Rightarrow} Q_1$ and $P_1 \approx_L Q_1$ then we reason as in the previous case. If $Q_0 \stackrel{\tau}{\Rightarrow} Q_1$ and $P_1 \approx_L (Q_1 \mid \overline{s}v)$ then by inductive hypothesis $(Q_1 \mid \overline{s}v) \Downarrow_L$. By proposition 9, if $(Q_1 \mid \overline{s}v) \Downarrow_L$ then $Q_1 \Downarrow_L$. Thus $Q_0 \Downarrow_L$. □

Thus labelled bisimulation equates all programs which cannot L-suspend and moreover it never equates a program which L-suspends to one which cannot. In this sense, L-suspension is reminiscent of the notion of *solvability* in the λ-calculus [6, p. 41]. In spite of these nice properties, one may wonder whether the L-suspension predicate could be replaced by the suspension or weak suspension predicate.

Definition 12 *We denote with $\approx_L^{\downarrow}$ ($\approx_L^{\Downarrow}$) the notion of labelled bisimulation obtained by replacing in $(L2)$ the condition $P \Downarrow_L$ with the condition $P \downarrow$ ($P \Downarrow$). Similarly, we denote with $\approx_B^{\downarrow}, \approx_C^{\downarrow}$ ($\approx_B^{\Downarrow}, \approx_C^{\Downarrow}$) the notions of barbed and contextual bisimulations obtained by replacing in $(B2)$ the condition $P \Downarrow_L$ with the condition $P \downarrow$ ($P \Downarrow$).*

Proposition 13 (comparing bisimulations) (1) *The following inclusions hold:*

$$\approx_B \subset \approx_B^{\Downarrow} \subset \approx_B^{\downarrow} , \qquad \approx_L \subset \approx_L^{\Downarrow} \subset \approx_L^{\downarrow} , \qquad \approx_C \subseteq \approx_C^{\Downarrow} \subseteq \approx_C^{\downarrow} \ .$$

(2) *The barbed bisimulations and the labelled bisimulations $\approx_L^{\Downarrow}$ and $\approx_L^{\downarrow}$ are not preserved by parallel composition.*

Proof. (1) The non-strict inclusions follow from the remark that $P \downarrow$ implies $P \Downarrow$ which implies $P \Downarrow_L$. We provide examples for the 4 strict inclusions.

- Consider $P = (\overline{s}_1 \mid (\overline{s}_2 \oplus \overline{s}_3))$ and $Q = (\overline{s}_1 \mid \overline{s}_2) \oplus (\overline{s}_1 \mid \overline{s}_3)$. Note that $P, Q \Downarrow$ but $\neg P, Q \downarrow$ and that to reach a suspension point, P and Q have to resolve their internal choices. Now we have $P \approx_L^{\downarrow} Q$ (and therefore $P \approx_B^{\downarrow} Q$) but $P \not\approx_B^{\Downarrow} Q$ (and therefore $P \not\approx_L^{\Downarrow} Q$). To see the latter, observe that $P \searrow \overline{s}_1$ and that to match this commitment Q must choose between $\overline{s}_2$ and $\overline{s}_3$.
- Let (t, t') abbreviate $[t; t']$ and $s \to 0, \Omega$ abbreviate $s(x).[x \trianglerighteq \mathsf{0}]0, \Omega$. Consider:

$$\begin{array}{ll} P_1 & = \nu t, t' \ (\overline{s}(t, t') \mid (t.\overline{s}_1 \oplus t.\overline{s}_2) \mid Q) \\ P_2 & = \nu t, t' \ (((\overline{s}(t, t') \mid (t.\overline{s}_1)) \oplus (\overline{s}(t, t') \mid (t.\overline{s}_2))) \mid Q) \\ Q & = t' \to 0, \Omega \mid \overline{t'}\mathsf{1} \end{array}$$

Note that $P_1, P_2 \Downarrow_L$ but $\neg P_1, P_2 \Downarrow$. The point is that the program Q loops unless the name t' is extruded to the environment and the latter provides a value $\mathsf{0}$ on the signal t'. Then $P_1 \approx_L^{\Downarrow} P_2$. However, $P_1 \not\approx_L P_2$. To see this, notice that $P_1 \searrow \overline{s}$ and that to match this commitment, P_2 has to resolve first the internal choice between $\overline{s}_1$ and $\overline{s}_2$. A variant of this example where we remove the input prefix $t.\_$ before the emissions $\overline{s}_i$, $i = 1, 2$, shows that $\approx_B$ is strictly included in $\approx_B^{\Downarrow}$.

(2) It is well known that barbed bisimulation is not preserved by parallel composition. For instance, $s.\overline{s}_1 \approx_B s.\overline{s}_2$, but $(s.\overline{s}_1 \mid \overline{s}) \not\approx_B (s.\overline{s}_2 \mid \overline{s})$ if $s_1 \neq s_2$. To show that $\approx_L^{\downarrow}$ and $\approx_L^{\Downarrow}$ are not preserved by parallel composition consider again the programs P_1 and P_2 above in parallel with:

$$R = s(t, t').((\overline{t} \mid \overline{t'}\mathsf{0}) \oplus (\overline{t} \mid \overline{t'}\mathsf{0} \mid \overline{s}_3))$$

where $s(t,t').P$ abbreviates $s(x).[x \trianglerighteq [t;t']]P, 0$. Remark that

$$(P_1 \mid R) \stackrel{\tau}{\Rightarrow} \nu t,t' \ (\overline{s}(t,t') \mid (t.\overline{s}_1 \oplus t.\overline{s}_2) \mid Q \mid \overline{t} \mid \overline{t'}0) \equiv P_1'$$

To match this move, suppose $(P_2 \mid R) \stackrel{\tau}{\Rightarrow} P_2'$. Now P_2' must be able to suspend while losing the possibility of committing on $\overline{s}_3$. Hence, there must be a synchronisation on s between P_2 and R. In turn, this synchronisation forces P_2 to choose between $\overline{s}_1$ and $\overline{s}_2$. Suppose, *e.g.*, $(P_2 \mid R)$ chooses $\overline{s}_1$, then in a following move P_1' chooses $\overline{s}_2$ and becomes:

$$\nu t,t' \ (\overline{s}(t,t') \mid \overline{s}_2 \mid 0 \mid \overline{t} \mid \overline{t'}0 \mid \overline{t'}1)$$

which is suspended and commits on $\overline{s}_2$. The program P_2' cannot match this move. □

Note that in (1) the inclusions for the barbed and labelled bisimulations are strict. On the other hand, we do not know whether the inclusions of the contextual bisimulations are strict. However, by (2) we do know that the notions of labelled bisimulation where L-suspension is replaced by (weak) suspension are not preserved by parallel composition and therefore cannot characterise the weaker notions of contextual bisimulation. The conclusion we draw from this analysis is that $\approx_L$ is the good notion of labelled bisimulation among those considered.

