Algorithms for Observational Equivalence

Stéphanie Delaune

LSV, ENS Cachan & CNRS & INRIA Saclay Île-de-France

Summary. Observational equivalence is crucial when specifying privacy-type properties like vote-privacy that states that an observer cannot see the difference when A is talking and when B is talking. The goal of this report is to summarize the advances we have made during the project on developing algorithms to verify this kind of security properties in presence of an active attacker.

In the previous report (Deliverable 2.1), we only focus on the indistinguishability problem between two sequences of messages generated by the protocol. Thus, we do not take into account the dynamic behaviour of the underlying protocol. Here, we consider an active attacker who may interact with the protocol. To the best of our knowledge, the only tool allowing one to analyse equivalence-based properties in presence of an active attacker is ProVerif [4]. However the tool has several limitations that prevent us to establish the equivalences needed to establish privacy on most of our examples.

First, we propose a technique for expanding the scope of the ProVerif tool (see Section 1). Then, following the constraint solving approach, we show how to reduce observational equivalence to symbolic equivalence of constraint systems (Section 2). For a large class of processes, this allows us to conclude by using an existing decision procedure by M. Baudet [1,2]. In Section 3, we propose an alternative procedure (to decide symbolic equivalence of pairs of constraint systems) that is more amenable to an implementation.

This report summarizes the following papers:

1. S. Delaune, M. D. Ryan and B. Smyth. *Automatic verification of privacy properties in the applied pi-calculus.* In Proceedings of the 2nd Joint iTrust and PST Conferences on Privacy, Trust Management and Security (IFIPTM'08), Trondheim, Norway, June 2008, IFIP Conference Proceedings 263, pages 263-278. Springer. → **see Section 1**

2. V. Cortier and S. Delaune. *A method for proving observational equivalence.* In Proceedings of the 22nd IEEE Computer Security Foundations Symposium (CSF'09), Port Jefferson, NY, USA, July 2009, pages 266-276. IEEE Computer Society Press.
3. S. Delaune, S. Kremer and M. D. Ryan. *Symbolic bisimulation for the applied pi calculus.* Journal of Computer Security 18(2), pages 317-377, 2010. → **see Section 2**

4. V. Cheval, H. Comon-Lundh and S. Delaune. *Automating security analysis: symbolic equivalence of constraint systems.* In Proceedings of the 5th International Joint Conference on Automated Reasoning (IJCAR'10), Edinburgh,

Scotland, UK, July 2010, LNAI 6173, pages 412-426. Springer-Verlag.
→ **see Section 3**

1 Going beyond with the ProVerif tool

We develop a formal verification technique for proving observational equivalence of cryptographic protocols. We focus on proving observational equivalence between processes P and Q having the same structure and differing only in the choice of terms. The ProVerif tool makes some progress in this direction. However, the method developed for proving observational equivalence is not complete and is unable to prove privacy-type properties as those needed to analyse electronic voting protocols.

In [14], we have expanded the scope of ProVerif, to provide reasoning about further equivalences. We have developed an algorithm to enable automated reasoning. Using this approach, we provided the first automated proof that the electronic voting protocol by Fujioka, Okamoto, and Ohta [15] satisfies privacy.

1.1 Some extensions

To overcome the limitations that prevent us to analyse several protocols using ProVerif, we have extended the applied pi calculus with strong phases and data swapping.

Strong phases. Many protocols can be broken into phases, and their security properties can be formulated in terms of these phases. Typically, for instance, if a protocol discloses a session key after the conclusion of a session, then the secrecy of the data exchanged during the session may be compromised but not its authenticity. To enable modelling of protocols with several phases, the syntax of processes is supplemented with a phase prefix "$\texttt{phase}\ t; P$". Intuitively t represents a global clock, and the process "$\texttt{phase}\ t; P$" is active only during phase t.

However, in order to model electronic voting protocols, we need to consider global synchronisation. We have to express that the registration is closed before the voting phase starts. To achieve this, the syntax of processes is supplemented with a strong phase prefix "$\texttt{strong phase}\ t; P$". A strong phase represents a global synchronisation and t represents the global clock. The process $\texttt{strong phase}\ t; P$ is active only during strong phase t and a strong phase progression may only occur once all the instructions under the previous phase have been executed.

Example 1. The process $\texttt{strong phase}\ 1; \texttt{out}(c, a) \mid \texttt{strong phase}\ 2; \texttt{out}(c, b)$ cannot output b without having previously output a. Note that this behaviour is possible in presence of a "weak" phase.

Data swapping. The equivalence $\mathtt{out}(c,a) \mid \mathtt{out}(c,b) \approx_\ell \mathtt{out}(c,b) \mid \mathtt{out}(c,a)$ holds trivially since the processes are in fact structurally equivalent. But the corresponding bi-process $P = \mathtt{out}(c, \mathrm{choice}[a,b]) \mid \mathtt{out}(c, \mathrm{choice}[b,a])$ does not satisfy diff-equivalence and therefore the equivalence cannot be proved by ProVerif.

Actually, we have that

- $\mathsf{fst}(P) = \mathtt{out}(c,a) \mid \mathtt{out}(c,b)$, and
- $\mathsf{snd}(P) = \mathtt{out}(c,b) \mid \mathtt{out}(c,a)$.

Since $\mathtt{out}(c,b) \mid \mathtt{out}(c,a) \equiv \mathtt{out}(c,a) \mid \mathtt{out}(c,b)$ it seems reasonable to rewrite the process $\mathsf{snd}(P)$ as $\mathtt{out}(c,a) \mid \mathtt{out}(c,b)$, enabling us to write P as

$$\mathtt{out}(c, \mathrm{choice}[a,a]) \mid \mathtt{out}(c, \mathrm{choice}[b,b]).$$

Our new bi-process satisfies diff-equivalence, and thus observational equivalence. It therefore seems possible (under certain circumstances) to *swap* values from the left to the right side of the parallel operator. Sometimes the swap is not done initially but instead immediately after a strong phase. In such a case, we have to ensure that the data used during the current phase (nonces, inputs) are passed to the next phase. This can be done by using private channel. To specify data swapping we introduce the special comment *(**swap*)* in process descriptions, which can be seen as a *proof hint*.

