The Journal of Logic and Algebraic Programming 81 (2012) 162–180

Contents lists available at SciVerse ScienceDirect

The Journal of Logic and Algebraic Programming

journal homepage: www.elsevier.com/locate/jlap

Computing strong and weak bisimulations for psi-calculi

Magnus Johansson, Björn Victor *, Joachim Parrow

Dept. of Information Technology, Uppsala University, Box 337, SE-751 05 Uppsala, Sweden

ARTICLE INFO

Article history:
Available online 1 February 2012

Keywords:
Symbolic semantics
Bisimulation
Psi-calculi
Full abstraction

ABSTRACT

We present a symbolic transition system and strong and weak bisimulation equivalences for psi-calculi, and show that they are fully abstract with respect to bisimulation congruences in the non-symbolic semantics. A procedure which computes the most general constraint under which two agents are bisimilar is developed and proved correct.

A psi-calculus is an extension of the pi-calculus with nominal data types for data structures and for logical assertions representing facts about data. These can be transmitted between processes and their names can be statically scoped using the standard pi-calculus mechanism to allow for scope migrations. Psi-calculi can be more general than other proposed extensions of the pi-calculus such as the applied pi-calculus, the spi-calculus, the fusion calculus, or the concurrent constraint pi-calculus.

Symbolic semantics are necessary for an efficient implementation of the calculus in automated tools exploring state spaces, and the full abstraction property means the symbolic semantics makes exactly the same distinctions as the original.

1. Introduction

A multitude of extensions of the pi-calculus have been defined, allowing higher-level data structures and operations on them to be used as primitives when modelling applications. Ranging from integers, lists, or booleans to encryption/decryption or hash functions, the extensions increase the applicability of the basic calculus. In order to implement automated tools for analysis and verification using state space exploration (e.g. bisimilarity or model checking), each extended calculus needs a symbolic semantics, where the state space of agents is reduced to a manageable size – the non-symbolic semantics typically generates infinite state spaces even for very simple agents.

The extensions thus require added efforts both in developing the theory of the calculus for each variant, and in constructing specialised symbolic semantics for them. As the complexity of the extensions increases, producing correct results in these areas can be very hard. For example the labelled semantics of the applied pi-calculus [2] and of the concurrent constraint pi-calculus [15] have both turned out to be non-compositional in the sense that agents with the same semantics may become different when used in a parallel composition. Another example is the rather complex bisimulations which have been developed for the spi-calculus [3] (see [12] for an overview of non-symbolic bisimulations, or [11,13,14] for symbolic ones).

The psi-calculi [7,24] improve the situation: a single framework allows a range of specialised calculi to be formulated with a lean and compositional labelled semantics: with the parameters appropriately instantiated, the resulting calculus can be used to model applications such as cryptographic protocols and concurrent constraints, but also more advanced scenarios with polyadic synchronization or higher-order data and logics. The expressiveness and modelling convenience of psi-calculi exceeds that of earlier pi-calculus extensions, while the purity of the semantics is on par with the original pi-calculus. Its meta-theory has been proved using the theorem prover Isabelle [4,5].

* Corresponding author. Tel.: +46 70 4250239; fax: +46 18 511925.
E-mail addresses: magnus.johansson@it.uu.se (M. Johansson), bjorn.victor@it.uu.se (B. Victor), joachim.parrow@it.uu.se (J. Parrow).

1567-8326/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2012.01.001

In this paper we develop a symbolic semantics for psi-calculi, admitting large parts of this range of calculi to be verified more efficiently. We define symbolic versions of labelled bisimulation equivalence and its weak counterpart, and show that they are fully abstract with respect to the corresponding bisimulation congruences in the original semantics. This means that our new symbolic semantics does not change which processes are considered equivalent. This paper is an extended version of [23] that adds clarifications and proofs, a symbolic treatment of weak bisimulation, and a procedure to compute a constraint under which two agents are weakly bisimilar.

A symbolic semantics abstracts the values received in an input action. Instead of a possibly infinite branching of concrete values, a single name is used to represent them all. When the received values are used in conditional constructions (e.g. if-then-else) or as communication channels, we do not know their precise value, but need to record the constraints which must be satisfied for a resulting transition to be valid.

A (non-symbolic) psi-calculus transition has the form $\Psi \rhd P \xrightarrow{\alpha} P'$, with the intuition that P can perform α leading to P' in an environment that asserts Ψ. For example, suppose P can do an α to P'. If we can deduce $\text{prime}(x)$ from the environment then **if** $\text{prime}(x)$ **then** P can make an α-transition to P', e.g.

$$\{x = 3\} \rhd \textbf{if } \text{prime}(x) \textbf{ then } P \xrightarrow{\alpha} P'.$$

In the symbolic semantics where we might not have the precise value of x, we instead decorate the transition with its requirement, so for any Ψ we have

$$\Psi \rhd \textbf{if } \text{prime}(x) \textbf{ then } P \xrightarrow[C \wedge \{\!|\Psi \vdash \text{prime}(x)|\!\}]{\alpha} P'$$

where C is the requirement for P to do an α to P' in the environment Ψ. Constraints also arise from communication between parallel agents, where, in the symbolic case, the precise channels might not be known; instead we allow communication over symbolic representations of channels and record the requirement in a transition constraint. As an example consider

$$a(x) \,.\, a(y) \,.\, (\overline{x}\, x \,.\, P \mid y(z) \,.\, Q)$$

which after its initial inputs only has symbolic values of x and y. The resulting agent has the symbolic transition

$$\Psi \rhd \overline{x}\, x \,.\, P \mid y(z) \,.\, Q \xrightarrow[\{\!|\Psi \vdash x \dot{\leftrightarrow} y|\!\}]{\tau} P \mid Q[z := x]$$

where $x \dot{\leftrightarrow} y$ means that x and y represent the same channel.

Communication channels in psi-calculi may be structured data terms, not only names. This leads to a new possibility of infinite branching: a subject in a prefix may be rewritten to another equivalent term before it is used in a transition. E.g. when $\text{first}(x, y)$ and x represent the same channel, $P = \overline{\text{first}(a, b)}c \,.\, P' \xrightarrow{\overline{a}c} P'$, but also $P \xrightarrow{\overline{\text{first}(a,c)}c} P'$, etc. The possibility of using structured channels gives significant expressive power (see [7]). Our symbolic semantics abstracts the equivalent forms of channel subject by using a fresh name as subject, and adds a suitable constraint to the transition label.

Given the symbolic semantics we proceed to define symbolic bisimulations, both strong and weak, closely following [21]. A symbolic bisimulation is a ternary relation containing triples (C, P, Q), where C is a constraint that denotes under which conditions P and Q are bisimilar. As an example, if

$$Q \quad = \quad \textbf{if } x = 3 \textbf{ then } P \textbf{ else } P$$

we have that P and Q are bisimilar under the constraint **true**. To see this consider $P \xrightarrow[C_P]{\alpha} P'$ (eliding the environmental assertion Ψ). The definition of simulation allows a case analysis to partition C_P into an equivalent disjunction and requires that Q can simulate for each disjunct, e.g.

$$C_P \quad \Leftrightarrow \quad (C_P \wedge \{\!|x = 3|\!\}) \vee (C_P \wedge \{\!|x \neq 3|\!\})$$

and $Q \xrightarrow[C_P \wedge \{\!|x=3|\!\}]{\alpha} P'$ and $Q \xrightarrow[C_P \wedge \{\!|x\neq 3|\!\}]{\alpha} P'$.

This partitioning is the key to a sound and complete symbolic semantics. Weak symbolic bisimulation is defined in essentially the same way, but requires Q to simulate with weak transitions that treats τ-transitions as invisible.

Finally we present a depth-first algorithm which computes the most general constraint under which a pair of agents are bisimilar, again closely following [21]. It follows transitions from pairs of agents and adds them to a table that ultimately will be a bisimulation. The algorithm assumes that the agents have finite symbolic transition graphs.

1.1. Comparison to related work

Symbolic bisimulations for process calculi have a long history. Our work is to a large extent based on the pioneering work by Hennessy and Lin [21] for value-passing CCS, later specialised for the pi-calculus by Boreale and De Nicola [10] and independently by Lin [26,27]. While [21] is parametrised by general boolean expressions on an underlying data signature

it does not handle names and mobility; on the other hand [10,26,27] handle *only* names and no other data structures. The number of follow-up works to these is huge, with applications ranging from pi-calculus to constraint programming; here we focus on the relation to the ones for applied pi-calculus and spi-calculus.

The existing tools for calculi based on the applied pi-calculus (e.g. [1,8,9]), are not fully abstract with regards to bisimulation. A symbolic semantics and bisimulation for applied pi-calculus has been defined in [17], but it is not complete. A complete version is instead defined in [29], and an axiomatisation is given in [30]. The original labelled bisimulation of applied pi-calculus is however not compositional (see [7]). The situation for the spi-calculus is better: fully abstract symbolic bisimulation for hedged bisimulation has been defined in [11], and for open hedged bisimulation in [14]. According to the authors, neither is directly mechanizable. The only symbolic bisimulation which to our knowledge has been implemented in a tool is not fully abstract [13].

It can be argued [13] that incompleteness is not a problem when verifying authentication and secrecy properties of security protocols, which appears to have been the main application of the applied pi-calculus so far. When going beyond security analysis we claim, based on experience from the Mobility Workbench [33], that completeness is very important: when analysing agents with huge state spaces, a positive result (the agents are equivalent) may be more difficult to achieve than a negative result (the agents differ). However, such a negative result can only be trusted if the analysis is fully abstract.

Our symbolic semantics is relatively simple compared to the ones presented for the applied pi-calculus or spi-calculus. In relation to the former, we are helped significantly by the absence of structural equivalence rules, which in the applied pi-calculus are rather complex. In [16,29] an intermediate semantics is used to handle the complexity, while in contrast we can directly relate the original and symbolic semantics. In relation to the symbolic semantics for the spi-calculus, our semantics has a straight-forward treatment of scope opening due to the simpler psi-calculi semantics. In addition, the complexities of spi-calculus bisimulations are necessarily inherited by the symbolic semantics, introducing, e.g. explicit environment knowledge representations with timestamps on messages and variables. In psi-calculi, bisimulation is much simpler and the symbolic counterpart is not significantly more complex than the one for value-passing CCS.

Disposition: In the next section we review the basic definitions of syntax, semantics, and strong bisimulation of psi-calculi. Section 3 presents the symbolic semantics and bisimulation, and illustrates the concrete and symbolic transitions and bisimulations by examples. In Section 4 we turn to the weak semantics and bisimulation for psi-calculi, and Section 5 describes the symbolic counterparts. In Section 6 we show our main results: the correspondence between concrete and symbolic transitions and the full abstraction of bisimulations. Section 7 presents and proves correct an algorithm to compute a constraint under which two agents are bisimilar, while Section 8 concludes and presents plans and ideas for future work. Detailed proofs of results in Sections 6 and 7 can be found in [25].

2. Psi-calculi

This section is a brief recapitulation of psi-calculi and nominal data types. Unless stated otherwise all definitions are from [7], which contains a more extensive treatment including motivations and examples.

2.1. Nominal datatypes

We assume a countably infinite set of atomic *names* $\mathcal{N}$ ranged over by $a, b, \ldots, x, y, z$. Intuitively, names will represent the symbols that can be statically scoped, and also represent symbols acting as variables in the sense that they can be subjected to substitution. A *nominal set* [18,31] is a set equipped with *name swapping* functions written $(a\ b)$, for any names a, b. An intuition is that for any member X of a nominal set it holds that $(a\ b) \cdot X$ is X with a replaced by b and b replaced by a. One main point of this is that even though we have not defined any particular syntax we can define what it means for a name to "occur" in an element: it is simply that it can be affected by swappings. The names occurring in this way in an element X constitute the *support* of X, written $\mathrm{n}(X)$. As an example, consider the terms of a lambda-calculus quotiented by alpha-equivalence in the ordinary sense, i.e. capture-avoiding renaming of bound names. Let X be such an equivalence class. Then all members of X have the same set of free names, so we can unambiguously say that this set is the free names of X itself, and this corresponds to the support of X. The advantage with this formulation is that it lets us reason directly in terms of equivalence classes of alpha-conversion. In all definitions to come we identify alpha-variants of bound names in this way, i.e. we implicitly consider equivalence classes of alpha-equivalence. We refer to [7] for the formal definitions and more examples.