5 Strong labelled bisimulation and an up-to technique

It is technically convenient to introduce a *strong* notion of labelled bisimulation which is used to bootstrap the reasoning about the weaker notion we are aiming at.

Definition 14 (strong labelled bisimulation) *A symmetric relation $\mathcal{R}$ on programs is a strong labelled bisimulation if whenever $P \ \mathcal{R} \ Q$ the following holds:*
(S1) $P \stackrel{\alpha}{\rightarrow} P'$ *and* $bn(\alpha) \cap fn(Q) = \emptyset$ *implies* $\exists Q' \ (Q \stackrel{\alpha}{\rightarrow} Q'$ *and* $P' \ \mathcal{R} \ Q')$.
(S2) $(P \mid S) \downarrow$ *with* $S = (\overline{s_1}v_1 \mid \cdots \mid \overline{s_n}v_n)$, $n \geq 0$ *and* $(P \mid S) \mapsto P'$ *implies* $(P \mid S) \ \mathcal{R} \ (Q \mid S)$ *and* $\exists Q' \ (Q \mapsto Q'$ *and* $P' \ \mathcal{R} \ Q')$.
We denote with $\equiv_L$ the largest strong labelled bisimulation.

Proposition 15 *If $P \equiv_L Q$ then $P \approx_L Q$.*

Proof. We check that $\equiv_L$ is a labelled bisimulation. Conditions $(L1-3)$ follow from condition $(S1)$. Condition $(L4)$ follows from condition $(S2)$ noticing that $(P \mid S) \equiv_L (Q \mid S)$ and $(P \mid S) \downarrow$ implies by $(S1)$ that $(Q \mid S) \downarrow$. □

When comparing strong labelled bisimulation with labelled bisimulation it should be noticed that in the former not only we forbid weak internal moves but we also drop the convergence condition in $(L2)$ and the possibility of matching an input with an internal transition in $(L3)$. For this reason, we adopt the notation $\equiv_L$ rather than the usual $\sim_L$.

Definition 16 *We say that a relation $\mathcal{R}$ is a strong labelled bisimulation up to strong labelled bisimulation if the conditions $(S1-2)$ hold when we replace $\mathcal{R}$ with the larger relation $(\equiv_L) \circ \mathcal{R} \circ (\equiv_L)$.*

The following proposition summarizes some useful properties of strong labelled bisimulation. In the present context, an *injective renaming* is an injective function mapping signal names to signal names.

Proposition 17 (properties of $\equiv_L$) (1) *If $P \equiv_L Q$ and σ is an injective renaming then $\sigma P \equiv_L \sigma Q$.*

(2) *$\equiv_L$ is a reflexive and transitive relation.*

(3) *The following laws hold:*

$$(P \mid 0) \equiv_L P, \qquad P_1 \mid (P_2 \mid P_3) \equiv_L (P_1 \mid P_2) \mid P_3, \qquad (P_1 \mid P_2) \equiv_L (P_2 \mid P_1),$$
$$\nu s_1, s_2\ P \equiv_L \nu s_2, s_1\ P \qquad \nu s\ P_1 \mid P_2 \equiv_L \nu s\ (P_1 \mid P_2) \text{ if } s \notin fn(P_2).$$

(4) *If $P \equiv_L Q$ then $(P \mid S) \equiv_L (Q \mid S)$ where $S = (P_1 \mid \cdots \mid P_n)$ and $P_i = 0$ or $P_i = \overline{s_i} v_i$, for $i = 1, \ldots, n$, $n \geq 0$.*

Proof hint. Most properties follow by routine verifications. We just highlight some points.

(2) Recalling that $P \equiv_L Q$ and $P \downarrow$ implies $Q \downarrow$.

(3) Introduce a notion of normalised program where parallel composition associates to the left, all restrictions are carried at top level, and 0 programs are the identity for parallel composition. Then define a relation $\mathcal{R}$ where two programs are related if their normalised forms are identical up to bijective permutations of the restricted names and the parallel components. A pair of programs equated by the laws under consideration is in $\mathcal{R}$. Show that $\mathcal{R}$ is a strong labelled bisimulation.

(4) Show that $\{(P \mid S, Q \mid S) \mid P \equiv_L Q\}$ is a strong labelled bisimulation where S is defined as in the statement. □

The following proposition summarizes the properties of the output transition.

Proposition 18 (emission) (1) *If $P \stackrel{\nu \mathbf{t}\ \overline{s}v}{\longrightarrow} P'$ then $P \equiv_L \nu \mathbf{t}\ (\overline{s}v \mid P'')$ and $P' \equiv_L (\overline{s}v \mid P'')$.*

(2) *If $P \stackrel{\nu \mathbf{t}\ \overline{s}v}{\longrightarrow} P'$ then $P \Downarrow_L$ if and only if $P' \Downarrow_L$.*

Proof. (1) In deriving $P \stackrel{\nu \mathbf{t}\ \overline{s}v}{\longrightarrow} P'$ one can only rely on the rules $(out, par, \nu, \nu_{ex})$. We use the laws of strong labelled bisimulation (proposition 17(2)) to put the program in the desired form.

(2) By definition, $P' \Downarrow_L$ implies $P \Downarrow_L$. In the other direction, relying on (1), assume that the program has the shape $\nu \mathbf{t}\ (\overline{s}v \mid P)$. We also know that this program L-suspends.

By proposition 9, there is a program Q such $\nu\mathbf{t}\ (\overline{s}v \mid P) \mid Q \Downarrow$. That is, assuming $\{\mathbf{t}\} \cap fn(Q) = \emptyset$, we have that $\nu\mathbf{t}\ (\overline{s}v \mid P \mid Q) \Downarrow$. The latter implies that there is a Q' such that $(\overline{s}v \mid P \mid Q) \stackrel{\tau}{\Rightarrow} Q'$ and $Q' \downarrow$. Again, by proposition 9, this means that $(\overline{s}v \mid P) \Downarrow_L$. □

Remark 19 *By proposition 18(2), in condition* $(L2)$ *of definition 3, it is equivalent to require* $P \Downarrow_L$ *or* $P' \Downarrow_L$.

Our main application of strong labelled bisimulation is in the context of a rather standard 'up to technique'.

Definition 20 *A relation* $\mathcal{R}$ *is a labelled bisimulation up to* $\equiv_L$ *if the conditions* $(L1-4)$ *are satisfied when we replace the relation* $\mathcal{R}$ *with the (larger) relation* $(\equiv_L) \circ \mathcal{R} \circ (\equiv_L)$.

Proposition 21 (up-to technique) *Let* $\mathcal{R}$ *be a labelled bisimulation up to* $\equiv_L$. *Then:*
(1) *The relation* $(\equiv_L) \circ \mathcal{R} \circ (\equiv_L)$ *is a labelled bisimulation.*
(2) *If* $P\ \mathcal{R}\ Q$ *then* $P \approx_L Q$.