Our translator. To allow automated reasoning we have proposed a translator which accepts as input processes written in our extended language. It will also include a single main process and sub-processes of the form "`let` $P = Q$", subject to some restrictions. The translator outputs processes in the standard language of ProVerif, which can be automatically reasoned about by the software tool. If ProVerif is now able to establish equivalence of the resulting bi-process, this means that the bi-process P given in input is such that $\mathsf{fst}(P) \approx_\ell \mathsf{snd}(P)$.

Recently, the transformation has been revisited [22] and implemented in the ProSwapper tool [21]:

http://www.bensmyth.com/proswapper.php.

1.2 Application

In this section, we demonstrate the usefulness of our translator on the electronic voting protocol by Fujioka, Okamoto, and Ohta (FOO) (see [14] for further details).

The protocol involves voters, an administrator and a collector and it is based on blind signatures and bit commitments. The administrator is responsible for verifying that only eligible voters can cast votes and the collector handles the collecting and publishing of votes. The protocol requires three strong phases (registration, voting, tallying). The separation of the protocol into strong phases is crucial for privacy to hold.

In [12], we rely on hand proof techniques to show privacy on FOO: ProVerif is unable to prove it directly. Our modelling of FOO is similar to the one we have proposed in [12]. In addition, we have to provide a data swapping hint to allow our translator to produce an output suitable for automatic verification using ProVerif.

We use our translator to remove all instances of strong phases and handle data swapping. Our translator produces a process, which is suitable for automatic verification using ProVerif. Hence, using this approach, we provided the first automated proof that the FOO protocol satisfies privacy.

2 From observational equivalence to symbolic equivalence

The aim of this section is to provide some reduction results. More precisely, we show that observational equivalence can be reduced to the problem of checking symbolic equivalences of pairs of constraint systems. We can then conclude by using the decision procedure proposed in [1,2] for the class of subterm convergent equational theories. We have also developed our own procedure that is presented in Section 3.

2.1 Symbolic equivalence of pair of constraint systems

The following definition of constraint system is consistent with the definition of constraint system given in several papers (see [18,9]) that study trace-based security properties (*e.g.* secrecy, authentication, ...). However, we need to generalize the usual definition. Indeed, the constraint solving method presented for instance in [18,9] is complete only *w.r.t.* what an attacker can deduce, but not *w.r.t.* how it can be deduced. If an attacker has different ways to deduce a given message in two different experiments, he could distinguish between them. This is the main purpose of the variables X_i: they are used to record the recipe that has been used to deduce s_i. This information is needed to capture observational equivalence.

An *initial deducibility constraint* is either $\bot$ or consists of:

1. a frame $\phi = \{ax_1 \rhd u_1, \ldots, ax_m \rhd u_m\}$, whose size is some m (terms $u_1, \ldots, u_m$ may contain variables);
2. a sequence $D = X_1, i_1 \stackrel{?}{\vdash} s_1;\ \ldots;\ X_n, i_n \stackrel{?}{\vdash} s_n$ where
   - $X_1, \ldots, X_n$ are distinct variables, $s_1, \ldots, s_n$ are terms, and $0 \leq i_1 \leq \ldots \leq i_n \leq m$.
   - for every $0 \leq k \leq m$, $var(u_k) \subseteq \bigcup_{i_j < k} var(s_j)$;
3. a conjunction E of equations between terms.

The variables X_i represent the recipes that might be used to deduce the right-hand side of the deducibility constraint. The indices indicate which initial segment of the frame can be used. An E-*solution* of an initial deducibility constraint $\mathcal{C} = (\phi, D, E)$ consists of

– a substitution σ from $var(\{s_1, \ldots, s_n\})$ to ground terms, and
– a substitution θ mapping $X_1, \ldots, X_n$ to ground recipes, *i.e.* terms built from public function symbols, and special variables $ax_1, \ldots, ax_m$,

such that:

– for every $X_i, j \stackrel{?}{\vdash} s_i$ in D, $var(X_i\theta) \subseteq \{ax_1, \ldots, ax_j\}$ and $X_i\theta(\phi\sigma) =_{\mathsf{E}} s_i\sigma$;
– for every equation $u \stackrel{?}{=} v$ in E, we have that $u\sigma =_{\mathsf{E}} v\sigma$.

We denote by $Sol_{\mathsf{E}}(\mathcal{C})$ (or simply $Sol(\mathcal{C})$ when E is clear from the context) the set of E-solutions of $\mathcal{C}$. By convention, we have that $Sol_{\mathsf{E}}(\bot) = \emptyset$.

Intuitively, we want to check static equivalence on any possible trace. More precisely, we want that any experiment θ performed by the attacker on $\mathcal{C}$ has a counterpart on $\mathcal{C}'$ such that the resulting sequence of messages that are observed on both sides are statically equivalent. This is captured by the following definition:

Definition 1 (symbolic equivalence). *Let $\mathcal{C}$ and $\mathcal{C}'$ be two constraint systems whose corresponding frames are ϕ and ϕ'. The constraint system $\mathcal{C}$ is* symbolically equivalent *to $\mathcal{C}'$, denoted $\mathcal{C} \approx_s \mathcal{C}'$, if:*

– *for all $(\theta, \sigma) \in Sol(\mathcal{C})$, there exists σ' such that $(\theta, \sigma') \in Sol(\mathcal{C}')$, and $\phi\sigma \sim \phi'\sigma'$,*
– *for all $(\theta, \sigma') \in Sol(\mathcal{C}')$, there exists σ such that $(\theta, \sigma) \in Sol(\mathcal{C})$, and $\phi\sigma \sim \phi'\sigma'$.*

In the following section, we will see how equivalence between simple processes can be expressed via symbolic equivalence of *initial pairs of constraints*, that is a pair of the form (ϕ_1, D_1, E_1), (ϕ_2, D_2, E_2) where both initial constraint systems have the same shape. More precisely, we have that:

– $\phi_1 = \{ax_1 \rhd u_1, \ldots, ax_m \rhd u_m\}$, and $D_1 = X_1, i_1 \stackrel{?}{\vdash} s_1;\ \ldots;\ X_n, i_n \stackrel{?}{\vdash} s_n$;
– $\phi_2 = \{ax_1 \rhd v_1, \ldots, ax_m \rhd v_m\}$, and $D_2 = X_1, i_1 \stackrel{?}{\vdash} t_1;\ \ldots;\ X_n, i_n \stackrel{?}{\vdash} t_n$.

Or else it is a pair as above, in which one of the components is replaced with $\bot$.