We write $a\#X$, pronounced "a is fresh for X", for $a \notin \mathrm{n}(X)$. If A is a set of names we write $A\#X$ to mean $\forall a \in A\ .\ a\#X$. We require all elements to have finite support, i.e. $\mathrm{n}(X)$ is finite for all X.

A function f on nominal sets is *equivariant* if $(a\ b) \cdot f(X) = f((a\ b) \cdot X)$ holds for all X, a, b, and similarly for functions and relations of any arity. Intuitively, this means that all names are treated equally.

A *nominal datatype* is just a nominal set together with a set of functions on it. In particular we shall consider a substitution function which intuitively substitutes elements for names. If X is an element of a datatype, $\tilde{a}$ is a sequence of names without duplicates and $\tilde{Y}$ is an equally long sequence of elements of possibly another datatype, the *substitution* $X[\tilde{a} := \tilde{Y}]$ is an

element of the same datatype as X. We need not define exactly what a substitution does; it is enough to assume the following properties:

1: if $\tilde{a} \subseteq \mathrm{n}(X)$ and $b \in \mathrm{n}(\tilde{T})$ then $b \in \mathrm{n}(X[\tilde{a} := \tilde{T}])$

2: if $\tilde{b}\#X, \tilde{a}$ then $X[\tilde{a} := \tilde{T}] = ((\tilde{b}\ \tilde{a}) \cdot X)[\tilde{b} := \tilde{T}]$

The first says that a substitution $X[\tilde{a} := \tilde{T}]$ may not erase names in $\tilde{T}$, and the second is a kind of alpha-conversion; see [7] for further explanations.

2.2. Agents

A psi-calculus is defined by instantiating three nominal data types and four operators:

Definition 1 (Psi-calculus parameters)**.** A psi-calculus requires the three (not necessarily disjoint) nominal data types:

T the (data) terms, ranged over by M, N

C the conditions, ranged over by φ

A the assertions, ranged over by Ψ

and the four equivariant operators:

$\leftrightarrow : \mathbf{T} \times \mathbf{T} \to \mathbf{C}$	Channel Equivalence
$\otimes : \mathbf{A} \times \mathbf{A} \to \mathbf{A}$	Composition
$\mathbf{1} : \mathbf{A}$	Unit
$\vdash \subseteq \mathbf{A} \times \mathbf{C}$	Entailment

and substitution functions $[\tilde{a} := \tilde{M}]$, substituting terms for names, on all of **T**, **C**, and **A**.

The binary functions above will be written in infix. Thus, if M and N are terms then $M \leftrightarrow N$ is a condition, pronounced "M and N are channel equivalent" and if Ψ and Ψ' are assertions then so is $\Psi \otimes \Psi'$. Also we write $\Psi \vdash \varphi$, "Ψ entails φ", for $(\Psi, \varphi) \in \vdash$.

The data terms are used to represent all kinds of data, including communication channels. Conditions are used as guards in agents, and $M \leftrightarrow N$ is a particular condition saying that M and N represent the same channel. The assertions will be used to declare information necessary to resolve the conditions. Assertions can be contained in agents and thus represent information postulated by that agent; they can contain names and thereby be syntactically scoped and thus represent information known only to the agents within that scope. The intuition of entailment is that $\Psi \vdash \varphi$ means that given the information in Ψ, it is possible to infer φ. We say that two assertions are equivalent if they entail the same conditions:

Definition 2 (Assertion equivalence)**.** Two assertions are *equivalent*, written $\Psi \simeq \Psi'$, if for all φ we have that $\Psi \vdash \varphi \Leftrightarrow \Psi' \vdash \varphi$.

A psi-calculus is formed by instantiating the nominal data types and operators so that the following requisites are satisfied:

Definition 3 (Requisites on valid psi-calculus parameters)**.**

Channel symmetry: $\Psi \vdash M \leftrightarrow N \implies \Psi \vdash N \leftrightarrow M$

Channel transitivity: $\Psi \vdash M \leftrightarrow N \wedge \Psi \vdash N \leftrightarrow L \implies \Psi \vdash M \leftrightarrow L$

Composition: $\Psi \simeq \Psi' \implies \Psi \otimes \Psi'' \simeq \Psi' \otimes \Psi''$

Identity: $\Psi \otimes \mathbf{1} \simeq \Psi$

Associativity: $(\Psi \otimes \Psi') \otimes \Psi'' \simeq \Psi \otimes (\Psi' \otimes \Psi'')$

Commutativity: $\Psi \otimes \Psi' \simeq \Psi' \otimes \Psi$

Weakening: $\Psi \vdash \varphi \implies \Psi \otimes \Psi' \vdash \varphi$

Names are terms: $\mathcal{N} \subseteq \mathbf{T}$

Our requisites on a psi-calculus are that the channel equivalence is a partial equivalence relation, that $\otimes$ preserves equivalence, and that the equivalence classes of assertions form an abelian monoid. The last two, Weakening and Names are terms, are not required in our previous expositions of psi-calculi. Weakening means that non-monotonic logics cannot be used. It simplifies our proofs in the present paper although we do not know if it is absolutely necessary. It is only used in one place in the proof of Theorem 32. Furthermore it allows us to only consider the simpler definition of weak bisimulation from [24] presented in Section 4. The requisite Names are terms simplifies the symbolic semantics; also here we do not know if it is strictly necessary.

In the following $\tilde{a}$ means a finite (possibly empty) sequence of names, $a_1, \ldots, a_n$. The empty sequence is written ϵ and the concatenation of $\tilde{a}$ and $\tilde{b}$ is written $\tilde{a}\tilde{b}$. When occurring as an operand of a set operator, $\tilde{a}$ means the corresponding set of names $\{a_1, \ldots, a_n\}$. We also use sequences of terms, conditions, assertions etc. in the same way.

A *frame* can intuitively be thought of as an assertion with local names:

Definition 4 (Frame)**.** A *frame* is of the form $(\nu\tilde{b})\Psi$ where $\tilde{b}$ is a sequence of names that bind into the assertion Ψ. We identify alpha variants of frames.

We use F, G to range over frames. Since we identify alpha variants we can always choose the bound names freely.

Notational conventions: We write just Ψ for $(\nu\epsilon)\Psi$ when there is no risk of confusing a frame with an assertion, and $\otimes$ to mean composition on frames defined by $(\nu\tilde{b}_1)\Psi_1 \otimes (\nu\tilde{b}_2)\Psi_2 = (\nu\tilde{b}_1\tilde{b}_2)(\Psi_1 \otimes \Psi_2)$ where $\tilde{b}_1 \# \tilde{b}_2$, Ψ_2 and vice versa. We write $(\nu c)((\nu\tilde{b})\Psi)$ to mean $(\nu c\tilde{b})\Psi$.

Intuitively a condition is entailed by a frame if it is entailed by the assertion and does not contain any names bound by the frame. Two frames are equivalent if they entail the same conditions:

Definition 5 (Equivalence of frames)**.** We define $F \vdash \varphi$ to mean that there exists an alpha variant $(\nu\tilde{b})\Psi$ of F such that $\tilde{b}\#\varphi$ and $\Psi \vdash \varphi$. We also define $F \simeq G$ to mean that for all φ it holds that $F \vdash \varphi$ iff $G \vdash \varphi$.

Definition 6 (Psi-calculus agents)**.** Given valid psi-calculus parameters as in Definitions 1 and 3, the psi-calculus *agents*, ranged over by P, Q, . . ., are of the following forms.

$\mathbf{0}$	Nil
$\overline{M}\,N\,.\,P$	Output
$\underline{M}(x).P$	Input
$\mathbf{case}\ \varphi_1 : P_1\ [] \cdots []\ \varphi_n : P_n$	Case
$(\nu a)P$	Restriction
$P \mid Q$	Parallel
$!P$	Replication
$(\!\mid \Psi \mid\!)$	Assertion

In the Input $\underline{M}(x)\,.\,P$, x binds its occurrences in P. Restriction binds a in P. An assertion is *guarded* in an agent P if every occurrence is in an Input or Output subexpression of P. An agent is *well formed* if in a replication $!P$ there are no unguarded assertions in P, and if in $\mathbf{case}\ \varphi_1 : P_1\ [] \cdots []\ \varphi_n : P_n$ there are no unguarded assertion in any P_i.

In the Output and Input forms M is called the subject and N and x the objects, respectively. Output and Input are similar to those in the pi-calculus, but arbitrary terms can function as both subjects and objects. [1] The **case** construct works by performing the action of any P_i for which the corresponding φ_i is true. So it embodies both an **if** (if there is only one branch) and an internal non-deterministic choice (if the conditions are overlapping). It is sometimes written as $\mathbf{case}\ \tilde{\varphi} : \tilde{P}$, or if $n = 1$ as $\mathbf{if}\ \varphi_1\ \mathbf{then}\ P_1$. The input subject is underlined to facilitate parsing of complicated expressions; in simple cases we often conform to a more traditional notation and omit the underline.

2.3. Operational semantics and bisimulation

In the standard pi-calculus the transitions from a parallel composition $P \mid Q$ can be uniquely determined by the transitions from its components, but in psi-calculi the situation is more complex. Here the assertions contained in P can affect the

[1] Our previous exposition [7] uses a more general form of input with pattern matching; as we discuss in Section 8 we here restrict attention to the traditional input form with one bound name, $\underline{M}(x)$, for simplicity. The general form of input prefix in [7] is $M(\lambda\tilde{x})N$, where N is a term, and $\underline{M}(x)$ is simply a short hand for the special case $M(\lambda x)x$.

Table 1
Late operational semantics. Symmetric versions of Com and Par are elided. In the rule Com we assume that $\mathcal{F}(P) = (\nu\tilde{b}_P)\Psi_P$ and $\mathcal{F}(Q) = (\nu\tilde{b}_Q)\Psi_Q$ where $\tilde{b}_P$ is fresh for all of $\Psi, \tilde{b}_Q, Q, M$ and P, and that $\tilde{b}_Q$ is correspondingly fresh. In the rule Par we assume that $\mathcal{F}(Q) = (\nu\tilde{b}_Q)\Psi_Q$ where $\tilde{b}_Q$ is fresh for Ψ, P and α. In Open the expression $\nu\tilde{a} \cup \{b\}$ means the sequence $\tilde{a}$ with b inserted anywhere.

$$\textsc{In}\ \frac{\Psi \vdash M \leftrightarrow K}{\Psi \rhd \underline{M}(x).P \xrightarrow{\underline{K}(x)} P} \qquad \textsc{Out}\ \frac{\Psi \vdash M \leftrightarrow K}{\Psi \rhd \overline{M}N.P \xrightarrow{\overline{K}N} P}$$

$$\textsc{Case}\ \frac{\Psi \rhd P_i \xrightarrow{\alpha} P' \quad \Psi \vdash \varphi_i}{\Psi \rhd \mathbf{case}\ \tilde{\varphi} : \tilde{P} \xrightarrow{\alpha} P'}$$

$$\textsc{Com}\ \frac{\begin{array}{c}\Psi\otimes\Psi_P\otimes\Psi_Q \vdash M \leftrightarrow K \\ \Psi_Q\otimes\Psi \rhd P \xrightarrow{\overline{M}(\nu\tilde{a})N} P' \qquad \Psi_P\otimes\Psi \rhd Q \xrightarrow{\underline{K}(x)} Q'\end{array}}{\Psi \rhd P \mid Q \xrightarrow{\tau} (\nu\tilde{a})(P' \mid Q'[x := N])}\ \tilde{a}\#Q$$

$$\textsc{Par}\ \frac{\Psi_Q\otimes\Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathrm{bn}(\alpha)\#Q$$

$$\textsc{Scope}\ \frac{\Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd (\nu b)P \xrightarrow{\alpha} (\nu b)P'}\ b\#\alpha, \Psi$$

$$\textsc{Open}\ \frac{\Psi \rhd P \xrightarrow{\overline{M}(\nu\tilde{a})N} P'}{\Psi \rhd (\nu b)P \xrightarrow{\overline{M}(\nu\tilde{a}\cup\{b\})N} P'}\ b\#\tilde{a}, \Psi, M \qquad b \in \mathrm{n}(N) \qquad \textsc{Rep}\ \frac{\Psi \rhd P \mid !P \xrightarrow{\alpha} P'}{\Psi \rhd !P \xrightarrow{\alpha} P'}$$

conditions tested in Q and vice versa. For this reason we introduce the notion of the *frame of an agent* as the combination of its top level assertions, retaining all the binders. It is precisely this that can affect a parallel agent.