Proof. (1) A direct diagram chasing using proposition 17.
(2) Follows directly from (1). □

6 Congruence properties of labelled bisimulation

We are now ready to study the congruence properties of labelled bisimulation. The most important part of the proof concerns the preservation under parallel composition and name generation and it is composed of 12 cases.

Proposition 22 (1) *If* $P_1 \approx_L P_2$ *and* σ *is an injective renaming then* $\sigma P_1 \approx_L \sigma P_2$.
(2) *If* $P_1 \approx_L P_2$ *then* $(P_1 \mid \overline{s}v) \approx_L (P_2 \mid \overline{s}v)$.
(3) *The relation* $\approx_L$ *is reflexive and transitive.*
(4) *If* $P_1 \approx_L P_2$ *then* $\nu s\ P_1 \approx_L \nu s\ P_2$ *and* $(P_1 \mid Q) \approx_L (P_2 \mid Q)$.

Proof. **(1)** By propositions 17(1) and 15.

(2) We show that the relation $\mathcal{R} = \approx_L \cup \{(\ P_1 \mid \overline{s}v, P_2 \mid \overline{s}v\ ) \mid P_1 \approx_L P_2\}$ is a labelled bisimulation up to $\equiv_L$. We assume $P_1 \approx_L P_2$ and we analyse the conditions $(L1-4)$.

$(L1)$ Suppose $(P_1 \mid \overline{s}v) \stackrel{\tau}{\rightarrow} (P_1' \mid \overline{s}v)$. If the action τ is performed by P_1 then the hypothesis and condition $(L1)$ allow to conclude. Otherwise, suppose $P_1 \stackrel{sv}{\rightarrow} P_1'$. Then we apply the hypothesis and condition $(L3)$. Two cases may arise: (1) If $P_2 \stackrel{sv}{\Rightarrow} P_2'$ and $P_1' \approx_L P_2'$ then the conclusion is immediate. (2) If $P_2 \stackrel{\tau}{\Rightarrow} P_2'$ and $P_1' \approx_L (P_2' \mid \overline{s}v)$ then we note that $(P_2' \mid \overline{s}v) \equiv_L (P_2' \mid \overline{s}v) \mid \overline{s}v$ and we close the diagram up to $\equiv_L$.

$(L2)$ Suppose $(P_1 \mid \overline{s}v) \Downarrow_L$ and $(P_1 \mid \overline{s}v) \stackrel{\nu \mathbf{t}\ \overline{s'}v}{\rightarrow} (P_1' \mid \overline{s}v)$. If the emission action is performed by $\overline{s}v$ then the conclusion is immediate. Otherwise, note that $P_1 \Downarrow_L$. Hence by $(L2)$, $P_2 \stackrel{\nu \mathbf{t}\ \overline{s'}v}{\Rightarrow} P_2'$ and $P_1' \approx_L P_2'$. But then $(P_2 \mid \overline{s}v) \stackrel{\nu \mathbf{t}\ \overline{s'}v}{\Rightarrow} (P_2' \mid \overline{s}v)$ and we can conclude.

$(L3)$ Suppose $(P_1 \mid \overline{s}v) \stackrel{s'v'}{\rightarrow} (P_1' \mid \overline{s}v)$. Necessarily, $P_1 \stackrel{s'v'}{\rightarrow} P_1'$. By $(L3)$, two cases may arise. If $P_2 \stackrel{s'v'}{\Rightarrow} P_2'$ and $P_1' \approx_L P_2'$ then the conclusion is direct. On the other hand, if $P_2 \stackrel{\tau}{\Rightarrow} P_2'$ and $P_1' \approx_L (P_2' \mid \overline{s'}v')$ then we note that

$$(P_1' \mid \overline{s}v)\ \mathcal{R}\ ((P_2' \mid \overline{s'}v') \mid \overline{s}v) \equiv_L ((P_2' \mid \overline{s}v) \mid \overline{s'}v')$$

and we close the diagram up to $\equiv_L$.

$(L4)$ Let $S = \overline{s_1}v_1 \mid \cdots \mid \overline{s_n}v_n$. Suppose $(P_1 \mid \overline{s}v \mid S) \downarrow$ and $(P_1 \mid \overline{s}v \mid S) \mapsto P_1'$. By $(L4)$ applied to $(\overline{s}v \mid S)$, we derive that $(P_2 \mid \overline{s}v \mid S) \stackrel{\tau}{\Rightarrow} (P_2'' \mid \overline{s}v \mid S)$, $(P_2'' \mid \overline{s}v \mid S) \downarrow$, $(P_1 \mid \overline{s}v \mid S) \approx_L (P_2'' \mid \overline{s}v \mid S)$, $(P_2'' \mid \overline{s}v \mid S) \mapsto P_2'$, and $P_1' \approx_L P_2'$.

(3) It is easily checked that the identity relation is a labelled bisimulation. Reflexivity follows. As for transitivity, we check that the relation $R = \approx_L \circ \approx_L$ is a labelled bisimulation up to $\equiv_L$. Suppose $P_1 \approx_L P_2 \approx_L P_3$.

$(L1)$ Standard argument.

$(L2)$ Suppose $P_1 \Downarrow_L$ and $P_1 \stackrel{\nu \mathbf{t}\ \overline{s}v}{\rightarrow} P_1'$. Note that by (1) we can assume that the names $\mathbf{t}$ are not in P_2. By $(L2)$, $P_2 \stackrel{\nu \mathbf{t}\ \overline{s}v}{\Rightarrow} P_2'$ and $P_1' \approx_L P_2'$. By proposition 18(2), $P_1 \Downarrow_L$ implies $P_1' \Downarrow_L$. By proposition 11(2), $P_1' \Downarrow_L$ and $P_1' \approx_L P_2'$ implies $P_2' \Downarrow_L$. We conclude by applying $(L1)$ and $(L2)$ to P_2 and P_3.