2.2 The case of simple processes

We consider any signature and equational theory. However, we do not consider the full applied pi calculus but only a restricted fragment. For example, it is generally assumed that all communications are controlled by the attacker thus private channels between processes are not accurate (they should rather be implemented using cryptography). In addition, the attacker schedules the communications between processes thus he knows exactly to whom he is sending messages and from whom he is listening. Thus we assume that each process communicates on a personal channel.

More precisely, we consider the fragment of *simple processes* built on *basic processes* (see [10] for a formal definition of simple processes). A basic process

represents a session of a protocol role where a party waits for a message of a certain form or checks some equalities and outputs a message accordingly. Then the party waits for another message or checks for other equalities and so on. Intuitively, any protocol whose roles have a deterministic behavior can be modeled as a simple process. Most of the roles are indeed deterministic since an agent should usually exactly know what to do once he has received a message. In particular, all protocols of the J. Clark and J. Jacob library [7] can be modelled as simple processes. However, it is interesting to notice that protocols with deterministic behavior are usually not modelled within our fragment since a single channel is used for all communications. We think however that using a single channel does not provide enough information to the attacker since he is not able to schedule exactly the messages to the processes and he does not know from which process a message comes from while this information is usually available (via *e.g.* IP adresses). For example, a role emitting the constant a twice would be modelled by $P_1 = \mathtt{out}(c, a).\mathtt{out}(c, a)$ while two roles emitting each the constant a would be modeled by $P_2 = \mathtt{out}(c, a) \mid \mathtt{out}(c, a)$. Then P_1 and P_2 are observationally equivalent while the two protocols could be distinguished in practice, which is reflected in our modelling in simple processes.

In [10], we have shown that simple processes are determinate, and for determinate processes, observational equivalence and trace equivalence coincide. Hence, to decide observational equivalence for simple processes, it remains to show how to decide trace equivalence. First, we can show that trace equivalence is exactly captured by a notion of symbolic trace equivalence. Moreover, since for simple processes without replication, there are a finite number of symbolic traces, we have the following result.

Theorem 1. *[10] Let* E *be a subterm convergent equational theory. Let* A *and* B *be two simple processes without else branch nor replication. The problem whether* A *and* B *are observationally equivalent is co-NP-complete.*

The decidability of observational equivalence follows from the fact that symbolic equivalence of constraint systems is decidable for subterm convergent equational theories [1,6]. The NP-TIME decision procedure for non-observational equivalence works as follows:

- Guess a symbolic (annotated) trace tr;
- Compute (in polynomial time) the corresponding initial constraint system $\mathcal{C}$ (resp. $\mathcal{C}'$) associated to A (resp. B);
- Check whether $\mathcal{C}$ and $\mathcal{C}'$ are in symbolic equivalence.

Due to a result first proved by M. Baudet [1], we know that the last step can be done in NP-TIME for convergent subterm theories thus we deduce that the overall procedure is NP-TIME. NP-hardness is obtained using the usual encoding [20]. Recently, M. Rusinowitch and Y. Chevalier [6] proposed another decision procedure based on an extension of the small attack property. They show that, if two processes are not equivalent, then there must exist a small witness of non-equivalence. A decision of equivalence can be derived by checking every possible small witness.

2.3 The case of general processes

The class of simple processes appears to be sufficient to represent most of the cryptographic protocols. However, even if private and/or anonymous channels are not accurate, they are useful for modelling purposes. Hence, it may be interesting to go beyond this class of simple processes. In [11,13], relying on constraint systems, we have proposed a symbolic semantics for the full applied pi without replication. As in the previous approaches, by treating inputs symbolically, our semantics avoids potentially infinite branching of execution trees due to the inputs from the environment.

The semantics of the applied pi calculus is not well-suited for defining such a symbolic semantics. In particular, defining a symbolic structural equivalence which is both sound and complete seems impossible. The absence of sound and complete symbolic structural equivalence significantly complicates the proofs showing that the symbolic semantics we proposed is sound and complete *w.r.t.* the concrete one. This lead us to define a more restricted semantics which will provide an intermediate representation of applied pi calculus processes. These intermediate processes are a selected (but sufficient) subset of the original processes. One may think of them as being processes in some kind of normal form. This calculus is of independent interest. Actually, we have partly reused this intermediate calculus to obtain the result presented in Section 2.2. This intermediate calculus has also been reused by J. Liu and H. Lin [17] to propose a complete symbolic bisimulation in presence of replication.

Then, we have proposed a definition of symbolic labelled bisimilarity based on the notion of symbolic equivalence of constraint systems. Note that since the processes we consider are not simple, we can not assume that one symbolic move of a process can be mimicked by a single symbolic move in the other process. Because of this, our techniques suffers from several sources of incompleteness. For instance, we are not able to establish that the two following processes are bisimilar.

$$P_1 = \mathtt{in}(c, x).\mathtt{out}(c, a)$$
$$Q_1 = \mathtt{in}(c, x).\text{if } x = a \text{ then } \mathtt{out}(c, a) \text{ else } \mathtt{out}(c, a)$$

In our setting, the instantiation of input variables is postponed until the point at which they are actually used. This allows one to not decide the instantiation of the variable when the input has to be performed but later on. This instantiation may depend on the choice that has been made to reach that point. Hence, this prevents us to establish that the two following processes are bisimilar.

$$P_2 = \nu c_1.\mathtt{in}(c_2, x).(\mathtt{out}(c_1, b) \mid \mathtt{in}(c_1, y) \mid \text{if } x = a \text{ then } \mathtt{in}(c_1, z).\mathtt{out}(c_2, a))$$
$$Q_2 = \nu c_1.\mathtt{in}(c_2, x).(\mathtt{out}(c_1, b) \mid \mathtt{in}(c_1, y) \mid \mathtt{in}(c_1, z).\text{if } x = a \text{ then } \mathtt{out}(c_2, a))$$

Although our symbolic bisimulation is not complete, as shown above, we are able to prove observational equivalence on interesting examples. Actually, our symbolic bisimulation is sufficiently complete to deal with examples of anonymity

properties (vote-privacy, receipt-freeness, ...) arising in protocol analysis. Again, we can directly build on existing works [1,6] and obtain a decision procedure for our symbolic bisimulation for the class of subterm convergent equational theories and processes that do not contain else branches.