Definition 7 (Frame of an agent)**.** The *frame* $\mathcal{F}(P)$ *of an agent* P is defined inductively as follows:

$$\mathcal{F}(\mathbf{0}) = \mathcal{F}(\underline{M}(x).P) = \mathcal{F}(\overline{M}N.P) = \mathcal{F}(\mathbf{case}\ \tilde{\varphi} : \tilde{P}) = \mathcal{F}(!P) = \mathbf{1}$$

$$\mathcal{F}((\!|\Psi|\!)) = \Psi$$

$$\mathcal{F}(P \mid Q) = \mathcal{F}(P) \otimes \mathcal{F}(Q)$$

$$\mathcal{F}((\nu b)P) = (\nu b)\mathcal{F}(P)$$

Our previous presentation of psi-calculi [7] gives a semantics of an early kind, where input actions are of kind $\underline{M}\,N$. In the present paper we give an operational semantics of the late kind, meaning that the labels of input transitions contain variables, in this case represented as names, for the object to be received. With this kind of semantics it is easier to establish a relation to the symbolic semantics.

Definition 8 (Actions)**.** The *actions* ranged over by α, β are of the following three kinds: $\overline{M}\,(\nu\tilde{a})N$ (Output, where $\tilde{a} \subseteq \mathrm{n}(N)$), $\underline{M}(x)$ (Input), and τ (Silent).

For actions we refer to M as the *subject* and N and x as the *objects*. We let $\mathrm{subj}(\overline{M}\,(\nu\tilde{a})N) = \mathrm{subj}(\underline{M}(x)) = M$. We define $\mathrm{bn}(\overline{M}\,(\nu\tilde{a})N) = \tilde{a}$, $\mathrm{bn}(\underline{M}(x)) = \{x\}$, and $\mathrm{bn}(\tau) = \emptyset$. The support of α is the empty set for $\alpha = \tau$, and $\mathrm{n}(M) \cup \mathrm{n}(N) \cup \mathrm{bn}(\alpha)$ for Input and Output.

Definition 9 (Transitions)**.** A *transition* is of the kind $\Psi \rhd P \xrightarrow{\alpha} P'$, meaning that when the environment contains the assertion Ψ the agent P can do an α to become P'. The transitions are defined inductively in Table 1.

We use alpha-conversion in its standard sense, i.e. a capture-avoiding swapping of bound names. Both agents and frames are identified by alpha equivalence. This means that we can choose the bound names fresh in the premise of a rule. In a transition the names in $\mathrm{bn}(\alpha)$ count as binding into both the action object and the derivative, and transitions are identified up to alpha equivalence. This means that the bound names can be chosen fresh, substituting each occurrence in both the object and the derivative. Therefore $\mathrm{bn}(\alpha)$ is in the support of the output action: otherwise it could be alpha-converted in the action alone. For the side conditions in Scope and Open it is important that $\mathrm{bn}(\alpha) \subseteq \mathrm{n}(\alpha)$. In rules Par and Com, the freshness conditions on the involved frames will ensure that if a name is bound in one agent its representative in a frame is distinct from names in parallel agents, and also (in Par) that it does not occur on the transition label. We defer a more precise account of this to [7].

We proceed to define early bisimulation with the late semantics:

Table 2
Early structured operational semantics. All other rules are as in the late semantics of Fig. 1.

$$\textsc{In}\ \frac{\Psi \vdash M \leftrightarrow K}{\Psi \rhd \underline{M}(x).P \xrightarrow{\underline{K}N} P[x := N]}$$

$$\textsc{Com}\ \frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{\overline{M}(\nu\tilde{a})N} P' \qquad \Psi_P \otimes \Psi \rhd Q \xrightarrow{\underline{K}N} Q' \qquad \Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K}{\Psi \rhd P \mid Q \xrightarrow{\tau} (\nu\tilde{a})(P' \mid Q')}\ \tilde{a}\#Q$$

Definition 10 ((Early) Bisimulation)**.** A *bisimulation* $\mathcal{R}$ is a ternary relation between assertions and pairs of agents such that $\mathcal{R}(\Psi, P, Q)$ implies all of

(1) Static equivalence: $\Psi \otimes \mathcal{F}(P) \simeq \Psi \otimes \mathcal{F}(Q)$
(2) Symmetry: $\mathcal{R}(\Psi, Q, P)$
(3) Extension of arbitrary assertion: $\forall \Psi'.\ \mathcal{R}(\Psi \otimes \Psi', P, Q)$
(4) Simulation: for all α, P' such that $\mathrm{bn}(\alpha)\#\Psi, Q$
   (a) if $\alpha = \underline{M}(x)$: $\Psi \rhd P \xrightarrow{\alpha} P' \Longrightarrow$
   $\forall L \exists Q'.\ \Psi \rhd Q \xrightarrow{\alpha} Q'$ and $\mathcal{R}(\Psi, P'[x := L], Q'[x := L])$.
   (b) otherwise: $\Psi \rhd P \xrightarrow{\alpha} P' \Longrightarrow \exists Q'.\ \Psi \rhd Q \xrightarrow{\alpha} Q'$ and $\mathcal{R}(\Psi, P', Q')$.

We define $P \dot{\sim} Q$ to mean that there exists a bisimulation $\mathcal{R}$ such that $\mathcal{R}(\mathbf{1}, P, Q)$. We also define $P \sim Q$ to mean that $P\sigma \dot{\sim} Q\sigma$, for all σ, where σ is a sequence of substitutions $[x_1 := L_1][x_2 := L_2] \ldots [x_n := L_n]$.

2.4. Relation to early semantics

In this subsection we formulate the relation between the semantics and bisimulation in the preceding subsection and the original in [7].

Table 2 gives the rules for input and communication of an early kind used in [7]. The following lemma clarifies the relation between the two semantics:

Lemma 11 (Late and early transitions)**.**

(1) $\Psi \rhd P \xrightarrow{\underline{M}N} P'$ in the early semantics iff there exist P'', and x such that $\Psi \rhd P \xrightarrow{\underline{M}(x)} P''$ in the late semantics, where $P' = P''[x := N]$.
(2) For output and τ actions, $\Psi \rhd P \xrightarrow{\alpha} Q$ in the early semantics iff the same transition can be derived in the late semantics.

The proof is by induction over the transition derivations. In the proof of (2), the case $\alpha = \tau$ needs both (1) and the case where α is an output. See [22] for further details.

Lemma 12. *A relation is a bisimulation according to Definition 10 precisely if it is a bisimulation according to [7].*

The proof is straightforward using Lemma 11. As a corollary the algebraic properties of $\sim$ established in [7] hold, notably that it is a congruence. See [22] for further details.

3. Symbolic semantics and equivalence

The idea behind a symbolic semantics is to reduce the state space of agents. One standard way is to avoid infinite branching in inputs by using a fresh name to represent whatever was received.

In psi-calculi there is an additional source of infinite branching: a subject in a prefix may get rewritten to many terms. Also here we use a fresh name to represent these terms. This means that the symbolic actions are the same as the concrete actions with the exception that only names are used as subjects.

A *symbolic transition* is of form

$$\Psi \rhd P \xrightarrow[C]{\alpha} P'$$

The intuition is that this represents a set of concrete transitions, namely those that satisfy the constraint C. Before the formal definitions we here briefly explain the rationale. Consider a psi-calculus with integers and integer equations; for example a condition can be "$x = 3$". An example agent is $P = \mathbf{case}\ x = 3 : P'$. If $P' \xrightarrow[C']{\alpha} P''$, then there should clearly be a transition $P \xrightarrow[C \wedge C']{\alpha} P''$ for some constraint C that captures that x must be 3. One context that can make this constraint

Table 3
Transition rules for the symbolic semantics. Symmetric versions of Com and Par are elided. In the rule Com we assume that $\mathcal{F}(P) = (\nu\widetilde{b_P})\Psi_P$ and $\mathcal{F}(Q) = (\nu\widetilde{b_Q})\Psi_Q$ where $\widetilde{b_P}$ is fresh for all of Ψ, $\widetilde{b}_Q$, Q and P, and that $\widetilde{b}_Q$ is correspondingly fresh. We also assume that $y, z\#\Psi, \widetilde{b}_P, P, \widetilde{b}_Q, Q, N, \widetilde{a}$. In Com, $C_{com} = (\nu\widetilde{b_P}, \widetilde{b_Q})\{\!|\Psi' \vdash M_P \leftrightarrow M_Q|\!\} \wedge (\nu\widetilde{b_Q})C_P \wedge (\nu\widetilde{b_P})C_Q$. In the rule Par we assume that $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ where $\widetilde{b}_Q$ is fresh for Ψ, P and α. In Open the expression $\nu\widetilde{a} \cup \{b\}$ means the sequence $\widetilde{a}$ with b inserted anywhere.

$$\textsc{In}\ \frac{}{\Psi \rhd \underline{M}(x).P \xrightarrow[\{\!|\Psi\vdash M\leftrightarrow y|\!\}]{\underline{y}(x)} P}\ y\#\Psi, M, P, x$$

$$\textsc{Case}\ \frac{\Psi \rhd P_i \xrightarrow[C]{\alpha} P'}{\Psi \rhd \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \xrightarrow[C\wedge\{\!|\Psi\vdash\varphi_i|\!\}]{\alpha} P'}$$

$$\textsc{Out}\ \frac{}{\Psi \rhd \overline{M}N.P \xrightarrow[\{\!|\Psi\vdash M\leftrightarrow y|\!\}]{\overline{y}N} P}\ y\#\Psi, M, N, P$$

$$\textsc{Com}\ \frac{\Psi_Q\otimes\Psi \rhd P \xrightarrow[(\nu\widetilde{b_P})\{\!|\Psi'\vdash M_P\leftrightarrow y|\!\}\wedge C_P]{\overline{y}(\nu\widetilde{a})N} P' \quad \Psi_P\otimes\Psi \rhd Q \xrightarrow[(\nu\widetilde{b_Q})\{\!|\Psi'\vdash M_Q\leftrightarrow z|\!\}\wedge C_Q]{\underline{z}(x)} Q'}{\Psi \rhd P \mid Q \xrightarrow[C_{com}]{\tau} (\nu\widetilde{a})(P' \mid Q'[x := N])}\ \begin{array}{l}\widetilde{a}\#Q,\\ y\#z\\ \Psi' = \Psi\otimes\Psi_P\otimes\Psi_Q\end{array}$$

$$\textsc{Par}\ \frac{\Psi\otimes\Psi_Q \rhd P \xrightarrow[C]{\alpha} P'}{\Psi \rhd P \mid Q \xrightarrow[(\nu\widetilde{b}_Q)C]{\alpha} P' \mid Q}\ \begin{array}{l}\mathrm{bn}(\alpha)\#Q\\ \alpha = \tau \vee \mathrm{subj}(\alpha)\#Q\end{array}$$

$$\textsc{Scope}\ \frac{\Psi \rhd P \xrightarrow[C]{\alpha} P'}{\Psi \rhd (\nu b)P \xrightarrow[(\nu b)C]{\alpha} (\nu b)P'}\ b\#\alpha, \Psi$$

$$\textsc{Open}\ \frac{\Psi \rhd P \xrightarrow[C]{\overline{y}(\nu\widetilde{a})N} P'}{\Psi \rhd (\nu b)P \xrightarrow[(\nu b)C]{\overline{y}(\nu\widetilde{a}\cup\{b\})N} P'}\ b \in \mathrm{n}(N)\ b\#\widetilde{a}, \Psi, y \qquad \textsc{Rep}\ \frac{\Psi \rhd P \mid !P \xrightarrow[C]{\alpha} P'}{\Psi \rhd !P \xrightarrow[C]{\alpha} P'}$$

true is an input, as in $a(x).P$. The input will give rise to a substitution for x, and if the substitution sends x to 3 the constraint is satisfied. In this way the constraints are similar to those for the pi-calculus [10,26]. In psi-calculi there is an additional way that a context can enable the transition: it can contain an assertion as in $(\!|x = 3|\!) \mid P$. Here we should have a transition $(\!|x = 3|\!) \mid P \xrightarrow[C']{\alpha} (\!|x = 3|\!) \mid P''$. Therefore a solution of a constraint will contain both a substitution of terms for names (representing the effect of an input) and an assertion (representing the effect of a parallel component).