$(L3)$ Suppose $P_1 \stackrel{sv}{\rightarrow} P_1'$. Two interesting cases arise when either P_2 or P_3 match an input action with an internal transition. (1) Suppose first $P_2 \stackrel{\tau}{\Rightarrow} P_2'$ and $P_1 \approx_L (P_2' \mid \overline{s}v)$. By $P_2 \approx_L P_3$ and repeated application of $(L1)$ we derive that $P_3 \stackrel{\tau}{\Rightarrow} P_3'$ and $P_2' \approx_L P_3'$. By property (2), the latter implies that $(P_2' \mid \overline{s}v) \approx_L (P_3' \mid \overline{s}v)$ and we combine with $P_1 \approx_L (P_2' \mid \overline{s}v)$ to conclude. (2) Next suppose $P_2 \stackrel{\tau}{\Rightarrow} P_2^1 \stackrel{sv}{\rightarrow} P_2^2 \stackrel{\tau}{\Rightarrow} P_2'$ and $P_1 \approx_L P_2'$. Suppose that P_3 matches these transitions as follows: $P_3 \stackrel{\tau}{\Rightarrow} P_3^1 \stackrel{\tau}{\Rightarrow} P_3^2$, $P_2^2 \approx_L (P_3^2 \mid \overline{s}v)$, and moreover $(P_3^2 \mid \overline{s}v) \stackrel{\tau}{\Rightarrow} (P_3' \mid \overline{s}v)$ with $P_2' \approx_L (P_3' \mid \overline{s}v)$. Two subcases may arise: (i) $P_3^2 \stackrel{\tau}{\Rightarrow} P_3'$. Then we have $P_3 \stackrel{\tau}{\Rightarrow} P_3'$, $P_2' \approx_L (P_3' \mid \overline{s}v)$ and we can conclude. (ii) $P_3^2 \stackrel{sv}{\Rightarrow} P_3'$. Then we have $P_3 \stackrel{sv}{\Rightarrow} P_3'$ and $P_2' \approx_L (P_3' \mid \overline{s}v) \equiv_L P_3'$. Note that P_3^2 does not need to perform the action sv more than once.

$(L4)$ Let $S = \overline{s_1}v_1 \mid \cdots \mid \overline{s_n}v_n$. Suppose $(P_1 \mid S) \downarrow$ and $(P_1 \mid S) \mapsto P_1'$. By $(L4)$, $(P_2 \mid S) \stackrel{\tau}{\Rightarrow} (P_2'' \mid S)$, $(P_2'' \mid S) \downarrow$, $(P_1 \mid S) \approx_L (P_2'' \mid S)$, $(P_2'' \mid S) \mapsto P_2'$, and $P_1' \approx_L P_2'$. By $(L1)$, $(P_3 \mid S) \stackrel{\tau}{\Rightarrow} (P_3'' \mid S)$ and $(P_2'' \mid S) \approx_L (P_3'' \mid S)$. By $(L4)$, $(P_3'' \mid S) \stackrel{\tau}{\Rightarrow} (P_3''' \mid S)$, $(P_3''' \mid S) \downarrow$, $(P_2'' \mid S) \approx_L (P_3''' \mid S)$, $(P_3''' \mid S) \mapsto P_3'$, $P_2' \approx_L P_3'$ and we can conclude.

(4) We show that $\mathcal{R} = \{(\nu \mathbf{t}\ (P_1 \mid Q), \nu \mathbf{t}\ (P_2 \mid Q)) \mid P_1 \approx_L P_2\} \cup \approx_L$ is a labelled bisimulation up to $\equiv_L$.

$(L1)$ Suppose $\nu \mathbf{t}\ (P_1 \mid Q) \stackrel{\tau}{\rightarrow} \cdot$. This may happen because either P_1 or Q perform a τ action or because P_1 and Q synchronise. We consider the various situations that may occur.

$(L1)[1]$ Suppose $Q \xrightarrow{\tau} Q'$. Then $\nu\mathbf{t}\ (P_2 \mid Q) \xrightarrow{\tau} \nu\mathbf{t}\ (P_2 \mid Q')$ and we can conclude.

$(L1)[2]$ Suppose $P_1 \xrightarrow{\tau} P_1'$. By $(L2)$ $P_2 \stackrel{\tau}{\Rightarrow} P_2'$ and $P_1' \approx_L P_2'$. Then $\nu\mathbf{t}\ (P_2 \mid Q) \stackrel{\tau}{\Rightarrow} \nu\mathbf{t}\ (P_2' \mid Q)$ and we can conclude.

$(L1)[3]$ Suppose $P_1 \xrightarrow{sv} P_1'$ and $Q \xrightarrow{\nu\mathbf{t}'\ \overline{s}v} Q'$. According to $(L3)$, we have two subcases.

$(L1)[3.1]$ Suppose $P_2 \stackrel{sv}{\Rightarrow} P_2'$ and $P_1' \approx_L P_2'$. Then $\nu\mathbf{t}\ (P_2 \mid Q) \stackrel{\tau}{\Rightarrow} \nu\mathbf{t}, \mathbf{t}'\ (P_2' \mid Q')$ and we can conclude.

$(L1)[3.2]$ Suppose $P_2 \stackrel{\tau}{\Rightarrow} P_2'$ and $P_1' \approx_L (P_2' \mid \overline{s}v)$. By proposition 18(2), $Q \equiv_L \nu\mathbf{t}'\ Q'$ and $Q' \equiv_L (Q'' \mid \overline{s}v)$ for some Q''. Then $\nu\mathbf{t}\ (P_2 \mid Q) \stackrel{\tau}{\Rightarrow} \nu\mathbf{t}\ (P_2' \mid Q) \equiv_L \nu\mathbf{t}, \mathbf{t}'\ (P_2' \mid \overline{s}v) \mid Q''$ and we can conclude up to $\equiv_L$.

$(L1)[4]$ Suppose $P_1 \xrightarrow{\nu\mathbf{t}'\ \overline{s}v} P_1'$ and $Q \xrightarrow{sv} Q'$. We have two subcases.

$(L1)[4.1]$ Suppose $\neg P_1 \Downarrow_L$. By propositions 9 and 11, $\neg\nu\mathbf{t}\ (P_1 \mid Q) \Downarrow_L$, $\neg P_2 \Downarrow_L$, $\neg\nu\mathbf{t}\ (P_2 \mid Q) \Downarrow_L$, $\neg P_1' \Downarrow_L$, and $\neg\nu\mathbf{t}, \mathbf{t}'\ (P_1' \mid Q') \Downarrow_L$. Hence, $\nu\mathbf{t}, \mathbf{t}'\ (P_1' \mid Q') \approx_L \nu\mathbf{t}\ (P_2 \mid Q)$ and we can conclude.

$(L1)[4.2]$ Suppose $P_1 \Downarrow_L$. By $(L2)$, $P_2 \stackrel{\nu\mathbf{t}'\ \overline{s}v}{\Rightarrow} P_2'$ and $P_1' \approx_L P_2'$. Hence $\nu\mathbf{t}\ (P_2 \mid Q) \stackrel{\tau}{\Rightarrow} \nu\mathbf{t}, \mathbf{t}'\ (P_2' \mid Q')$ and we can conclude.

$(L2)$ Suppose $\nu\mathbf{t}\ (P_1 \mid Q) \xrightarrow{\nu\mathbf{t}'\ \overline{s}v} \cdot$ and $\nu\mathbf{t}\ (P_1 \mid Q) \Downarrow_L$. Also assume $\mathbf{t} = \mathbf{t_1}, \mathbf{t_2}$ and $\mathbf{t}' = \mathbf{t_1}, \mathbf{t_3}$ up to reordering so that the emission extrudes exactly the names $\mathbf{t}_1$ among the names in $\mathbf{t}$. We have two subcases depending which component performs the action.