Related work. Recently, a relatively simple symbolic transition system and bisimulation equivalence that is fully abstract *w.r.t.* concrete bisimulation has been proposed for psi-calculi [3,16]. Psi-calculi can be more general than other proposed extensions of the pi-calculus such as the applied pi calculus that we consider here. In psi-calculi, they are helped significantly by the absence of structural equivalence rules which are rather complex in the applied pi calculus.

3 Decision procedure for symbolic equivalence

In [5], we have proposed a procedure to decide symbolic equivalence. We consider pairing, signature, symmetric and asymmetric encryptions only. Our main goal was to provide a procedure together with an efficient implementation. The main result stated in [5] is a decision procedure for symbolic equivalence of an initial pair of constraint systems.

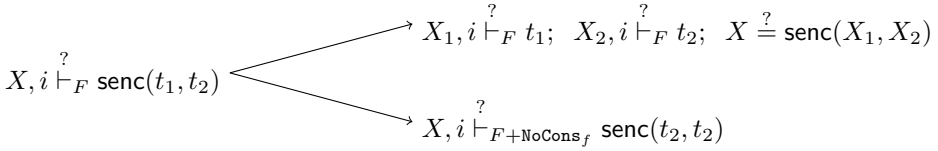
Theorem 2. *[5] Given an initial pair* $(\mathcal{C}, \mathcal{C}')$*, it is decidable whether* $\mathcal{C} \approx_s \mathcal{C}'$*.*

This result in itself is already known (*e.g.* [1,6]), but the known algorithms cannot yield any reasonable implementation. Even, the recent algorithm proposed by A. Tiu and J. Dawson [23] has not been implemented and should probably be improved to lead to a reasonable implementation. Similarly, in [6], they show that, if two processes are not equivalent, then there must exist a small witness of non-equivalence. However, the number of small witnesses is very large as all terms of size smaller than a given bound have to be considered. Consequently, neither this method nor the previous one have been implemented.

3.1 Our algorithm in a nutshell

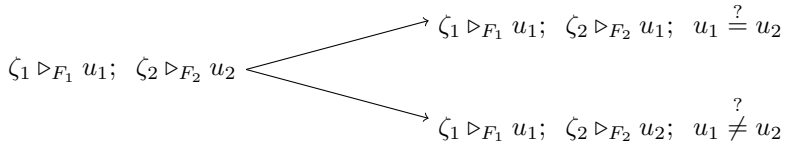
Our decision algorithm works by rewriting pairs of constraint systems that have been extended to keep track of some information (for instance, we consider equations between recipes), until a trivial failure or a trivial success is found. These rules are branching: they rewrite a pair of constraint systems into two pairs of constraint systems. Transforming the pairs of constraints therefore builds a binary tree. Termination requires to keep track of some information, that is recorded using a set F of flags attached to each deducibility constraint (*e.g.* $\mathtt{NoCons}_f, \ldots$). The flags are additional constraints that restrict the recipes. The complete set of transformation rules can be found in [5]. Below, we only present some instances of those rules to explain how the algorithm works. They are displayed for a single constraint system only.

Rule $\textsc{Cons}_{\mathsf{senc}}$:



If $\mathtt{NoCons}_{\mathsf{senc}} \notin F$ and X_1, X_2 are fresh variables.
This rule simply guesses whether the top symbol of the recipe used to deduce $\mathsf{senc}(t_1, t_2)$ is senc or not. Either it is, and then we can split the constraint, or it is not and we add a flag forbidding this.

Rule Eq-left-left:



This rule covers the comparisons that an attacker could perform at various stages. This equality rule guesses equalities between terms known by the attacker.

Rules above are displayed for a single constraint system only. We explain here how they are extended to pairs of constraint systems. If one of the constraint systems is $\bot$, then we proceed as if there was a single constraint. Otherwise, the indices i and the recipes X, ζ (resp. $X_1, X_2, \zeta_1, \zeta_2$) matching the left side of the rules *must be identical in both constraints*: we apply the rules at the same positions in both constraint systems. We have to explain now what happens when, on a given pair $(\mathcal{C}, \mathcal{C}')$ a rule can be applied on $\mathcal{C}$ and not on $\mathcal{C}'$ (or the converse). Therefore, the rule $\textsc{Cons}_{\mathsf{senc}}$, when applied to pairs of constraint systems comes in three versions: either the rule is applied on both sides or, if $X, i \overset{?}{\vdash} \mathsf{senc}(t_1, t_2)$ is in $\mathcal{C}$, and $X, i \overset{?}{\vdash} x$ is in $\mathcal{C}'$, we may apply the rule on the pair of constraint systems, adding to $\mathcal{C}'$ the equation $x \overset{?}{=} \mathsf{senc}(x_1, x_2)$ where x_1 and x_2 are fresh variables. The third version is obtained by switching $\mathcal{C}$ and $\mathcal{C}'$. Note that this may introduce new variables, that yield a termination issue. For the rules Eq-left-left, we require that at least one new non-trivial equality (or disequality) is added to one of the two constraint systems (otherwise there is a trivial loop). For all rules, if a rule is applicable on one constraint and not the other, we do perform the transformation, however replacing a constraint with $\bot$ when a condition becomes false.

Our algorithm can be stated as follows:

– Construct, from an initial pair of constraint systems $(\mathcal{C}_0, \mathcal{C}'_0)$ a tree, by applying as long as possible a transformation rule to a leaf of the tree.
– If, at some point, there is a leaf to which no rule is applicable and that is labeled $(\mathcal{C}, \bot)$ or $(\bot, \mathcal{C})$ where $\mathcal{C} \neq \bot$, then we stop with $\mathcal{C}_0 \not\approx_s \mathcal{C}'_0$.
– Otherwise, if the construction of the tree stops without reaching such a failure, return $\mathcal{C}_0 \approx_s \mathcal{C}'_0$.

Our algorithm can also be used to decide static equivalence of frames, as well as the (un)satisfiability of a constraint system. Furthermore, in case of failure, a witness of the failure can be returned, using the equations of the non-$\bot$ constraint.

Termination requires to keep track of some information, that is recorded using flags. In general the rules might not terminate. Fortunately, there is however a simple and complete strategy that avoids the non-terminating behaviours. Our transformation rules are sound: if all leaves are success leaves, then the original pair of constraint systems is equivalent. They are finally complete: if the two original constraint systems are equivalent then any pairs of constraint systems resulting from a rewriting steps are also equivalent.