Definition 13. A *solution* is a pair (σ, Ψ) where σ is a substitution sequence of terms for names, and Ψ is an assertion. The *transition constraints*, ranged over by C, C_t and corresponding solutions, $\mathrm{sol}(C)$ are defined by:

	Constraint	Solutions
$C, C' ::=$	**true**	$\{(\sigma, \Psi) \ : \ \sigma$ is a substitution sequence $\wedge\ \Psi \in \mathbf{A}\}$
	false	$\emptyset$
	$(\nu\widetilde{a})\{\!|\Psi \vdash \varphi|\!\}$	$\{(\sigma, \Psi') \ : \ \exists\widetilde{b}.\widetilde{b}(\#\sigma, \Psi', \Psi, \varphi) \wedge ((\widetilde{a}\ \widetilde{b}) \cdot \Psi)\sigma\otimes\Psi' \vdash ((\widetilde{a}\ \widetilde{b}) \cdot \varphi)\sigma\}$
	$C \wedge C'$	$\mathrm{sol}(C) \cap \mathrm{sol}(C')$

In $(\nu\widetilde{a})\{\!|\Psi \vdash \varphi|\!\}, \widetilde{a}$ are binding occurrences into Ψ and φ. We let $(\nu\widetilde{a})(C \wedge C')$ mean $(\nu\widetilde{a})C \wedge (\nu\widetilde{a})C'$, and we let $(\nu\widetilde{a})$**true** mean **true**, and similarly for **false**. We adopt the notation $(\sigma, \Psi) \models C$ to say that $(\sigma, \Psi) \in \mathrm{sol}(C)$.

A transition constraint C defines a set of solutions $\mathrm{sol}(C)$, namely those where the entailment becomes true by applying the substitution and adding the assertion. For example, the transition constraint $\{\!|\mathbf{1} \vdash x = 3|\!\}$ has solutions $([x := 3], \mathbf{1})$ and $(\mathrm{Id},\ x = 3)$, where Id is the identity substitution.

The purpose of the ν-binder in constraints is just to exclude names for use in solutions; this motivates that $(\nu\widetilde{a})(C \wedge C') = (\nu\widetilde{a})C \wedge (\nu\widetilde{a})C'$. In contrast, the ν-binder on a frame or an agent postulates the existence of a local name and does not distribute over operators in that way.

The structured operational symbolic semantics is defined in Table 3. First consider the Out rule: $\Psi \rhd \overline{M}N.P \xrightarrow[\{\!|\Psi\vdash M\leftrightarrow y|\!\}]{\overline{y}N} P$. This constraint means the transition can be taken in any solution that implies that the subject M of the syntactic prefix is channel equivalent to y.

The rule Com is of particular interest. The intuition is that the symbolic action subjects are placeholders for the values M_P and M_Q. In the conclusion the constraint is that these are channel equivalent, while y and z will not occur again.

We will often write $P \xrightarrow[C]{\alpha} P'$ for $\mathbf{1} \rhd P \xrightarrow[C]{\alpha} P'$.

3.1. Symbolic bisimulation

In order to define a symbolic bisimulation we need additional kinds of constraints. If a process P does a bound output $\overline{y}\ (\nu\widetilde{a})N$ that is matched by a bound output $\overline{y}\ (\nu\widetilde{a})N'$ from Q we need constraints that keep track of the fact that N and N' should be syntactically the same, and that $\widetilde{a}$ is sufficiently fresh.

Definition 14. The *constraints* **Cstr**, ranged over by C, are of the forms

	Constraint	The solutions sol(C) are all pairs (σ, Ψ) such that
$C, C' ::=$	C_t	$(\sigma, \Psi) \models C_t$
	$\{\!\|M = N\|\!\}$	$M\sigma = N\sigma$
	$\{\!\|a\#X\|\!\}$	$(a\#X)\sigma$ and $a\#dom(\sigma)$
	$C \wedge C'$	$(\sigma, \Psi) \models C$ and $(\sigma, \Psi) \models C'$
	$C \vee C'$	$(\sigma, \Psi) \models C$ or $(\sigma, \Psi) \models C'$
	$C \Rightarrow C'$	$\forall\Psi'.(\sigma, \Psi\otimes\Psi') \models C$ implies $(\sigma, \Psi\otimes\Psi') \models C'$

where C_t are the transition constraints. In $\{\!\|a\#X\|\!\}$, X is any nominal data type.

Note that the assertion part of the solution is irrelevant for constraints of kind $\{\!\|M = N\|\!\}$ and $\{\!\|a\#X\|\!\}$, and that the substitution does not affect a in $\{\!\|a\#X\|\!\}$. The constraint $\{\!\|M = N\|\!\}$ is used in the bisimulation for matching output objects, and thus must check for syntactic equivalence (cf. Definition 10). The constraint $\{\!\|a\#X\|\!\}$ is used in the bisimulation for recording what an opened name must be fresh for. This corresponds to distinctions in open bisimulation for the pi-calculus [32]. The constraint $C \Rightarrow C'$ will only be used in the definition of bisimulation. The solutions are those where, for any extension of the assertion, membership in sol(C) implies membership in sol(C'). This is to accommodate clause 3 of Definition 10. A similar extension of arbitrary assertion for the other kinds of constraint is not necessary because of Weakening. We write $\{\!\|a\#X, Y\|\!\}$ for $\{\!\|a\#X\|\!\} \wedge \{\!\|a\#Y\|\!\}$, and we extend the notation to sets of names, e.g. $\{\!\|\widetilde{a}\#X\|\!\}$.

Before we can give the definition of symbolic bisimulation we need to define a symbolic variant of the concrete static equivalence. Given a constraint C, two processes are symbolically statically equivalent if they are statically equivalent for all solutions of the constraint.

Definition 15 (Symbolic static equivalence)**.** Two processes P and Q are *statically equivalent* for C, written $P \simeq_C Q$, if for each $(\sigma, \Psi) \in \text{sol}(C)$ we have that $\Psi\otimes\mathcal{F}(P)\sigma \simeq \Psi\otimes\mathcal{F}(Q)\sigma$.

We now have everything we need to define symbolic bisimulation. This definition follows the corresponding one in [21] closely.

Definition 16 ((Early) Symbolic bisimulation)**.** A *symbolic bisimulation* $\mathcal{S}$ is a ternary relation between constraints and pairs of agents such that $\mathcal{S}(C, P, Q)$ implies all of

(1) $P \simeq_C Q$, and
(2) $\mathcal{S}(C, Q, P)$, and
(3) If $P \xrightarrow[C_P]{\tau} P'$ then there exists a set of constraints $\widehat{C}$ such that $C \wedge C_P \Leftrightarrow \bigvee \widehat{C}$ and for all $C' \in \widehat{C}$ there exists Q' and C_Q such that
   (a) $Q \xrightarrow[C_Q]{\tau} Q'$, and
   (b) $C' \Rightarrow C_Q$, and
   (c) $\mathcal{S}(C', P', Q')$
(4) If $P \xrightarrow[C_P]{y(x)} P'$, $x\#(P, Q, C, C_P, y)$ and $y\#(P, Q, C)$ then there exists $\widehat{C}$ such that $C \wedge C_P \Leftrightarrow \bigvee \widehat{C}$ and for all $C' \in \widehat{C}$ there exists Q' and C_Q such that

(a) $Q \xrightarrow[C_Q]{\underline{y}(x)} Q'$, and
(b) $C' \Rightarrow C_Q$, and
(c) $\mathcal{S}(C', P', Q')$

(5) If $P \xrightarrow[C_P]{\bar{y}\,(\nu\tilde{a})N} P'$, $\tilde{a}\#(P, Q, C, C_P, y)$ and $y\#(P, Q, C)$ then there exists $\widehat{C}$ such that $C \wedge C_P \wedge \{\!|\tilde{a}\#P, Q|\!\} \Leftrightarrow \bigvee \widehat{C}$ and for all $C' \in \widehat{C}$ there exists Q' and C_Q such that
(a) $Q \xrightarrow[C_Q]{\bar{y}\,(\nu\tilde{a})N'} Q'$, and
(b) $C' \Rightarrow C_Q \wedge \{\!|N = N'|\!\}$, and
(c) $\mathcal{S}(C', P', Q')$

We write $P \sim_s Q$ if $\mathcal{S}(\mathbf{true}, P, Q)$ for some symbolic bisimulation $\mathcal{S}$, and say that P is *symbolically bisimilar* to Q.

The set $\widehat{C}$ allows a case analysis on the constraint solutions, as exemplified in the next section. The output objects need to be equal in a solution to C'. Since the solutions of $\{\!|N = N'|\!\}$ only depend on the substitutions, this constraint corresponds to the fact that the objects must be identical in the concrete bisimulation. Note that $\mathrm{bn}(\alpha)$ may occur in $\widehat{C}$. Based on [10,26], we conjecture that analogously to the case there, adding the requirement $\mathrm{bn}(\alpha)\#\widehat{C}$ would give late symbolic bisimulation.

Note that in [23], the case analysis is defined with an implication, e.g. $C \wedge C_P \Rightarrow \bigvee \widehat{C}$, while here it is defined with equivalence, $C \wedge C_P \Leftrightarrow \bigvee \widehat{C}$. This is related to the completeness of the algorithm, as explained in Section 7.1.

In Section 6 we will show the correspondence between the symbolic and concrete semantics and equivalences.

3.2. Examples

We now look at a few examples to illustrate the concrete and symbolic transitions and bisimulations. First consider a simple example from the pi-calculus. This can be expressed as a psi-calculus: let the only data terms be names, the only assertion be $\mathbf{1}$, the conditions be equality and inequality tests on names, and entailment defined by $\forall a.\mathbf{1} \vdash a = a$, $\forall a, b : a \neq b.\mathbf{1} \vdash a \neq b$ and $\forall a.\mathbf{1} \vdash a \leftrightarrow a$. For a more thorough discussion, see [7]. We use $\tau\,.\,P$ as a shorthand for $(\nu b)(\bar{b}b\,.\,\mathbf{0} \mid \underline{b}(b)\,.\,P)$ for some $b\#P$. In the following examples we drop a trailing $.\,\mathbf{0}$. Consider the two agents P_1 and Q_1:

$$P_1 = a(x)\,.\,P_1' \quad \text{where } P_1' = \tau\,.\,\bar{a}\,b$$
$$Q_1 = a(x)\,.\,Q_1' \quad \text{where } Q_1' = (\mathbf{case}\; x = b : \tau\,.\,\bar{a}\,b \;\; [] \;\; x \neq b : \tau\,.\,\bar{a}\,b)$$

These are bisimilar. A concrete bisimulation between these agents is

$$\{(\mathbf{1}, P_1, Q_1)\} \cup \bigcup_{n \in \mathcal{N}} \{(\mathbf{1}, P_1', Q_1'[x := n])\} \cup \{(\mathbf{1}, \bar{a}\,b, \bar{a}\,b)\}$$

The bisimulation needs to be infinite because of the infinite branching in the input. In contrast, a symbolic bisimulation only contains four triples:

$$\Big\{(\mathbf{true}, P_1, Q_1),\;\; (\mathbf{true}, P_1', Q_1'),\;\; (\{\!|\mathbf{1} \vdash x = b|\!\}, \bar{a}\,b, \bar{a}\,b),\;\; (\{\!|\mathbf{1} \vdash x \neq b|\!\}, \bar{a}\,b, \bar{a}\,b)\Big\}$$

When checking the second triple $(\mathbf{true}, P_1', Q_1')$, the transition of P_1' is matched by a case analysis: $\widehat{C}$ in the definition of symbolic bisimulation (Definition 16) is $\{\{\!|\mathbf{1} \vdash x = b|\!\}, \{\!|\mathbf{1} \vdash x \neq b|\!\}\}$, and a matching transition for Q_1' can be found for each of these cases, so the agents are bisimilar. In contrast, they are not equivalent in the incomplete symbolic bisimulations in [13,17].