$(L2)[1]$ Suppose $Q \xrightarrow{\nu\mathbf{t_3}\ \overline{s}v} Q'$. Then $\nu\mathbf{t}\ (P_2 \mid Q) \xrightarrow{\nu\mathbf{t}'\ \overline{s}v} \nu\mathbf{t_2}\ (P_2 \mid Q')$ and we can conclude.

$(L2)[2]$ Suppose $P_1 \xrightarrow{\nu\mathbf{t_3}\ \overline{s}v} P_1'$. By proposition 9, we know that $P_1 \Downarrow_L$. Hence $P_2 \stackrel{\nu\mathbf{t_3}\ \overline{s}v}{\Rightarrow} P_2'$ and $P_1' \approx_L P_2'$. Then $\nu\mathbf{t}\ (P_2 \mid Q) \xrightarrow{\nu\mathbf{t}'\ \overline{s}v} \nu\mathbf{t_2}\ (P_2' \mid Q)$ and we can conclude.

$(L3)$ Suppose $\nu\mathbf{t}\ (P_1 \mid Q) \xrightarrow{sv} \cdot$ We have two subcases depending which component performs the action.

$(L3)[1]$ Suppose $Q \xrightarrow{sv} Q'$. Then $\nu\mathbf{t}\ (P_2 \mid Q) \xrightarrow{sv} \nu\mathbf{t}\ (P_2 \mid Q')$ and we can conclude.

$(L3)[2]$ Suppose $P_1 \xrightarrow{sv} P_1'$. According to $(L3)$ we have two subcases.

$(L3)[2.1]$ Suppose $P_2 \stackrel{sv}{\Rightarrow} P_2'$ and $P_1' \approx_L P_2'$. Then $\nu\mathbf{t}\ (P_2 \mid Q) \stackrel{sv}{\Rightarrow} \nu\mathbf{t}\ (P_2' \mid Q)$ and we can conclude.

$(L3)[2.2]$ Suppose $P_2 \stackrel{\tau}{\Rightarrow} P_2'$ and $P_1' \approx_L (P_2' \mid \overline{s}v)$. Then $\nu\mathbf{t}\ (P_2 \mid Q) \stackrel{\tau}{\Rightarrow} \nu\mathbf{t}\ (P_2' \mid Q)$ and since $\nu\mathbf{t}\ (P_2' \mid Q) \mid \overline{s}v \equiv_L \nu\mathbf{t}\ ((P_2' \mid \overline{s}v) \mid Q)$ we can conclude up to $\equiv_L$.

$(L4)$ Suppose $S = \overline{s_1}v_1 \mid \cdots \mid \overline{s_n}v_n$ and $\nu\mathbf{t}\ (P_1 \mid Q) \mid S \downarrow$. Up to strong labelled bisimulation, we can express Q as $\nu\mathbf{t}_Q\ (S_Q \mid I_Q)$ where S_Q is the parallel composition of emissions and I_Q is the parallel composition of receptions. Thus we have: $\nu\mathbf{t}\ (P_1 \mid Q) \mid S \equiv_L \nu\mathbf{t}, \mathbf{t}_Q\ (P_1 \mid S_Q \mid I_Q \mid S)$, and $\nu\mathbf{t}\ (P_2 \mid Q) \mid S \equiv_L \nu\mathbf{t}, \mathbf{t}_Q\ (P_2 \mid S_Q \mid I_Q \mid S)$ assuming $\{\mathbf{t}\} \cap fn(S) = \emptyset$ and $\{\mathbf{t}_Q\} \cap fn(P_i \mid S) = \emptyset$ for $i = 1, 2$.

If $\nu\mathbf{t}\ (P_1 \mid Q) \mid S \mapsto P$ then $P \equiv_L \nu\mathbf{t}, \mathbf{t}_Q\ (P_1'' \mid Q')$ where in particular, we have that $(P_1 \mid S_Q \mid S) \downarrow$ and $(P_1 \mid S_Q \mid S) \mapsto (P_1' \mid 0 \mid 0)$.

By the hypothesis $P_1 \approx_L P_2$ and $(L4)$ we derive that: (i) $(P_2 \mid S_Q \mid S) \stackrel{\tau}{\Rightarrow} (P_2'' \mid S_Q \mid S)$, (ii) $(P_2'' \mid S_Q \mid S)\downarrow$, (iii) $(P_2'' \mid S_Q \mid S) \mapsto (P_2' \mid 0 \mid 0)$, (iv) $(P_1 \mid S_Q \mid S) \approx_L (P_2'' \mid S_Q \mid S)$, and (v) $(P_1' \mid 0 \mid 0) \approx_L (P_2' \mid 0 \mid 0)$.

Because $(P_1 \mid S_Q \mid S)$ and $(P_2'' \mid S_Q \mid S)$ are suspended and labelled bisimilar, the two programs must commit (cf. definition 4) on the same signal names and moreover on each signal name they must emit the same set of values up to renaming of bound names. It follows that the program $\nu \mathbf{t}, \mathbf{t}_Q\ (P_2'' \mid S_Q \mid I_Q \mid S)$ is suspended. The only possibility for an internal transition is that an emission in P_2'' enables a reception in I_Q but this contradicts the hypothesis that $\nu \mathbf{t}, \mathbf{t}_Q\ (P_1 \mid S_Q \mid I_Q \mid S)$ is suspended. Moreover, $(P_2'' \mid S_Q \mid I_Q \mid S) \mapsto (P_2' \mid 0 \mid Q' \mid 0)$.

Therefore, we have that

$$\nu \mathbf{t}\ (P_2 \mid Q) \mid S \equiv_L \nu \mathbf{t}, \mathbf{t}_Q\ (P_2 \mid S_Q \mid I_Q \mid S) \stackrel{\tau}{\Rightarrow} \nu \mathbf{t}, \mathbf{t}_Q\ (P_2'' \mid S_Q \mid I_Q \mid S),$$

$\nu \mathbf{t}, \mathbf{t}_Q\ (P_2'' \mid S_Q \mid I_Q \mid S)\downarrow$, and $\nu \mathbf{t}, \mathbf{t}_Q\ (P_2'' \mid S_Q \mid I_Q \mid S) \mapsto \nu \mathbf{t}, \mathbf{t}_Q\ (P_2' \mid 0 \mid Q' \mid 0)$. Now $\nu \mathbf{t}, \mathbf{t}_Q\ (P_1 \mid S_Q \mid I_Q \mid S)\ \mathcal{R}\ \nu \mathbf{t}, \mathbf{t}_Q\ (P_2'' \mid S_Q \mid I_Q \mid S)$ because $(P_1 \mid S_Q \mid S) \approx_L (P_2'' \mid S_Q \mid S)$ and $\nu \mathbf{t}, \mathbf{t}_Q\ (P_1' \mid Q')\ \mathcal{R}\ \nu \mathbf{t}, \mathbf{t}_Q\ (P_2' \mid Q')$ because $P_1' \approx_L P_2'$. □

We can now derive the first half of the proof of theorem 8.