Towards an automatic tool. An Ocaml implementation of the procedure described in [5], as well as several examples, are available at http://www.lsv.ens-cachan.fr/ cheval/programs/index.php (around 5000 lines of Ocaml). The implementation closely follows the transformation rules that we described. For efficiency reasons, a strategy on checking the rules applicability has been designed in addition. Note that in order to get an automatic tool for deciding equivalence of processes, we have to move from symbolic equivalence between constraint systems (or sets of constraint systems) to equivalence between processes. This requires computing all interleavings of actions, a step which could be prohibitive from the computation time point of view. Hence, in order to get an efficient procedure, it is necessary to come with some optimisations in order to reduce the search space and the number of interleavings. This problem is not specific to equivalence-based properties and has already been studied in the context of trace-based properties [8,19]. However, discarding some "symbolic" interleavings appears to be challenging for equivalence-based properties such as those studied in this chapter.

References

1. M. Baudet. Deciding security of protocols against off-line guessing attacks. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS'05)*, pages 16–25. ACM Press, 2005.
2. M. Baudet. *Sécurité des protocoles cryptographiques : aspects logiques et calculatoires*. Phd thesis, École Normale Supérieure de Cachan, France, 2007.
3. J. Bengtson, M. Johansson, J. Parrow, and B. Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science*, pages 39–48. IEEE Computer Society Press, 2009.

4. B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of the 14th Computer Security Foundations Workshop (CSFW'01)*, pages 82–96, Cape Breton (Canada), 2001. IEEE Computer Society Press.
5. V. Cheval, H. Comon-Lundh, and S. Delaune. Automating security analysis: symbolic equivalence of constraint systems. In J. Giesl and R. Haehnle, editors, *Proceedings of the 5th International Joint Conference on Automated Reasoning (IJCAR'10)*, volume 6173 of *Lecture Notes in Artificial Intelligence*, pages 412–426, Edinburgh, Scotland, UK, July 2010. Springer-Verlag.
6. Y. Chevalier and M. Rusinowitch. Decidability of symbolic equivalence of derivations. Unpublished draft, 2009.
7. J. Clark and J. Jacob. A survey of authentication protocol literature. http://www.cs.york.ac.uk/ jac/papers/drareviewps.ps. 1997.
8. E. M. Clarke, S. Jha, and W. R. Marrero. Efficient verification of security protocols using partial-order reductions. *International Journal on Software Tools for Technology Transfer*, 4(2):173–188, 2003.
9. H. Comon-Lundh, V. Cortier, and E. Zalinescu. Deciding security properties of cryptographic protocols. application to key cycles. *Transaction on Computational Logic*, 2009. To appear.
10. V. Cortier and S. Delaune. A method for proving observational equivalence. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium (CSF'09)*, pages 266–276, Port Jefferson, NY, USA, July 2009. IEEE Computer Society Press.
11. S. Delaune, S. Kremer, and M. D. Ryan. Symbolic bisimulation for the applied pi-calculus. In V. Arvind and S. Prasad, editors, *Proceedings of the 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, volume 4855 of *Lecture Notes in Computer Science*, pages 133–145, New Delhi, India, Dec. 2007. Springer.
12. S. Delaune, S. Kremer, and M. D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, July 2009.
13. S. Delaune, S. Kremer, and M. D. Ryan. Symbolic bisimulation for the applied pi calculus. *Journal of Computer Security*, 18(2):317–377, Mar. 2010.
14. S. Delaune, M. D. Ryan, and B. Smyth. Automatic verification of privacy properties in the applied pi-calculus. In Y. Karabulut, J. Mitchell, P. Herrmann, and C. D. Jensen, editors, *Proceedings of the 2nd Joint iTrust and PST Conferences on Privacy, Trust Management and Security (IFIPTM'08)*, volume 263 of *IFIP Conference Proceedings*, pages 263–278, Trondheim, Norway, June 2008. Springer.
15. A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *Advances in Cryptology – AUSCRYPT '92*, volume 718 of *Lecture Notes in Computer Science*, pages 244–251. Springer, 1992.
16. M. Johansson, B. Victor, and J. Parrow. A fully abstract symbolic semantics for psi-calculi. In *Proceedings of the 6th Workshop on Structural Operational Semantics (SOS'09)*, EPTCS, pages 17–31, 2010.
17. J. Liu and H. Lin. A complete symbolic bisimulation for full applied pi calculus. In *Proceedings of the 36th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'10)*, volume 5901 of *Lecture Notes in Computer Science*, pages 552–563. Springer, 2010.
18. J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS'01)*, pages 166–175, 2001.

19. S. Mödersheim, L. Viganò, and D. A. Basin. Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. *Journal of Computer Security*, 18(4):575–618, 2010.
20. M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions, composed keys is NP-complete. *Theoretical Computer Science*, 1-3(299):451–475, 2003.
21. B. Smyth. ProSwapper, 2010. http://www.bensmyth.com/proswapper.php.
22. B. Smyth and B. Blanchet. Automated verification of selected equivalences for synchronous cryptographic protocols. 2010.
23. A. Tiu and J. E. Dawson. Automating open bisimulation checking for the spi calculus. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium (CSF'10)*, pages 307–321, Edinburgh, United Kingdom, 2010. IEEE Computer Society Press.

Automatic verification of privacy properties in the applied pi calculus*

Stéphanie Delaune, Mark Ryan, and Ben Smyth

Abstract We develop a formal method verification technique for cryptographic protocols. We focus on proving observational equivalences of the kind $P \sim Q$, where the processes P and Q have the same structure and differ only in the choice of terms. The calculus of ProVerif, a variant of the applied pi calculus, makes some progress in this direction. We expand the scope of ProVerif, to provide reasoning about further equivalences. We also provide an extension which allows modelling of protocols which require global synchronisation. Finally we develop an algorithm to enable automated reasoning. We demonstrate the practicality of our work with two case studies.