In general, there is an additional reason for a psi-calculus bisimulation relation to be infinite: Definition 10 requires extension of arbitrary assertions. In the example psi-calculus above, the only assertion is $\mathbf{1}$ and thus the only source of infiniteness is the input action.

Next we look at an example where we have tuples of channels and projection, e.g. the entailment relation gives us that $\mathbf{1} \vdash \mathrm{first}(M, N) \leftrightarrow M$. Consider the agent

$$R = \overline{M}\,N\,.\,R'$$

Concretely this agent has infinitely many transitions even in an empty frame: $R \xrightarrow{\overline{M}\,N} R'$, and equivalent actions $\overline{\mathrm{first}(M, K)}\,N$ for all K, and $\overline{\mathrm{first}(\mathrm{first}(M, L), K)}\,N$ for all L and K, etc. Symbolically, however, it has only one transition: $R \xrightarrow[\{\!|\mathbf{1} \vdash M \leftrightarrow y|\!\}]{\bar{y}\,N} R'$.

For another example, consider the two agents

$$P_2 = \overline{F}\,N\,.\,P' \qquad Q_2 = \mathbf{0}$$

where F is a term such that for no Ψ, M does it hold that $\Psi \vdash F \leftrightarrow M$, i.e. F is not a channel. Then we have that P_2 and Q_2 are concretely bisimilar since neither one of them has a transition. But symbolically P_2 has the transition $P_2 \xrightarrow[\{\!|\mathbf{1} \vdash F \leftrightarrow y|\!\}]{\bar{y}N} P'$, while Q_2 has no symbolic transition. Perhaps surprisingly they are still symbolically bisimilar: Definition 16 requires that we find a disjunction $\widehat{C}$ such that $C \wedge C_P \Leftrightarrow \bigvee \widehat{C}$, or in this case such that $\mathbf{true} \wedge \{\!|\mathbf{1} \vdash F \leftrightarrow y|\!\} \Leftrightarrow \bigvee \widehat{C}$. Since F is not channel equivalent to anything, the left hand side has no solutions, which means that an empty $\widehat{C}$ will do, since $\bigvee \emptyset = \mathbf{false}$. The condition "for all $C' \in \widehat{C}$" in the definition becomes trivially true, so Q_2 does not have to mimic the transition.

4. Weak semantics and bisimulation

Weak bisimulation $\dot{\approx}$ for psi-calculi is introduced in [24], and in this section we briefly recapitulate some key ideas and definitions. Since we adopt Weakening as a requisite we can use the definition which in that paper is notated $\underset{\text{smp}}{\dot{\approx}}$.

As is standard, we define weak bisimulation by adjusting Definition 10 (strong bisimulation) so that τ actions can be inserted or removed when simulating a transition. We define weak transitions in the usual way:

Definition 17 (Weak transitions)**.**

- $\Psi \rhd P \Longrightarrow P$
- if $\Psi \rhd P \xrightarrow{\tau} P''$ and $\Psi \rhd P'' \Longrightarrow P'$ then $\Psi \rhd P \Longrightarrow P'$
- if $\Psi \rhd P \Longrightarrow P''$, $\Psi \rhd P'' \xrightarrow{\alpha} P'''$, and $\Psi \rhd P''' \Longrightarrow P'$ then $\Psi \rhd P \overset{\alpha}{\Longrightarrow} P'$, where $\alpha = \tau$ or $\alpha = \overline{M}\,(\nu\tilde{a})N$.

Note that we do not define a weak version of input transitions. In the late semantics, the definition becomes unnecessarily complex, and instead we choose to spell out the transitions in Definition 19 below. We first define static implication, which is simply a one-directional static equivalence.

Definition 18 (Static implication)**.** P statically implies Q in Ψ, written $P \leq_\Psi Q$, means that $\forall\varphi.\ \Psi \otimes \mathcal{F}(P) \vdash \varphi \Rightarrow \Psi \otimes \mathcal{F}(Q) \vdash \varphi$. We write $P \leq Q$ for $P \leq_{\mathbf{1}} Q$.

Definition 19 (Weak bisimulation)**.** A *weak bisimulation* $\mathcal{R}$ is a ternary relation between assertions and pairs of agents such that $\mathcal{R}(\Psi, P, Q)$ implies all of

(1) Weak static implication: There exists Q' such that $\Psi \rhd Q \Longrightarrow Q'$ and $P \leq_\Psi Q'$ and $\mathcal{R}(\Psi, P, Q')$.
(2) Symmetry: $\mathcal{R}(\Psi, Q, P)$
(3) Extension of arbitrary assertion: $\forall\Psi'.\ \mathcal{R}(\Psi \otimes \Psi', P, Q)$
(4) Weak simulation: for all α, P' such that $\mathrm{bn}(\alpha)\#(\Psi, Q)$ and $\Psi \rhd P \xrightarrow{\alpha} P'$ it holds

$$
\begin{aligned}
&\text{if } \alpha = \tau: && \exists Q'.\ \Psi \rhd Q \Longrightarrow Q' \wedge \mathcal{R}(\Psi, P', Q')\\
&\text{if } \alpha = \overline{M}\,(\nu\tilde{a})N: && \exists Q'.\ \Psi \rhd Q \overset{\alpha}{\Longrightarrow} Q' \wedge \mathcal{R}(\Psi, P', Q')\\
&\text{if } \alpha = \underline{M}(x): && \forall L \exists Q''', Q'', Q'.\\
& && \quad \Psi \rhd Q \Longrightarrow Q''' \wedge\\
& && \quad \Psi \rhd Q''' \xrightarrow{\alpha} Q'' \wedge\\
& && \quad \Psi \rhd Q''[x := L] \Longrightarrow Q' \wedge\\
& && \quad \mathcal{R}(\Psi, P'[x := L], Q')
\end{aligned}
$$

We define $P \dot{\approx}_\Psi Q$ to mean that there exists a weak bisimulation $\mathcal{R}$ such that $\mathcal{R}(\Psi, P, Q)$, and write $P \dot{\approx} Q$ for $P \dot{\approx}_{\mathbf{1}} Q$.

Comparing to strong bisimulation (Definition 10), clause 1 in the definition, that P and Q are statically equivalent, is adjusted so that if P can make conditions true, then Q can make them true possibly after performing some τ actions. Clauses 2 and 3 are unchanged. Clause 4 (simulation) is split in three parts. If the action α to be simulated is τ then Q should simulate by doing zero or more τ actions. If it is an output or an input action then Q simulates by doing an arbitrary number of τ actions before and after the action.

One point which may not be immediately obvious is Clause 1, weak static implication, where the conjunct $\mathcal{R}(\Psi, P, Q')$ may be surprising. It states that Q must evolve to a Q' that is statically implied by P, and also bisimilar to P. This last

requirement may seem unnecessarily strong, but in fact without it the resulting weak bisimulation equivalence would not be preserved by the parallel operator. See [24] for examples and further motivation.

In [24] this bisimulation is called *simple* weak bisimulation, and is defined with the early semantics. To ensure that all results in [24] also hold for the definition in the present paper, we have the following result:

Lemma 20. *A relation is a weak bisimulation according to Definition 19 precisely if it is a simple weak bisimulation according to [24].*

The proof is straightforward using Lemma 11 (see [22]).

Note that weak bisimulation is not preserved by the **case** construct. The reasoning is analogous to why weak bisimulation is not preserved by the operator $+$ in CCS or the pi-calculus: $\tau\,.\,\mathbf{0} \dot{\approx} \mathbf{0}$ but $a\,.\,\mathbf{0} + \tau\,.\,\mathbf{0} \not\dot{\approx} a\,.\,\mathbf{0} + \mathbf{0}$. If the left-hand process does its τ action, the right-hand can only simulate by standing still. In the next step, the right-hand can do the action a which the left-hand can no longer simulate. This problem is solved in a standard way: in the simulation clause of bisimulation where $\alpha = \tau$, Q must simulate the τ action made by P with a τ chain containing at least one τ action.

Weak bisimulation is also not preserved by input prefixes, again for the same reason as in the pi-calculus. Closing the relation under substitution in the same way as is done for strong bisimulation leads to the definition of weak congruence, denoted $\approx$.

Definition 21 (Weak congruence)**.** P and Q are weakly τ-bisimilar, written $\Psi \rhd P \underset{\text{tau}}{\dot{\approx}} Q$, if $P \dot{\approx}_\Psi Q$ and they also satisfy *weak congruence simulation*:
for all P' such that $\Psi \rhd P \xrightarrow{\tau} P'$ it holds:

$$\exists Q'.\ \Psi \rhd Q \overset{\tau}{\Rightarrow} Q' \ \wedge\ P' \dot{\approx}_\Psi Q'$$

and similarly with the roles of P and Q exchanged. We define $P \approx Q$ to mean that for all Ψ, and for all sequences of substitutions σ it holds that $\Psi \rhd P\sigma \underset{\text{tau}}{\dot{\approx}} Q\sigma$.

5. Weak symbolic semantics and bisimulation

In this section we define the weak symbolic semantics and bisimulation. We begin by defining weak symbolic transitions, similarly to how we defined weak transitions in Definition 17.

Definition 22 (Weak symbolic transitions)**.**

- $\Psi \rhd P \underset{\mathbf{true}}{\Longrightarrow} P$
- if $\Psi \rhd P \underset{C}{\overset{\tau}{\longrightarrow}} P'' \ \wedge\ \Psi \rhd P'' \underset{C'}{\Longrightarrow} P'$ then $\Psi \rhd P \underset{C \wedge C'}{\Longrightarrow} P'$
- if $\Psi \rhd P \underset{C}{\Longrightarrow} P'' \wedge \Psi \rhd P'' \underset{C'}{\overset{\alpha}{\longrightarrow}} P''' \wedge \Psi \rhd P''' \underset{C''}{\Longrightarrow} P'$
  then $\Psi \rhd P \underset{C \wedge C' \wedge C''}{\overset{\alpha}{\Longrightarrow}} P'$

The constraint of a weak transition is simply the conjunction of the individual steps of the transition. Note in passing that here also the weak input transition is straightforward.

We also need the symbolic counterpart to static implication. A process statically implies another symbolically for a constraint C if it statically implies it for all solutions of C.

Definition 23 (Symbolic static implication)**.** A process P *statically implies* another process Q *symbolically* for C, written $P \leq_C Q$, if for each $(\sigma, \Psi) \in \mathrm{sol}(C)$ we have that $P\sigma \leq_\Psi Q\sigma$.

The weak symbolic bisimulation is a straight-forward modification of the strong symbolic bisimulation (Definition 16), with the addition of Clause 1 matching the one of weak (non-symbolic) bisimulation (Definition 19).

Definition 24 (Weak symbolic bisimulation)**.** A *weak symbolic bisimulation* $\mathcal{S}$ is a ternary relation between constraints and pairs of agents such that $\mathcal{S}(C, P, Q)$ implies all of

(1) there exists a set of constraints $\widehat{C}$ such that $C \Leftrightarrow \bigvee \widehat{C}$ and for all $C' \in \widehat{C}$ there exists Q' and C_Q such that
  (a) $Q \underset{C_Q}{\Longrightarrow} Q'$,
  (b) $C' \Rightarrow C_Q$,

(c) $P \leq_{C'} Q'$, and
(d) $\mathcal{S}(C', P, Q')$

(2) $\mathcal{S}(C, Q, P)$, and
(3) If $P \xrightarrow[C_P]{\tau} P'$ then there exists $\widehat{C}$ such that $C \wedge C_P \Leftrightarrow \bigvee \widehat{C}$ and for all $C' \in \widehat{C}$ there exists Q' and C_Q such that
(a) $Q \xRightarrow[C_Q]{} Q'$, and
(b) $C' \Rightarrow C_Q$, and
(c) $\mathcal{S}(C', P', Q')$
(4) If $P \xrightarrow[C_P]{y(x)} P'$, $x\#(P, Q, C, C_P, y)$ and $y\#(P, Q, C)$ then there exists $\widehat{C}$ such that $C \wedge C_P \Leftrightarrow \bigvee \widehat{C}$ and for all $C' \in \widehat{C}$ there exists Q' and C_Q such that
(a) $Q \xRightarrow[C_Q]{\underline{y}(x)} Q'$, and
(b) $C' \Rightarrow C_Q$, and
(c) $\mathcal{S}(C', P', Q')$
(5) If $P \xrightarrow[C_P]{\bar{y}\,(\nu\tilde{a})N} P'$, $\tilde{a}\#(P, Q, C, C_P, y)$ and $y\#(P, Q, C)$ then there exists $\widehat{C}$ such that $C \wedge C_P \wedge \{\!|\tilde{a}\#P, Q|\!\} \Leftrightarrow \bigvee \widehat{C}$ and for all $C' \in \widehat{C}$ there exists Q' and C_Q such that
(a) $Q \xRightarrow[C_Q]{\bar{y}\,(\nu\tilde{a})N'} Q'$, and
(b) $C' \Rightarrow C_Q \wedge \{\!|N = N'|\!\}$, and
(c) $\mathcal{S}(C', P', Q')$

We write $P \dot{\approx}^S_C Q$ if $\mathcal{S}(C, P, Q)$ for some symbolic bisimulation $\mathcal{S}$. We write $P \dot{\approx}^S Q$ for $P \dot{\approx}^S_{\mathbf{true}} Q$, and say that P is *symbolically bisimilar* to Q.