Corollary 23 *Let P, Q be programs. Then $P \approx_L Q$ implies $P \approx_C Q$.*

Proof. Labelled bisimulation is a barbed bisimulation and by proposition 22 it is preserved by the contexts C. Hence it is a contextual bisimulation. □

7 Building discriminating contexts

To complete the proof of theorem 8, it remains to show that our contexts are sufficiently strong to make all distinctions labelled bisimulation does. First we note the analogous of proposition 11 for contextual bisimulation.

Proposition 24 (1) *If $\neg P \Downarrow_L$ and $\neg Q \Downarrow_L$ then $P \approx_C Q$.*
(2) *If $P \approx_C Q$ and $P \Downarrow_L$ then $Q \Downarrow_L$.*

Proof. (1) By proposition 11, $P \approx_L Q$ and by corollary 23, $P \approx_C Q$.

(2) By proposition 9, there is a program R such that $(P \mid R) \Downarrow$, *i.e.*, $(P \mid R) \stackrel{\tau}{\Rightarrow} P_1$ and $P_1 \downarrow$. By $(C1)$, $(P \mid R) \approx_C (Q \mid R)$. By $(B1)$, $(Q \mid R) \stackrel{\tau}{\Rightarrow} Q_1'$ and $P_1 \approx_C Q_1'$. By $(B3)$, $Q_1' \stackrel{\tau}{\Rightarrow} Q_1$ and $Q_1 \downarrow$. Thus $(Q \mid R) \Downarrow$ and again by proposition 9 this implies that $Q \Downarrow_L$. □

Proposition 25 *If $P \approx_C Q$ then $P \approx_L Q$.*

PROOF. We denote with $a_i, b_i, c_i, \ldots$ 'fresh' signal names not occurring in the programs under consideration. We will rely on the signal names a_i to extrude the scope of some signal names and on the signal names b_i, c_i to monitor the internal transitions of the programs. We define a relation $\mathcal{R}$:

$$\begin{array}{ll} P_1 \mathcal{R} P_2 & \text{if } \nu\mathbf{t}\ (P_1 \mid O) \approx_C \nu\mathbf{t}\ (P_2 \mid O) \text{ for some } \mathbf{t}, O, \\ & \text{where: } \mathbf{t} = t_1 \ldots, t_n, O = \overline{a_1}t_1 \mid \cdots \mid \overline{a_n}t_n, \{a_1, \ldots, a_n\} \cap fn(P_1 \mid P_2) = \emptyset. \end{array}$$

By definition, if $P_1 \approx_C P_2$ then $P_1 \mathcal{R} P_2$ taking $\mathbf{t}$ as the empty vector and O as the empty parallel composition. The purpose of the relation $\mathcal{R}$ is to enlarge the definition of contextual bisimulation so that some signal names $\mathbf{t}$ are at once restricted and observable thanks to the emission performed by O. We will will show that $\mathcal{R}$ is a labelled bisimulation up to strong labelled bisimulation so that we have the following implications:

$$P_1 \approx_C P_2 \quad \Rightarrow \quad P_1 \mathcal{R} P_2 \quad \Rightarrow \quad P_1 \approx_L P_2\ .$$

• We have seen in section 2.5 that an internal choice operator $\oplus$ is definable in the $S\pi$-calculus. In order to simplify the notation, in the following we assume that $P_1 \oplus P_2$ reduces to either P_1 or P_2 by just *one* τ-transition. In reality, the reduction takes one τ-transition to perform the internal choice, a second deterministic τ-transition to select the right branch of the matching operator, and some garbage collection to remove signals that are under the scope of a restriction and cannot be received. The second transition and the garbage collection do not affect the structure of the proof and we will ignore them.

• Assuming $O = \overline{a_1}t_1 \mid \cdots \mid \overline{a_n}t_n$ and $\mathbf{a} = a_1, \ldots, a_n$, we will repeatedly use a program $R(\mathbf{a})[P]$ which is defined as follows:

$$\begin{array}{rcl} R(\mathbf{a})[P] & = & a_1(t_1).\overline{b_1} \oplus (\overline{c_1} \oplus \\ & & \quad a_2(t_2).\overline{b_2} \oplus (\overline{c_2} \oplus \\ & & \quad\quad \cdots \\ & & \quad\quad a_n(t_n).\overline{b_n} \oplus (\overline{c_n} \oplus P) \ldots) \end{array}$$

Next we assume $P_1 \mathcal{R} P_2$ because $\nu\mathbf{t}\ (P_1 \mid O) \approx_C \nu\mathbf{t}\ (P_2 \mid O)$ for some $\mathbf{t}, O$, and consider the conditions $(L1-4)$.

$(L1)$ Suppose $P_1 \xrightarrow{\tau} P_1'$. Then $\nu\mathbf{t}\ (P_1 \mid O) \xrightarrow{\tau} \nu\mathbf{t}\ (P_1' \mid O)$. By $(B1)$, $\nu\mathbf{t}\ (P_2 \mid O) \stackrel{\tau}{\Rightarrow} Q$ and $\nu\mathbf{t}\ (P_1' \mid O) \approx_C Q$. Note however that O cannot interact with P_2 and its derivatives because the signal names $\mathbf{a}$ do not occur in $(P_1 \mid P_2)$. Hence it must be that $P_2 \stackrel{\tau}{\Rightarrow} P_2'$ and $Q = \nu\mathbf{t}\ (P_2' \mid O)$. Then by definition of the relation $\mathcal{R}$, we derive that $P_1' \mathcal{R} P_2'$.

$(L2)$ Suppose $P_1 \Downarrow_L$ and $P_1 \stackrel{\nu\mathbf{t}'\ \overline{s}v}{\longrightarrow} P_1'$ with $\mathbf{t}' = t_1', \ldots, t_m'$. Let $X = fn(P_1 \mid P_2)$. Let

$$\begin{array}{ll} R & = R(\mathbf{a})[s(x).[x = \nu\mathbf{t}'\ v]_{X \cup \{\mathbf{t}'\}}\ (\overline{b_{n+1}} \oplus (\overline{c_{n+1}} \oplus O'))], \text{ where} \\ O' & = a_{n+1}t_1' \mid \cdots \mid a_{n+m}t_m' \end{array}$$

Now we have:

$$\nu\mathbf{t}\ (P_1 \mid O) \mid R \stackrel{\tau}{\Rightarrow} \nu\mathbf{t}, \mathbf{t}'\ (P_1' \mid O \mid O')$$

by a series of reductions where first R interacts with O to learn the names $t_1 \ldots, t_n$, then it interacts with P_1 to read a value $\nu \mathbf{t}'\ v$ (note that the freshness of $\mathbf{t}'$ is checked with respect to both X and $\mathbf{t}$), and finally it emits with O' the names $\mathbf{t}'$ extruded by P_1. We remark that in all the intermediate steps the program has the L-suspension property, thus condition $(B2)$ applies and in particular the commitments on $\overline{b}_i, \overline{c}_i$ are observable.