1 Introduction

Security protocols are small distributed programs that aim to provide some security related objective over a public communications network like the Internet. Considering the increasing size of networks and their dependence on cryptographic protocols, a high level of assurance is needed in the correctness of such protocols. It is difficult to ascertain whether or not a cryptographic protocol satisfies its security requirements. Numerous protocols have appeared in literature and have subsequently been found to be flawed [13, 14, 5]. Typically, cryptographic protocols are expected to achieve their objectives in

Stéphanie Delaune
LSV, ENS Cachan & CNRS & INRIA, France, e-mail: delaune@lsv.ens-cachan.fr

Mark Ryan · Ben Smyth
School of Computer Science, University of Birmingham, UK, e-mail: {B.A.Smyth, M.D.Ryan}@cs.bham.ac.uk

* This work has been partly supported by the ARA SESUR project AVOTÉ and the EPSRC projects *Verifying anonymity and privacy properties* (EP/E040829/1) & *UbiVal* (EP/D076625/1).

the presence of an attacker that is assumed to have full control of the network (sometimes called the Dolev-Yao attacker). He can eavesdrop, replay, inject and block messages. The attacker can also modify them by performing cryptographic operations when in possession of the required keys. Furthermore the attacker may be in control of one or more of the protocol's participants. With no more than the abilities listed, and irrespective of the underlying cryptographic algorithms, numerous protocols have been found to be vulnerable to attack. Formal verification of cryptographic protocols is therefore required to ensure that cryptographic protocols can be deployed without the risk of damage.

Traditionally cryptographic protocols have been required to satisfy secrecy and authentication properties [6]. These requirements have been successfully verified by modelling them as reachability problems. Current research into applications such as electronic voting, fair exchange and trusted computing has resulted in a plethora of new requirements which protocols must satisfy (e.g. [11, 4]). Some of these properties cannot easily be expressed using traditional reachability techniques but can be written as equivalences. For example, the privacy, receipt-freeness and coercion-resistance properties of electronic voting protocols can be expressed using equivalences (see [12, 7]).

We focus on proving equivalences of the kind $P \sim Q$, where the processes P and Q have the same structure and differ only in the choice of terms. For example, the secret ballot (privacy) property of an electronic voting protocol can be expressed as

$$P(skva, v_1) \mid P(skvb, v_2) \;\sim\; P(skva, v_2) \mid P(skvb, v_1)$$

where P is the voter process with two parameters: its secret key ($skva$, $skvb$) and the candidate for whom he wish to cast their vote (here v_1, v_2). Historically many applications of equivalences to prove security requirements of cryptographic protocols have relied upon hand written proofs [12, 7]. Such proofs are time consuming and error prone. Accordingly, we direct our attention to automated techniques. The calculus developed by Blanchet *et al.* makes some progress in this direction [3]. However, the method developed for proving observational equivalence is not complete and is unable to prove certain interesting equivalences.

Contribution. We build upon [3] to provide reasoning about further equivalences (see Section 2). We also extend the syntax to allow the modelling of a new class of processes which require global synchronisation. Finally we develop an algorithm to enable automated reasoning about security requirements. The focus of our work is to model the privacy properties increasingly found in cryptographic protocols (Section 3). We demonstrate the practical application of our contribution with case studies (Sections 4 and 5). Using our approach we provide the first automated proof that the electronic voting protocol due to Fujioka, Okamoto & Ohta (FOO) [10] satisfies privacy. As a second case study we provide a formal methods proof that the Direct Anonymous Attestation (DAA) [4] protocol also satisfies privacy (the DAA authors

provided a provable security proof). An extended version of this paper [9] and our ProVerif source code are available at http://www.cs.bham.ac.uk/~bas/.

Related work. Kremer & Ryan [12] have previously demonstrated the electronic voting protocol FOO satisfies fairness, eligibility and privacy. The first two properties were verified automatically using ProVerif, and the third relied on a hand proof. Backes *et al.* [2] model a variant of DAA and provide some proofs. We observe that their model is not accurate with regards to DAA due to some subtleties in their formalisation. Nevertheless their idea of modelling synchronisation by private channel communication influenced the design of our translator.

2 Calculus of ProVerif

The process calculi of Blanchet *et al.* [3], used by the tool ProVerif, is a variant of the applied pi calculus [1], a process calculi for formally modelling concurrent systems and their interactions. In this paper we use the phrase *calculus of ProVerif* to mean the calculus defined in [3], and *ProVerif software tool* to refer to the software tool developed in accompaniment of [3].

2.1 Syntax and informal semantics

The calculus assumes an infinite set of *names* and an infinite set of *variables*. It also assumes a *signature* Σ, i.e. a finite set of *function symbols* each with an associated arity. A function symbol with arity 0 is also called a *constant*. We distinguish two categories of function symbols: *constructors* f and *destructors* g and we use h to range over both. We use standard notation for function application, i.e. $h(M_1, \ldots, M_n)$. Destructors are partial, nondeterministic operations, that processes can apply to terms. They represent primitives that can visibly succeed or fail, while constructors and the associated equational theory apply to primitives that always succeed but may return "junk". The grammar for terms/term evaluations is given below.

$M, N ::=$		term	$D ::=$		term evaluation
	a, b, c	name		M	term
	x, y, z	variable		choice$[D, D']$	choice term eval.
	choice$[M, M']$	choice term		$h(D_1, \ldots, D_n)$	function eval.
	$f(M_1, \ldots, M_n)$	constructor			

We equip the signature Σ with an *equational theory*, say E, i.e. a finite set of equations of the form $M_i = N_i$, where M_i and N_i are terms without names. The equational theory is then obtained from this set of equations by

reflexive, symmetric and transitive closure, closure by substitution of terms for variables and closure by context application. We write $M =_{\mathsf{E}} N$ (resp. $M \neq_{\mathsf{E}} N$) for equality (resp. inequality) modulo E.

Processes are built up in a similar way to processes in the pi calculus, except that messages can contain terms/term evaluations (rather than just names). In the grammar described below, M and N are terms, D is a term evaluation, a is a name, x a variable and t an integer.

$P, Q, R ::=$	processes
$null$	null process
$P \mid Q$	parallel composition
$!P$	replication
new $a; P$	name restriction
let $x = D$ in P else Q	term evaluation
$\mathrm{in}(M, x); P$	message input
$\mathrm{out}(M, N); P$	message output
phase $t; P$	weak phase

We note that the ProVerif software tool allows the definition of a single main process which in turn may refer to subprocesss of the form "let $P = Q$." The tool also permits the use of comments in the form *(* comment *)*.