We also define the symbolic counterpart to weak congruence:

Definition 25 (Symbolic weak congruence). P and Q are symbolic weakly τ-bisimilar, written $P \dot{\approx}^S_{C\,\text{tau}} Q$, if $P \dot{\approx}^S_C Q$ and they also satisfy *weak congruence simulation*:
If $P \xrightarrow[C_P]{\tau} P'$ then there exists $\widehat{C}$ such that $C \wedge C_P \Leftrightarrow \bigvee \widehat{C}$ and for all $C' \in \widehat{C}$ there exists Q' and C_Q such that

(1) $Q \xRightarrow[C_Q]{\tau} Q'$, and
(2) $C' \Rightarrow C_Q$, and
(3) $P' \dot{\approx}^S_{C'} Q'$

and similarly with the roles of P and Q exchanged. We define $P \approx^s Q$ to mean $P \dot{\approx}^S_{C\,\text{tau}} Q$.

6. Full abstraction

In this section we show that the symbolic and concrete equivalences coincide in both the strong and the weak case, but first we make precise the form of the concrete agents we consider.

For relating the concrete equivalences presented in Sections 2 and 4 with the symbolic equivalences presented in Sections 3 and 5 it is convenient to write a concrete agent as $P\sigma$, where σ is a possibly empty finite sequence of substitutions denoting the values received so far by P. Since it is always possible to α-convert the input object to be fresh for previously received values we only consider sequences of substitutions with the following property:

$$\text{if } [x_1 := L_1] \ldots [x_n := L_n] \text{ is a substitution sequence then } x_i \# L_j \text{ for } j < i. \tag{1}$$

Two other properties of sequences of substitutions are:

$$\text{if } [x_1 := L_1] \ldots [x_n := L_n] \text{ is a substitution sequence then } x_i \# L_j \text{ for } j > i. \tag{2}$$
$$\text{if } [x_1 := L_1] \ldots [x_n := L_n] \text{ is a substitution sequence then } x_i \# x_j \text{ for } j \neq i. \tag{3}$$

Property (2) states that a name in the domain of the sequence does not occur in the range of the rest of the sequence, while property (3) states that a name in the domain only occurs once in the domain of the sequence. These properties are used in the proofs of the results in this section.

We assume that *sch* is injective, and we impose five additional requirements on the substitution function. They are needed in various proofs, and all of them are natural. We require substitution to be homomorphic on *sch* and if X is a member of a nominal set additionally that:

$$\begin{aligned} X[x := x] &= X \\ x[x := M] &= M \\ X[x := M] &= X \text{ if } x\#X \\ X[x := L][y := M] &= X[y := M][x := L] \text{ if } x\#y, M \text{ and } y\#L \end{aligned}$$

Definition 26 (Interference free)**.** A substitution sequence is *interference free* if it has properties 1, 2, and 3.

Properties 2 and 3 can be derived for concrete agents:

Lemma 27. *Let $P\sigma$ be an agent where σ has property 1. Then there exists a permutation p and an interference free sequence of substitutions σ' such that $(p \cdot P)\sigma' = P\sigma$.*

Proof. The proof is by induction on the length of σ. See [25]. □

The point of this lemma is that whenever we encounter a substitution applied to an agent, we can assume that the substitution is interference-free by applying a suitable permutation. The substitutions sequences are generated either by substituting arbitrarily chosen names (representing subjects) or names in input prefixes. With the help of Lemma 27 the latter can be alpha-converted so that the only sequences we ever need consider are interference free.

We now turn to showing that the concrete and symbolic equivalences coincide. We define substitution on symbolic actions by $\tau[\tilde{z} := \tilde{M}] = \tau$, $(\underline{y}(x))[\tilde{z} := \tilde{M}] = \underline{y[\tilde{z} := \tilde{M}]}(x[\tilde{z} := \tilde{M}])$, and $(\overline{y}\ (\nu\tilde{a})N)[\tilde{z} := \tilde{M}] = \overline{y[\tilde{z} := \tilde{M}]}\ (\nu\tilde{a})N[\tilde{z} := \tilde{M}]$, where $x, \tilde{a}\#\tilde{M}, \tilde{z}$.

The following two lemmas show the operational correspondence between the symbolic semantics and the concrete semantics: given a symbolic transition where the transition constraint has a solution, there is always a corresponding concrete transition (Lemma 28) and vice versa (Lemma 30).

Lemma 28 (Correspondence symbolic-concrete)**.**

(1) If $P \xrightarrow[C]{\underline{y}(x)} P'$, $(\sigma, \Psi) \models C$, *and* $x\#\sigma$ *then* $\Psi \rhd P\sigma \xrightarrow{(\underline{y}(x))\sigma} P'\sigma$.

(2) If $P \xrightarrow[C]{\overline{y}\ (\nu\tilde{a})N} P'$, $(\sigma, \Psi) \models C$, *and* $\tilde{a}\#\sigma$ *then* $\Psi \rhd P\sigma \xrightarrow{(\overline{y}\ (\nu\tilde{a})N)\sigma} P'\sigma$.

(3) If $P \xrightarrow[C]{\tau} P'$ *and* $(\sigma, \Psi) \models C$ *then* $\Psi \rhd P\sigma \xrightarrow{\tau} P'\sigma$.

Proof. The proof is by induction on the length of the derivation of the transition. See [25]. □

Lemma 29 (Weak correspondence symbolic-concrete)**.**

(1) If $P \underset{C}{\Longrightarrow} P'$ *and* $(\sigma, \Psi) \models C$ *then* $\Psi \rhd P\sigma \Rightarrow P'\sigma$.

(2) If $P \underset{C'_P}{\Longrightarrow} P'', P'' \xrightarrow[C''_P]{\underline{y}(x)} P''', P''' \underset{C'''_P}{\Longrightarrow} P'$, $(\sigma[x := L], \Psi) \models C'_P \wedge C''_P \wedge C'''_P$, *and* $x\#\sigma, P, y$ *then* $\Psi \rhd P\sigma \Rightarrow P''\sigma$, $\Psi \rhd P''\sigma \xrightarrow{(\underline{y}(x))\sigma} P'''\sigma$, *and* $\Psi \rhd P'''\sigma[x := L] \Rightarrow P'\sigma[x := L]$.

(3) If $P \xrightarrow[C]{\overline{y}\ (\nu\tilde{a})N} P'$, $(\sigma, \Psi) \models C$, *and* $\tilde{a}\#\sigma, P, y$ *then* $\Psi \rhd P\sigma \xrightarrow{(\overline{y}\ (\nu\tilde{a})N)\sigma} P'\sigma$.

(4) If $P \underset{C}{\overset{\tau}{\Rightarrow}} P'$, $(\sigma, \Psi) \models C$ *then* $\Psi \rhd P\sigma \overset{\tau}{\Rightarrow} P'\sigma$.

Proof. The proof is by induction on the length of the transition. See [25]. □

Lemma 30 (Correspondence concrete-symbolic)**.**

(1) If $\Psi \rhd P\sigma \xrightarrow{\underline{M}(x)} P'\sigma$, $y\#\Psi, P, \sigma, x$, *where* $x\#\sigma$, P *then there exists* C *such that* $P \xrightarrow[C]{\underline{y}(x)} P'$ *and* $(\sigma[y := M], \Psi) \models C$.

(2) *If* $\Psi \rhd P\sigma \xrightarrow{\overline{M}\,(\nu\tilde{a})N\sigma} P'\sigma$, $y\#\Psi, P, \sigma, \tilde{a}$, *and* $\tilde{a}\#\sigma, P$ *then there exists* C *such that* $P \xrightarrow[C]{\bar{y}\,(\nu\tilde{a})N} P'$ *and* $(\sigma[y := M], \Psi) \models C$.

(3) *If* $\Psi \rhd P\sigma \xrightarrow{\tau} P'\sigma$ *then there exists* C *such that* $P \xrightarrow[C]{\tau} P'$ *and* $(\sigma, \Psi) \models C$.

Proof. The proofs are by induction over the transition derivation (one case for each rule); for the details see [25]. □

Lemma 31 (Weak correspondence concrete-symbolic)**.**

(1) *If* $\Psi \rhd P\sigma \Rightarrow P'\sigma$ *then there exists* C *such that* $P \underset{C}{\Rightarrow} P'$ *and* $(\sigma, \Psi) \models C$.

(2) *If* $\Psi \rhd P\sigma \Rightarrow P''\sigma, \Psi \rhd P''\sigma \xrightarrow{(\underline{y}(x))\sigma} P'''\sigma, P'''\sigma[x := L] \Rightarrow P'\sigma[x := L]$, $y\#\Psi, P, x$, *and* $x\#\sigma, P$ *then there exists* C *such that* $P \underset{C}{\overset{\underline{y}(x)}{\Longrightarrow}} P'$ *and* $(\sigma[x := L], \Psi) \models C$.

(3) *If* $\Psi \rhd P\sigma \xRightarrow{(\bar{y}\,(\nu\tilde{a})N)\sigma} P'\sigma$, $y\#\Psi, P, \tilde{a}$, *and* $\tilde{a}\#\sigma, P$ *then there exists* C *such that* $P \xRightarrow[C]{\bar{y}\,(\nu\tilde{a})N} P'$ *and* $(\sigma, \Psi) \models C$.

(4) *If* $\Psi \rhd P\sigma \overset{\tau}{\Rightarrow} P'\sigma$ *then there exists* C *such that* $P \underset{C}{\overset{\tau}{\Rightarrow}} P'$ *and* $(\sigma, \Psi) \models C$.

Proof. The proof is by induction on the length of the transition. See [25] for details. □

Theorem 32 (Soundness (strong))**.** *Assume* $\mathcal{S}$ *is a symbolic bisimulation and let* $\mathcal{R} = \{(\Psi, P\sigma, Q\sigma) : \exists C.(\sigma, \Psi) \models C$ *and* $(C, P, Q) \in \mathcal{S}\}$. *Then* $\mathcal{R}$ *is a concrete bisimulation.*

The full proof is in [25]. The proof idea to show that $\mathcal{R}$ is a concrete bisimulation is to assume $(\Psi, P\sigma, Q\sigma) \in \mathcal{R}$ and that $P\sigma$ has a transition in environment Ψ. We use Lemma 30 to find a symbolic transition from P, then the fact that $\mathcal{S}$ is a symbolic bisimulation to find a simulating symbolic transition from Q, and finally Lemma 28 to find the required concrete transitions from $Q\sigma$.

Similarly to [21] we need an extra assumption about the expressiveness of constraints. Say that the constraints **Cstr** are *bisimulation complete* if for all $\mathcal{R}, P, Q, \Psi, \sigma$ such that $\mathcal{R}$ is a concrete bisimulation there exists a constraint C such that $(\Psi, \sigma) \models C \iff (\Psi, P\sigma, Q\sigma) \in \mathcal{R}$. This property will be used in the proof of Theorem 33 below. In order to determine symbolic bisimilarity in an efficient way we need to compute this constraint, which is easy for the pi-calculus [10,26,27] and harder (but in many practical cases possible) for cryptographic signatures [11]. These results suggest that our constraints are sufficiently expressive, but for other instances of psi-calculi we may have to extend the constraint language. We leave this as an area of further research.