Next, we decompose this series of reductions in several steps and analyse how the program $\nu \mathbf{t}\ (P_2 \mid O) \mid R$ may match them according to the definition of contextual bisimulation. Suppose first

$$\nu \mathbf{t}\ (P_1 \mid O) \mid R \stackrel{\tau}{\Rightarrow} \nu t_1\ (\nu t_2, \ldots, t_n\ (P_1 \mid O) \mid (\overline{c_1} \oplus a_2(t_2) \cdots))$$

The reduced program cannot commit on $\overline{b}_1$ while it can commit on $\overline{c}_1$. If $\nu \mathbf{t}\ (P_2 \mid O) \mid R$ has to match this reduction, then R must necessarily perform the input action and stop at the same point of the control $(\overline{c_1} \oplus a_2(t_2) \cdots)$. By this communication, the scope of the restricted name t_1 is extruded to R. The program O is composed only of emissions and therefore it cannot change. The program P_2 may perform some internal actions but it cannot interact with O and R.

If we repeat this argument n times, we conclude that $\nu \mathbf{t}\ (P_1 \mid O) \mid R \stackrel{\tau}{\Rightarrow} \nu \mathbf{t}\ (P_1 \mid O \mid \overline{c_n} \oplus s(x) \cdots)$ and $\nu \mathbf{t}\ (P_2 \mid O) \mid R \stackrel{\tau}{\Rightarrow} \nu \mathbf{t}\ (P_2' \mid O \mid \overline{c_n} \oplus s(x) \cdots)$ where $P_2 \stackrel{\tau}{\Rightarrow} P_2'$. Now the first program performs a communication on s between P_1 and the residual of R and, provided the emitted value has the expected shape $\nu \mathbf{t}'\ v$, it reduces to $\nu \mathbf{t}, \mathbf{t}'\ (P_1' \mid O \mid \overline{c_{n+1}} \oplus O')$. In order to match this transition, it must be that $P_2' \stackrel{\nu \mathbf{t}'\ \overline{s}v}{\Rightarrow} P_2''$ and the second program reduces to $\nu \mathbf{t}, \mathbf{t}'\ (P_2'' \mid O \mid \overline{c_{n+1}} \oplus O')$. Now if the first program moves to $\nu \mathbf{t}, \mathbf{t}'\ (P_1' \mid O \mid O')$, the second must move to $\nu \mathbf{t}, \mathbf{t}'\ (P_2''' \mid O \mid O')$ where $P'' \stackrel{\tau}{\Rightarrow} P_2'''$ and $\nu \mathbf{t}, \mathbf{t}'\ (P_1' \mid O \mid O') \approx_C \nu \mathbf{t}, \mathbf{t}'\ (P_2''' \mid O \mid O')$. Since $P_2 \stackrel{\tau}{\Rightarrow} \cdot \stackrel{\nu \mathbf{t}'\ \overline{s}v}{\Rightarrow} \cdot \stackrel{\tau}{\Rightarrow} P_2'''$, we can conclude that $P_2 \stackrel{\nu \mathbf{t}'\ \overline{s}v}{\Rightarrow} P_2'''$ and $P_1' \mathcal{R} P_2'''$.

$(L3)$ Suppose $P_1 \stackrel{sv}{\rightarrow} P_1'$. We consider two subcases.

$(L3)[1]$ Suppose $\neg P_1 \Downarrow_L$. Then, $\neg P_1' \Downarrow_L$. By proposition 9, $\neg \nu \mathbf{t}\ (P_1 \mid O) \Downarrow_L$ and $\neg \nu \mathbf{t}\ (P_1' \mid O) \Downarrow_L$. By proposition 24, $\neg \nu \mathbf{t}\ (P_2 \mid O) \Downarrow_L$. Let us show that the latter implies $\neg P_2 \Downarrow_L$. If $P_2 \Downarrow_L$, by proposition 9 there is a Q such that $(P_2 \mid Q) \stackrel{\tau}{\Rightarrow} Q'$ and $Q' \downarrow$. Then we would have:

$$\nu \mathbf{t}\ (P_2 \mid O) \mid R(\mathbf{a})[Q] \stackrel{\tau}{\Rightarrow} \nu \mathbf{t}\ (P_2 \mid O \mid Q) \stackrel{\tau}{\Rightarrow} \nu \mathbf{t}\ Q' \mid O\ .$$

Now if $Q' \downarrow$ then $\nu \mathbf{t}\ Q' \mid O \downarrow$, and this contradicts the hypothesis that $\neg \nu \mathbf{t}\ (P_2 \mid O) \Downarrow_L$. Thus $P_2 \stackrel{\tau}{\Rightarrow} P_2$, $\neg (P_2 \mid \overline{s}v) \Downarrow_L$, and $P_1' \approx_L (P_2 \mid \overline{s}v)$.

$(L3)[2]$ Suppose $P_1 \Downarrow_L$. In this case, the commitments are observable. We define

$$R = R(\mathbf{a})[\overline{s}v]$$

Then $\nu \mathbf{t}\ (P_1 \mid O) \mid R \stackrel{\tau}{\Rightarrow} \nu \mathbf{t}\ (P_1' \mid O \mid \overline{s}v)$ and $\nu \mathbf{t}\ (P_2 \mid O) \mid R \stackrel{\tau}{\Rightarrow} \nu \mathbf{t}\ (P_2' \mid O \mid \overline{s}v)$. We note that $\nu \mathbf{t}\ (P_1' \mid O \mid \overline{s}v) \equiv_L \nu \mathbf{t}\ (P_1' \mid O)$ since $P_1 \stackrel{sv}{\rightarrow} P_1'$. We have two subcases.

$(L3)[2.1]$ Suppose $P_2 \stackrel{sv}{\Rightarrow} P_2'$. Then $P_2' \equiv_L (P_2' \mid \overline{s}v)$ and therefore $P_1' \mathcal{R} P_2'$ up to $\equiv_L$.

$(L3)[2.2]$ Suppose $P_2 \stackrel{\tau}{\Rightarrow} P_2'$. Then $P_1' \mathcal{R} (P_2' \mid \overline{s}v)$ up to $\equiv_L$.