The choice operator allows us to model a pair of processes which have the same structure and differ only in the choice of terms and terms evaluations. We call such a pair of processes a *biprocess*. Given a biprocess P, we define two processes $\mathsf{fst}(P)$ and $\mathsf{snd}(P)$ as follows: $\mathsf{fst}(P)$ is obtained by replacing all occurrences of $\mathrm{choice}[M, M']$ with M and $\mathrm{choice}[D, D']$ with D in P. Similarly, $\mathsf{snd}(P)$ is obtained by replacing $\mathrm{choice}[M, M']$ with M' and $\mathrm{choice}[D, D']$ with D' in P. We define $\mathsf{fst}(D)$, $\mathsf{fst}(M)$, $\mathsf{snd}(D)$ and $\mathsf{snd}(M)$ similarly.

As usual, names and variables have scopes, which are delimited by restrictions and by inputs. We write $fv(P)$, $bv(P)$ (resp. $fn(P)$ and $bn(P)$) for the sets of free and bound variables (resp. names) in P. A process is *closed* if it has no free variables (but it may contain free names). A *context* $C[\_]$ is a process with a hole. We obtain $C[P]$ as the result of filling $C[\_]$'s hole with P. An *evaluation context* C is a closed context built from $[\_]$, $C \mid P$, $P \mid C$ and new $a; C$. We sometimes refer to contexts without choice as *plain contexts*.

The major difference between the syntax of the applied pi calculus and the calculus of ProVerif, is the introduction of the choice operator. In addition there are some minor changes. For instance, communication is permitted on arbitrary terms, not just names. Function symbols are supplemented with destructors. Active substitutions are removed in favour of term evaluations. The syntax does not include the conditional "if $M = N$ then P else Q", which can be defined as "let $x = equals(M, N)$ in P else Q" where $x \notin fv(P)$ and $equals$ is a destructor with the equation $equals(x, x) = x$. We omit "else Q" when the process Q is $null$. Finally the calculus of ProVerif does not rely on a sort system. We believe that processes written in the calculus of

ProVerif, can be mapped to semantically equivalent processes in the applied pi calculus and vice-versa, although proving this remains an open problem. This can easily be extended to biprocesses.

2.2 Operational semantics

The operational semantics of processes in the calculus of ProVerif, are defined by three relations, namely *term evaluation* $\Downarrow$, *structural equivalence* $\equiv$ and *reduction* $\rightarrow$. Structural equivalence and reductions are only defined on closed processes. We write $\rightarrow^*$ for the reflexive and transitive closure of $\rightarrow$, and $\rightarrow^*\equiv$ for its union with $\equiv$. The operational semantics for the calculus of ProVerif differ in minor ways from the semantics of the applied pi calculus. *Structural equivalence* is the smallest equivalence relation on processes that is closed under application of evaluation contexts and some other standard rules such as associativity and commutativity of the parallel operator. *Reduction* is the smallest relation on biprocesses closed under structural equivalence and application of evaluation contexts such that

RED I/O	$\mathrm{out}(N, M); Q \mid \mathrm{in}(N', x); P \rightarrow Q \mid P\{^{M}/_{x}\}$ if $\mathsf{fst}(N) = \mathsf{fst}(N')$ and $\mathsf{snd}(N) = \mathsf{snd}(N')$
RED FUN 1	$\mathrm{let}\ x = D\ \mathrm{in}\ P\ \mathrm{else}\ Q \rightarrow P\{^{\mathrm{choice}[M_1, M_2]}/_{x}\}$ if $\mathsf{fst}(D) \Downarrow M_1$ and $\mathsf{snd}(D) \Downarrow M_2$
RED FUN 2	$\mathrm{let}\ x = D\ \mathrm{in}\ P\ \mathrm{else}\ Q \rightarrow Q$ if there is no M_1 such that $\mathsf{fst}(D) \Downarrow M_1$ and there is no M_2 such that $\mathsf{snd}(D) \Downarrow M_2$
RED REPL	$!P \rightarrow P \mid !P$

2.3 Extension to processes with weak phases

Many protocols can be broken into phases, and their security properties can be formulated in terms of these phases. Typically, for instance, if a protocol discloses a session key after the conclusion of a session, then the secrecy of the data exchanged during the session may be compromised but not its authenticity. To enable modelling of protocols with several phases the calculus of ProVerif is extended [3]. The syntax of processes is supplemented with a phase prefix "phase $t; P$", where t is a non-negative integer. Intuitively, t represents a global clock, and the process "phase $t; P$" is active only during phase t. However, it is possible that *not* all instructions of a particular phase are executed prior to a phase transition. Moreover, parallel processes may only communicate if they are under the same phase.

Example 1. Let $P = \text{phase } 1; \text{out}(c, a) \mid \text{phase } 2; \text{out}(c, b)$. The process P can output b without having first output a.

The semantics of processes are extended to deal with weak phases (see [3]).

2.4 Observational equivalence

The notion of observational equivalence was introduced by Abadi & Fournet [1], subsequently Blanchet, Abadi & Fournet [3] defined strong observational equivalence. This paper will use strong observational equivalence, henceforth we shall use observational equivalence to mean strong observational equivalence. We first recall the standard definition of observational equivalence. We write $P \downarrow_M$ when P emits a message on the channel M, that is, when $P \equiv C[\text{out}(M', N); R]$ for some evaluation context $C[\_]$ that does not bind $fn(M)$ and $M =_{\mathsf{E}} M'$.

Definition 1 ([3]). *Observational equivalence* $\sim$ is the largest symmetric relation $\mathcal{R}$ on closed processes such that $P \mathcal{R} Q$ implies:

1. if $P \downarrow_M$ then $Q \downarrow_M$;
2. if $P \to P'$ then there exists Q' such that $Q \to Q'$ and $P' \mathcal{R} Q'$;
3. $C[P] \mathcal{R} C[Q]$ for all evaluation contexts C.

Intuitively, a context may represent an attacker, and two processes are observationally equivalent if they cannot be distinguished by any attacker. Given a biprocess P, we say that P satisfies observational equivalence when we have that $\mathsf{fst}(P) \sim \mathsf{snd}(P)$.