Theorem 33 (Completeness (strong))**.** *Let* $\mathcal{R}$ *be a concrete bisimulation and define* $\mathcal{S} = \{(C, P, Q) : (\sigma, \Psi) \models C$ *implies* $(\Psi, P\sigma, Q\sigma) \in \mathcal{R}\}$. *If the constraints* ***Cstr*** *are bisimulation complete then* $\mathcal{S}$ *is a symbolic bisimulation.*

The full proof is in [25]. The proof idea is the converse of the proof for Theorem 32. The expressiveness assumption of constraints mentioned above is needed in order to construct the disjunction of constraints in the symbolic bisimulation. From these two theorems we get:

Corollary 34 (Full abstraction (strong))**.** $P \sim Q$ *if and only if* $P \sim_s Q$.

We now turn to showing the correspondence between weak bisimulations.

Theorem 35 (Full abstraction (weak))**.** $P \approx Q$ *if and only if* $P \approx^s Q$.

Proof. Soundness is proved by showing that if $\mathcal{S}$ is a weak symbolic congruence then $\mathcal{R} = \{(\Psi, P\sigma, Q\sigma) : \exists C.(\sigma, \Psi) \models C$ and $\mathcal{S}(C, P, Q)\}$ is a weak concrete congruence.

Completeness is proved by showing that if $\mathcal{R}$ is a weak concrete congruence then $\mathcal{S} = \{(C, P, Q) : (\sigma, \Psi) \models C$ implies $\mathcal{R}(\Psi, P\sigma, Q\sigma)\}$ is a weak symbolic congruence.

The details are found in [25]. □

7. The bisimulation algorithm and its correctness

In Figs. 1–3 we present an algorithm that computes a constraint C such that $P \overset{\cdot}{\approx}{}^s_C Q$ and a witnessing bisimulation. The algorithm, adapted from [21], does a depth-first search of the underlying symbolic transition graph. The main function

```
/* bisim(P, Q)
   P and Q are agents. Returns a pair (C, T) where C is a constraint such
   that P ≁_C Q and T is a table describing a witnessing bisimulation. */

bisim(P, Q) = close(P, Q, true, ∅)

/* close(P, Q, C, W)
   P and Q are agents, C represents the constraints seen so far, and W is
   a set of pairs of agents that have already been visited by the algorithm.
   Returns a pair (C', T), where C' is a constraint necessary for P and Q to
   be bisimilar, and T is a table describing a partial witnessing bisimulation.
   */

close(P, Q, C, W)
   if (P, Q) ∈ W then
      (true, ∅)
   else let (C_stimp, T_stimp) = match-stimp(P, Q, C, W)
            (C'_stimp, T'_stimp) = match-stimp(Q, P, C, W)
            (C_τ, T_τ) = match-τ(P, Q, C, W)
            (C'_τ, T'_τ) = match-τ(Q, P, C, W)
            (C_out, T_out) = match-out(P, Q, C, W)
            (C'_out, T'_out) = match-out(Q, P, C, W)
            (C_in, T_in) = match-in(P, Q, C, W)
            (C'_in, T'_in) = match-in(Q, P, C, W)
        in (C_stimp ∧ C'_stimp ∧ C_τ ∧ C'_τ ∧ C_out ∧ C'_out ∧ C_in ∧ C'_in,
            T_stimp ⊔ T'_stimp ⊔ T_τ ⊔ T'_τ ⊔ T_out ⊔ T'_out ⊔ T_in ⊔ T'_in ⊔
            {(P, Q) ↦ {C ∧ C_stimp ∧ C_τ ∧ C_out ∧ C_in}} ⊔
            {(Q, P) ↦ {C ∧ C'_stimp ∧ C'_τ ∧ C'_out ∧ C'_in}})
```

Fig. 1. `bisim` and `close` functions.

`bisim(`P, Q`)` calls `close(`P, Q, **true**, $\emptyset$`)`. The first two arguments to `close(`P, Q, C, W`)` are the current agents being compared, the third argument are the constraints accumulated so far, which are used to construct a witnessing bisimulation, and the fourth argument contains the pairs of agents that have already been compared.

The function `close` calls `match-stimp(`P, Q, C, W`)`, `match-`τ `(`P, Q, C, W`)`, `match-out(`P, Q, C, W`)`, and `match-in (`P, Q, C, W`)` in order to compute the constraints for static implication and matching τ, output, and input actions respectively. The function `match-stimp(`P, Q, C, W`)` computes a constraint for which P statically implies Q, and a table that represents a witnessing bisimulation. The other functions compute a constraint for which Q simulates P, and a table of the witnessing bisimulation. The conjunction of these constraints is then returned as a constraint for which P and Q are bisimilar. These functions correspond to the different clauses in the definition of weak bisimulation.

The algorithm assumes the presence of yet another type of constraint, $F \preceq G$. This is used to capture static implication, and its solutions are all pairs (σ, Ψ) such that $\forall\varphi.\Psi\otimes(F\sigma) \vdash \varphi \Leftrightarrow \Psi\otimes(G\sigma) \vdash \varphi$. We define $\bigwedge \emptyset = \mathbf{true}$ and $\bigvee \emptyset = \mathbf{false}$.

A table is a finite function from pairs of agents to constraints. We write $T \sqcup T'$ for the union of T and T' defined only when $dom(T) \cap dom(T') = \emptyset$. We thus have that $(T \sqcup T')(P, Q) = T(P, Q)$ if $(P, Q) \in dom(T)$ and $(T \sqcup T')(P, Q) = T'(P, Q)$ if $(P, Q) \in dom(T')$. The operator $\sqcup$ is used instead of $\cup$ in the algorithm since we work under the assumption that the graphs are finite with a tree structure.

When choosing a fresh y in functions `match-out` and `match-in`, the function `newName` is used. It picks a name that is fresh for all its arguments. The argument $\mathcal{X}$ is the set of names that have been used as subjects previously in the algorithm, and as a side effect, the newly choosen y is added to this set. This ensures that all such y are globally unique.

Note that nowhere in the algorithm the constraints are checked for consistency, they are only accumulated. This means that the algorithm can be run also on psi-instances that are equipped with an undecidable entailment relation. However in these cases it might be difficult to interpret the result.

7.1. Correctness

We now turn to show the correctness of the algorithm, and we follow [21] closely. Similarly to [21] we here restrict ourselves to finite symbolic transition graphs, i.e. they are finitely branching and have a finite number of nodes. Like [21], for simplicity we also assume that the graphs have a tree structure. Since every finite graph can be expanded to an equivalent tree, this restriction is not too limiting.

We first look at termination of the algorithm. For each recursive call to `close`, the parameter W is increased by a pair (P, Q) not already in W. For this reason, since we assume that the symbolic transition graphs are finite, eventually the test $(P, Q) \in W$ will be true, and the recursion stops. Hence we have the following lemma:

Lemma 36 (Termination). *If the symbolic transition graphs of P and Q are finite then* `bisim(`P, Q`)` *terminates.*

We now state the soundness and completeness theorems for the algorithm:

```
/* match-stimp(P, Q, C, W)
   The parameters are as in close(P, Q, C, W). Returns a pair (C', T) where
   C' is a constraint that is necessary for P to statically imply Q, and T is
   a table describing a partial witnessing bisimulation. */

match-stimp(P, Q, C, W)
   let Qtr = {(C_Qi, Q_i) : Q ==(C_Qi)==> Q_i}
       (C̃, T̃) = map λ(C_Qi, Q_i).
                    let (C_i, T_i) =
                        close(P, Q_i, C ∧ C_Qi, W ∪ {(P, Q)})
                    in (C_Qi ∧ C_i ∧ (C_i ∧ C_Qi ⇒ F(P) ≼ F(Q_i)), T_i)
                Qtr
   in (true ⇒ ⋁C̃, ⨆T̃)

/* match-τ(P, Q, C, W)
   The parameters are as in close(P, Q, C, W). Returns a pair (C', T) where
   C' is a constraint that is necessary for Q to simulate P for τ-actions,
   and T is a table describing a partial witnessing bisimulation. */

match-τ(P, Q, C, W)
    let Ptr = {(C_Pi, P_i) : P -τ->(C_Pi) P_i}
        Qtr = {(C_Qj, Q_j) : Q ==(C_Qj)==> Q_j}
        (C̃, T̃) = map (λ(C_Pi, P_i).
                    let (C̃_i, T̃_i) = map (λ(C_Qj, Q_j) .
                        let (C_ij, T_ij) =
                            close(P_i, Q_j, C ∧ C_Pi ∧ C_Qj, W ∪ {(P, Q)})
                        in (C_Qj ∧ C_ij, T_ij)) Qtr
                    in (C_Pi ⇒ ⋁C̃_i, ⨆T̃_i)) Ptr
    in (⋀C̃, ⨆T̃)
```

Fig. 2. `match-stimp` and `match-τ` functions.

Theorem 37 (Soundness of the algorithm). *If* $C = \text{bisim}(P, Q)$ *then* $P \dot{\approx}_C Q$.

Theorem 38 (Completeness of the algorithm). *If* $P \dot{\approx}_C Q$ *and* $\text{bisim}(P, Q, C_m, T)$ *then* $C \Rightarrow C_m$.

Proof sketch: These theorems are proven by defining an invariant that is an approximation of bisimulation, and then showing that the functions `close` and `match-*` maintain this invariant. The proofs follow [21] closely and are found in [25].

The completeness proof only works if the case analysis in the definition of symbolic bisimulation is defined with equality, e.g. $C \wedge C_P \Leftrightarrow \bigvee \widehat{C}$, instead of implication, $C \wedge C_P \Rightarrow \bigvee \widehat{C}$, while the other proofs work with either definition. This is the reason the definition of symbolic bisimulation has changed in this respect, compared to [23]. The same problem manifests itself in [21], where the completeness proof of the algorithm uses a definition with equality instead of implication [28], not found in the paper. The correct definition is given in a subsequent paper by the same authors [20].

7.2. Extensions

To compute the congruence of Definition 25 the algorithm in Fig. 4 is used. It calculates a constraint C such that $P \dot{\approx}^s_{C,\text{tau}} Q$. The additions are straight-forward. The function `match-τ-strict` is essentially the same as `match-τ`, but requires Q to simulate with at least one τ-transition. The function `τ-bisim` captures the two requirements of Definition 25 that the agents should be bisimilar and that they satisfy weak congruence simulation (`match-τ-strict`).

The algorithm can easily be made to compute strong bisimulation instead of weak. The modification is to change `match-τ`, `match-in`, and `match-out` to use strong transitions instead, remove the call to `match-stimp` in function `close`, and replace C_{stimp} in `close` with $\mathcal{F}(P) \preceq \mathcal{F}(Q)$, and C'_{stimp} with $\mathcal{F}(Q) \preceq \mathcal{F}(P)$.

8. Conclusion and future work

We have defined a symbolic operational semantics for psi-calculi and both strong and weak symbolic bisimulations which are fully abstract with regards to the original semantics. While the developments in [7,24] give meta-theory for a wide range of calculi of mobile processes with nominal data and logic, the work presented in this paper gives a solid foundation for automated tools for the analysis of systems modelled in such calculi.