$(L4)$ Suppose $(P_1 \mid S)\downarrow$ and $(P_1 \mid S) \mapsto P_1'$. We consider

$$R_1 = R(\mathbf{a})[S] \qquad R_2 = R(\mathbf{a})[S \mid \mathsf{pause}.O]$$

By $(C1)$, $\nu\mathbf{t}\ (P_1 \mid O) \mid R_i \approx_C \nu\mathbf{t}\ (P_2 \mid O) \mid R_i$ for $i = 1, 2$. Also

$$\nu\mathbf{t}\ (P_1 \mid O) \mid R_1 \stackrel{\tau}{\Rightarrow} \nu\mathbf{t}\ (P_1 \mid O \mid S)\downarrow$$

and

$$\nu\mathbf{t}\ (P_1 \mid O) \mid R_2 \stackrel{\tau}{\Rightarrow} \nu\mathbf{t}\ (P_1 \mid O \mid S \mid \mathsf{pause}.O) \mapsto \nu\mathbf{t}\ (P_1' \mid O)\ .$$

Then we must have:

(1) $\nu\mathbf{t}\ (P_2 \mid O) \mid R_1 \stackrel{\tau}{\Rightarrow} \nu\mathbf{t}\ (P_2'' \mid O \mid S)\downarrow$ and $\nu\mathbf{t}\ (P_1 \mid O \mid S) \approx_C \nu\mathbf{t}\ (P_2'' \mid O \mid S)$. By definition of O and R_1 this implies that $(P_2 \mid S) \stackrel{\tau}{\Rightarrow} (P_2'' \mid S)$ and $(P_2'' \mid S)\downarrow$.

(2) $\nu\mathbf{t}\ (P_2 \mid O) \mid R_2 \stackrel{\tau}{\Rightarrow} \nu\mathbf{t}\ (P_2'' \mid O \mid S \mid \mathsf{pause}.O) \mapsto \nu\mathbf{t}\ (P_2' \mid O)$ and $\nu\mathbf{t}\ (P_1' \mid O) \approx_C \nu\mathbf{t}\ (P_2' \mid O)$. Again by definition of O we have that $(P_2'' \mid S) \mapsto P_2'$. □

8 Conclusion

We have proposed a *synchronous* version of the π-calculus which borrows the notion of instant from the SL model–a relaxation of the ESTEREL model. We have shown that the resulting language is amenable to a semantic treatment similar to that available for the π-calculus. Retrospectively, we feel that the developed theory relies on two key insights: the introduction of the notion of L-suspension and the remark that the observation of signals is similar to the observation of channels with asynchronous communication.

References

[1] R. Amadio. The SL synchronous language, revisited. *Journal of Logic and Algebraic Programming*, 70:121-150, 2007.

[2] R. Amadio, G. Boudol, F. Boussinot and I. Castellani. Reactive programming, revisited. In Proc. Workshop on *Algebraic Process Calculi: the first* 25 *years and beyond*, *Electronic Notes in Theoretical Computer Science*, 162:49-60, 2006.

[3] R. Amadio, I. Castellani and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. In *Theoretical Computer Science*, 195:291-324, 1998.

[4] R. Amadio, S. Dal-Zilio. Resource control for synchronous cooperative threads. *Theoretical Computer Science* 358:229-254, 2006.

[5] D. Austry and G. Boudol. Algèbre de processus et synchronisation. In *Theoretical Computer Science*, 30:91-131, 1984.

[6] H. Barendregt. The lambda calculus. North-Holland, revised edition, 1984.

[7] M. Berger. Congruence for two timed asynchronous π-calculi. In Proc. *CONCUR*, Springer LNCS 3170:115-130, 2004.

[8] G. Berry and G. Gonthier. The Esterel synchronous programming language. *Science of computer programming*, 19(2):87–152, 1992.

[9] M. Boreale, R. De Nicola and R. Pugliese. Trace and testing equivalence on asynchronous processes. *Information and Computation*, 172(2):139-164, 2002.

[10] G. Boudol. ULM, a core programming model for global computing. In *Proc. of ESOP*, Springer LNCS 2986:234–248, 2004.

[11] F. Boussinot. Reactive C: An extension of C to program reactive systems. *Software Practice and Experience*, 21(4):401–428, 1991.

[12] F. Boussinot and R. De Simone. The SL synchronous language. *IEEE Trans. on Software Engineering*, 22(4):256–266, 1996.

[13] I. Castellani and M. Hennessy. Testing theories for asynchronous languages. In Proc *FST-TCS*, SLNCS 1530:90–101, 1998.

[14] P. Caspi and D. Pilaud and N. Halbwachs and J. Plaice. LUSTRE: a declarative language for programming synchronous systems. *ACM POPL*, pages 178-188, 1987.

[15] C. Fournet and G. Gonthier. A hierarchy of equivalences for asynchronous calculi (extended abstract) In *Proc. ICALP*, SLNCS 1443:844–855, 1998.

[16] M. Hennessy and J. Rathke. Bisimulations for a calculus of broadcasting systems. In *Theoretical Computer Science*, 200(1-2):225-260, 1998.

[17] K. Honda and N. Yoshida. On reduction-based process semantics. In *Theoretical Computer Science*, 151(2):437-486, 1995.

[18] J. Hopcroft and J. Ullman. Introduction to automata theory, languages, and computation. Prentice-Hall, 1989.

[19] L. Lamport and N. Lynch. Distributed computing: models and methods. In *Handbook of Theoretical Computer Science*, volume B. Elsevier, 1990.

[20] N. Lynch. Distributed algorithms. Morgan-Kaufmann, 1996.

[21] L. Mandel and M. Pouzet. ReactiveML, a reactive extension to ML. In *Proc. ACM Principles and Practice of Declarative Programming*, pages 82–93, 2005.

[22] A. Matos, G. Boudol and I. Castellani. Typing non-inteference for reactive programs. RR-INRIA 5594, June 2005. To appear in *Journal of Logic and Algebraic Programming.*

[23] M. Merro, F. Zappa Nardelli. Behavioral theory for mobile ambients. *Journal of the ACM*, 52(6):961-1023, 2005.

[24] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25(3):267–310, 1983.

[25] R. Milner. Communication and concurrency. Prentice-Hall, 1989.

[26] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts 1-2. *Information and Computation*, 100(1):1–77, 1992.

[27] R. Milner and D. Sangiorgi. Barbed bisimulation. In Proc. *ICALP*, SLNCS 623:685–695, 1992.

[28] J. Ousterhout. Why threads are a bad idea (for most purposes). Invited talk at the USENIX Technical Conference, 1996.

[29] K.V.S. Prasad. A calculus of broadcasting systems. In *Sci. Comput. Program.*, 25(2-3):285-327, 1995.

[30] Reactive programming, INRIA, Mimosa Project. `http://www-sop.inria.fr/mimosa/rp`.

[31] D. Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33(1):69-97, 1996.

[32] V. Saraswat, R. Jagadeesan, and V. Gupta. Timed default concurrent constraint programming. In *Journal of Symbolic computation*, 22(5,6) 475-520, 1996.

[33] M. Serrano, F. Boussinot, and B. Serpette. Scheme fair threads. In *Proc. ACM Principles and practice of declarative programming*, pages 203-214, 2004.

[34] G. Tel Introduction to distributed algorithms. Cambridge University Press, 1994