As mentioned in the introduction, the purity of the original semantics of psi-calculi has made the symbolic semantics easier to develop. There are no structural equivalence rules, which are a complication in the applied pi-calculus. The scope opening rule is because of this straight-forward which makes knowledge representation simpler than in spi-calculi, and the bisimulation less complex. Nevertheless, the technical challenges have not been absent: the precise design of the constraints

```
/* match-out(P,Q,C,W)
   The parameters are as in close(P,Q,C,W). Returns a pair (C',T) where
   C' is a constraint that is necessary for Q to simulate P for outputs, and
   T is a table describing a partial witnessing bisimulation. */

match-out(P, Q, C, W)
  let Ptr = {(ȳ(νã)N, C_Pi, P_i) : P --ȳ(νã)N, C_Pi--> P_i
                                  ∧ y = newName(P,Q,C,X) ∧ ã#P,Q,C,C_Pi,y}
      (C̃, T̃) = map λ(ȳ(νã)N, C_Pi, P_i).
    let Qtr = {(z̄(νc̃)N', C_Qj, Q_j) : Q ==z̄(νc̃)N', C_Qj==> Q_j
                                  ∧ y = z ∧ ã = c̃}
        (C̃_i, T̃_i) = map λ(z̄(νc̃)N', C_Qj, Q_j)
      let (C_ij, T_ij) =
        close(P_i, Q_j, C ∧ C_Pi ∧ C_Qj ∧ {|N = N'|} ∧ {|ã#P,Q|}, W ∪ {(P,Q)})
      in (C_Qj ∧ {|N = N'|} ∧ C_ij, T_ij) Qtr
    in (C_Pi ∧ {|ã#P,Q|} ⇒ ⋁C̃_i, ⨆T̃_i) Ptr
  in (⋀C̃, ⨆T̃)

/* match-in(P,Q,C,W)
   The parameters are as in close(P,Q,C,W). Returns a pair (C',T) where
   C' is a constraint that is necessary for Q to simulate P for inputs, and
   T is a table describing a witnessing bisimulation. */

match-in(P, Q, C, W)
  let Ptr = {(y(x), C_Pi, P_i) : P --y(x), C_Pi--> P_i
                                  ∧ y = newName(P,Q,C,X) ∧ x#P,Q,C,C_Pi,y}
      (C̃, T̃) = map λ(y(x), C_Pi, P_i).
          let Qtr = {(z(x'), C_Qj, Q_j) : Q ==z(x'), C_Qj==> Q_j
                                  ∧ y = z ∧ x = x'}
              (C̃_i, T̃_i) = map λ(z(x'), C_Qj, Q_j) .
                  let (C_ij, T_ij) =
                     close(P_i, Q_j, C ∧ C_Pi ∧ C_Qj, W ∪ {(P,Q)})
                  in (C_Qj ∧ C_ij, T_ij) Qtr
          in (C_Pi ⇒ ⋁C̃_i, ⨆T̃_i) Ptr
  in (⋀C̃, ⨆T̃)
```

Fig. 3. `match-out` and `match-in` functions.

```
/* τ-bisim(P,Q)
   P and Q are agents. Returns a pair (C,T) where C is a constraint such
   that P ≈̇^s_{C,tau} Q and T is a table describing a witnessing bisimulation. */

τ-bisim(P, Q)
    let (C, T) = bisim(P, Q)
        (C_τ, T_τ) = match-τ-strict(P, Q)
        (C'_τ, T'_τ) = match-τ-strict(Q, P)
    in (C ∧ C_τ ∧ C'_τ, T ∪ T_τ ∪ T_τ)

/* match-τ-strict(P,Q)
   The parameters are as in close(P,Q) Returns a pair (C',T) where C'
   is a constraint that is necessary for Q to simulate P with at least one
   τ-action, and T is a table describing a witnessing bisimulation. */

match-τ-strict(P, Q)
    let Ptr = {(P, C_Pi, P_i) : P --τ, C_Pi--> P_i}
        Qtr = {(Q, C_Qj, Q_j) : Q ==τ, C_Qj==> Q_j}
        (C̃, T̃) = map (λ(P, C_Pi, P_i).
                    let (C̃_i, T̃_i) = map (λ(Q, C_Qj, Q_j) .
                          let (C_ij, T_ij) = close(P_i, Q_j, C_Pi ∧ C_Qj, ∅)
                          in (C_Qj ∧ C_ij, T_ij)) Qtr
                    in (C_Pi ⇒ ⋁C̃_i, ⨆T̃_i)) Ptr
    in (⋀C̃, ⨆T̃)
```

Fig. 4. Congruence algorithm.

and their solution has been delicate. Since assertions may occur under a prefix, the environment can change after a transition. Keeping the assertion Ψ in the transition constraints (on the form $(\nu\widetilde{a})\{\!|\Psi \vdash \varphi|\!\}$) essentially keeps a snapshot of the environment that gives rise to the transition. An alternative would be to use time stamps to keep track of which environment made which condition true, but that approach seems more difficult. It is also worth mentioning the freshness constraints, $\{\!|a\#X|\!\}$. They solve the problem of keeping track of which names have been opened in the bisimulation. Since this does not need to be part of the partitioning in the bisimulation, another approach is to make this another parameter of the bisimulation as done in [29], but since freshness constraints fit nicely into our formalism we chose this solution.

The original psi-calculi admit pattern matching in inputs. In a symbolic semantics this would lead to complications in the COM-rule, which should introduce a substitution for the names bound in the pattern. This means introducing more fresh names and constraints, and it is not clear that the convenience of pattern matching outweighs such an awkward semantic rule. We leave this as an area for further study.

The algorithm amounts to a straightforward adaptation of the algorithm in [21]. A tool which implements the bisimulation algorithm is currently in development [19]. The algorithm in itself only computes a constraint under which agents are equivalent. In order to decide whether the constraint is satisfiable, a constraint solver for the parameters of the particular psi-calculus is needed. Work is ongoing to create a generic constraint solver for psi-calculi with parameters that form free algebras, and we intend to interface constraint solvers developed for specific application domains (e.g. security). We will also produce mechanized proofs of the adequacy of the symbolic semantics, using the Isabelle theorem prover.

When typing schemes have been developed for psi-calculi, a natural progression would be to take advantage of those also in the symbolic semantics, to further constrain the possible values and thus the size of state spaces.

References

[1] M. Abadi, B. Blanchet, Analyzing security protocols with secrecy types and logic programs, J. ACM 52 (1) (2005) 102–146.
[2] M. Abadi, C. Fournet, Mobile values, new names, and secure communication, in: Proceedings of POPL '01, ACM, 2001, pp. 104–115.
[3] M. Abadi, A.D. Gordon, A calculus for cryptographic protocols: the spi calculus, J. Inform. Comput. 148 (1) (1999) 1–70.
[4] J. Bengtson, Formalising process calculi, Ph.D. thesis, Uppsala University, 2010.
[5] J. Bengtson, J. Parrow, Psi-calculi in Isabelle, in: S. Berghofer, T. Nipkow, C. Urban, M. Wenzel (Eds.), Proceedings of TPHOLs 2009, LNCS, vol. 5674, Springer, 2009, pp. 99–114.
[6] J. Bengtson, M. Johansson, J. Parrow, B. Victor, Psi-calculi: mobile processes, nominal data, and logic, in: Proceedings of LICS 2009, IEEE Computer Society Press, 2009, pp. 39–48.
[7] J. Bengtson, M. Johansson, J. Parrow, B. Victor, Psi-calculi: a framework for mobile processes with nominal data and logic, Logical Methods Comput. Sci. 7 (1:11) (2011) 1–44 (this is an extended version of [6]).
[8] B. Blanchet, An efficient cryptographic protocol verifier based on prolog rules, in: CSFW'01: Proceedings of the 14th IEEE Computer Security Foundations Workshop, 2001, pp. 82.
[9] B. Blanchet, M. Abadi, C. Fournet, Automated verification of selected equivalences for security protocols, J. Logic Algebraic Program. 75 (1) (2008) 3–51.
[10] M. Boreale, R. De Nicola, A symbolic semantics for the π-calculus, J. Inform. Comput. 126 (1) (1996) 34–52 (available as Report SI 94 RR 04, Università "La Sapienza" di Roma; an extended abstract appeared in Proceedings of CONCUR '94, pp. 299–314, LNCS 836).
[11] J. Borgström, Equivalences and calculi for formal verification of cryptographic protocols, Ph.D. thesis, EPFL, Lausanne, 2008.
[12] J. Borgström, U. Nestmann, On bisimulations for the spi calculus, in: H. Kirchner, C. Ringeissen (Eds.), Proceedings of AMAST 2002, LNCS, vol. 2422, Springer, 2002, pp. 287–303.
[13] J. Borgström, S. Briais, U. Nestmann, Symbolic bisimulation in the spi calculus, in: Proceedings of CONCUR 2004, LNCS, vol. 3170, Springer, 2004, pp. 161–176.
[14] S. Briais, Theory and tool support for the formal verification of cryptographic protocols, Ph.D. thesis, EPFL, Lausanne, 2008.
[15] M.G. Buscemi, U. Montanari, Open bisimulation for the concurrent constraint pi-calculus, in: S. Drossopoulou (Ed.), Proceedings of ESOP 2008, LNCS, vol. 4960, Springer, 2008, pp. 254–268.
[16] S. Delaune, S. Kremer, M. Ryan, Symbolic bisimulation for the applied pi calculus, in: V. Arvind, S. Prasad (Eds.), Proceedings of FSTTCS 2007, Lecture Notes in Computer Science, vol. 4855, Springer, 2007, pp. 133–145.
[17] S. Delaune, S. Kremer, M.D. Ryan, Symbolic bisimulation for the applied pi calculus, J. Comput. Security 18 (2) (2010) 317–377, doi:10.3233/JCS-2010-0363.
[18] M. Gabbay, A. Pitts, A new approach to abstract syntax with variable binding, Formal Aspects Comput. 13 (2001) 341–363.
[19] R. Gutkovas, Exercising psi-calculi: a psi-calculi workbench, M.Sc. thesis, Department of Information Technology, Uppsala University, 2011. Available from: <http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-157883>.
[20] M. Hennessy, H. Lin, Proof systems for message-passing process algebras, in: Proceedings of the 4th International Conference on Concurrency Theory, CONCUR '93, Springer-Verlag, London, UK, 1993, pp. 202–216.
[21] M. Hennessy, H. Lin, Symbolic bisimulations, Theoret. Comput. Sci. 138 (2) (1995) 353–389 (earlier version published as Technical Report 1/92, School of Cognitive and Computing Sciences, University of Sussex, UK).
[22] M. Johansson, Psi-calculi: a framework for mobile process calculi, Ph.D. thesis, Uppsala University, 2010.
[23] M. Johansson, B. Victor, J. Parrow, A fully abstract symbolic semantics for psi-calculi, in: Proceedings of SOS 2009, 2009, pp. 17–31.
[24] M. Johansson, J. Bengtson, J. Parrow, B. Victor, Weak equivalences in psi-calculi, in: Proceedings of LICS 2010, IEEE Computer Society Press, 2010, pp. 322–331.
[25] M. Johansson, B. Victor, J. Parrow, Computing strong and weak bisimulations for psi-calculi – with proofs, Tech. Rep. 2011-018, Department of Information Technology, Uppsala University, 2011. Available from: <http://www.it.uu.se/research/publications/reports/2011-018>.
[26] H. Lin, Symbolic transition graph with assignment, in: U. Montanari, V. Sassone (Eds.), Proceedings of CONCUR '96, LNCS, vol. 1119, Springer, 1996, pp. 50–65.
[27] H. Lin, Computing bisimulations for finite-control pi-calculus, J. Comput. Sci. Technol. 15 (1) (2000) 1–9.
[28] H. Lin, Personal communication, 2011.
[29] J. Liu, H. Lin, A complete symbolic bisimulation for full applied pi calculus, in: J. van Leeuwen, A. Muscholl, D. Peleg, J. Pokorný, B. Rumpe (Eds.), SOFSEM 2010: Theory and Practice of Computer Science, Lecture Notes in Computer Science, vol. 5901, Springer, Berlin/Heidelberg, 2010, pp. 552–563.
[30] J. Liu, H. Lin, Proof system for applied pi calculus, in: C. Calude, V. Sassone (Eds.), Theoretical Computer Science, IFIP Advances in Information and Communication Technology, vol. 323, Springer, Boston, 2010, pp. 229–243.
[31] A.M. Pitts, Nominal logic, a first order theory of names and binding, Inform. Comput. 186 (2003) 165–193.
[32] D. Sangiorgi, A theory of bisimulation for the π-calculus, Acta Informatica 33 (1996) 69–97 (earlier version published as Report ECS-LFCS-93-270, University of Edinburgh. An extended abstract appeared in the Proceedings of CONCUR '93, LNCS 715).
[33] B. Victor, F. Moller, The Mobility Workbench – a tool for the π-calculus, in: D. Dill (Ed.), Proceedings of FSTTCS 2007, Lecture Notes in Computer Science, vol. 4855, Springer, 1994, pp. 428–440.