A reduction $P \to Q$ for a biprocess P implies the corresponding processes have reductions $\mathsf{fst}(P) \to \mathsf{fst}(Q)$ and $\mathsf{snd}(P) \to \mathsf{snd}(Q)$. However, reductions in $\mathsf{fst}(P)$ and $\mathsf{snd}(P)$ do not necessarily correspond to any biprocess reduction. When such a corresponding reduction does exist the processes $\mathsf{fst}(P)$ and $\mathsf{snd}(P)$ satisfy uniformity under reduction (UUR):

Definition 2 ([3]). A biprocess P satisfies *uniformity under reduction* if:

1. $\mathsf{fst}(P) \to Q_1$ implies that $P \to Q$ for some biprocess Q with $\mathsf{fst}(Q) \equiv Q_1$, and symmetrically for $\mathsf{snd}(P) \to Q_2$;
2. for all plain evaluation contexts C, for all biprocess Q, $C[P] \to Q$ implies that Q satisfies UUR.

Blanchet *et al.* [3] have shown that if a biprocess P satisfies uniformity under reductions then P satisfies observational equivalence. The ProVerif software automatically verifies whether its input satisfies uniformity under reduction and thus enables us to prove observational equivalence in some cases.

2.5 Limitations of the calculus

There are trivial equivalences (see Example 2 described below) which the calculus of ProVerif is unable to prove since the definition of observational equivalence by uniformity under reductions is too strong. We overcome this problem with *data swapping*.

Example 2. The equivalence $\mathrm{out}(c, a) \mid \mathrm{out}(c, b) \sim \mathrm{out}(c, b) \mid \mathrm{out}(c, a)$ holds trivially since the processes are in fact structurally equivalent. But the corresponding biprocess $\mathrm{out}(c, \mathrm{choice}[a, b]) \mid \mathrm{out}(c, \mathrm{choice}[b, a])$ does not satisfy uniformity under reductions and therefore the equivalence cannot be proved by ProVerif.

Moreover, the phase semantics introduced by the calculus of ProVerif [3] are insufficient to model protocols which require synchronisation, as the phase semantics do not enforce that all instances of a phase must be completed prior to phase progression. We solve this problem with the introduction of *strong phases*.

Both of these problems are encountered when modelling cryptographic protocols from literature. As case studies we demonstrate the suitability of our approach by modelling the privacy properties of the electronic voting protocol FOO [10] and Direct Anonymous Attestation (DAA) [4].

3 Extending the calculus

To overcome the limitations stated in the previous section, we extend the calculus with strong phases and data swapping.

3.1 Extension to processes with strong phases

Similarly to weak phases the syntax of processes is supplemented with a strong phase prefix "strong phase $t; P$", where t is a non-negative integer. A strong phase represents a global synchronisation and t represents the global clock. The process strong phase $t; P$ is active only during strong phase t and a strong phase progression may only occur once all the instructions under the previous phase have been executed.

Example 3. Consider our earlier example (Example 1) with the use of strong phase. Now, the process

$$\text{strong phase } 1; \mathrm{out}(c, a) \mid \text{strong phase } 2; \mathrm{out}(c, b)$$

cannot output b without having previously output a.

3.2 Extension to processes with data swapping

Let us first consider the background to our approach. Referring back to Example 2 we recall the biprocess $Q = \text{out}(c, \text{choice}[a, b]) \mid \text{out}(c, \text{choice}[b, a])$ which does not satisfy UUR. We note that $\mathsf{fst}(Q) = \text{out}(c, a) \mid \text{out}(c, b)$ and $\mathsf{snd}(Q) = \text{out}(c, b) \mid \text{out}(c, a)$. Since $\text{out}(c, b) \mid \text{out}(c, a) \equiv \text{out}(c, a) \mid \text{out}(c, b)$ it seems reasonable to rewrite $\mathsf{snd}(Q)$ as $\text{out}(c, a) \mid \text{out}(c, b)$, enabling us to write Q as $\text{out}(c, \text{choice}[a, a]) \mid \text{out}(c, \text{choice}[b, b])$ which is semantically equivalent to $\text{out}(c, a) \mid \text{out}(c, b)$. Our new biprocess satisfies uniformity under reduction, and thus observational equivalence. It therefore seems possible (under certain circumstances) to *swap* values from the left to the right side of the parallel operator. Sometimes the swap is not done initially but instead immediately after a strong phase. To specify data swapping we introduce the special comment *(**swap*)* in process descriptions, which can be seen as a *proof hint*. Returning to our example, we would rewrite Q as

$$\begin{aligned} Q' &= \textit{(**swap*)}\ \text{out}(c, \text{choice}[a, b]) \mid \textit{(**swap*)}\ \text{out}(c, \text{choice}[b, a]) \\ &= \text{out}(c, \text{choice}[a, a]) \mid \text{out}(c, \text{choice}[b, b]). \end{aligned}$$

3.3 Automated reasoning with ProVerif

To allow automated reasoning we describe a translator which accepts as input processes written in our extended language. It will also include a single main process and subprocesses of the form "let $P = Q$", subject to the following restrictions.

1. The commands strong phase t; and *(**swap*)* can only appear in a single subprocess defined using the let keyword (not in the main process);
2. The subprocess defined using the let keyword that contain strong phases and data swapping must be instantiated precisely twice in the main process. Moreover, it must be of the form let $P = \alpha$, where α is a process that is sequential until its last strong phase, at which point it is an arbitrary process. Formally α is given by the grammar below:

   $$\alpha := R \big| \text{new } a; \alpha \big| \text{in}(M, x); \alpha \big| \text{out}(M, N); \alpha \big| \text{let } x = D \text{ in } \alpha \big| \text{strong phase } t; \alpha$$

   where R is an arbitrary processes without data swapping and strong phases;
3. We further require that *(**swap*)* may only occur at the start of a subprocess definition or immediately after a strong phase.

The translator outputs processes in the standard language of ProVerif, which can be automatically reasoned about by the software tool. The pseudocode of our algorithm is presented in Figure 1.

Step one of our translator makes the necessary modifications to subprocesses. It defines each strong phase as an individual subprocess. Step two handles the main process which combines the subprocesses defined in step one in such a way that preserves notion of strong phases. The other parts of the translator's input are copied to the output verbatim. We demonstrate its application with several toy examples (see Section 3.4) and two case studies (see Sections 4 & 5).

Step 1: We replace any subprocess declaration of the form

$$\text{let } P = \alpha_0;\ \text{strong phase 1};\ \alpha_1;\ \text{strong phase 2};\ \alpha_2;\ \ldots;\ \text{strong phase } n;\ \alpha_n.$$

with the declarations

$$\begin{array}{l}\text{let } P_0 = \alpha_0; \text{out}(pc, M_0).\\ \text{let } P_1 = \alpha_1; \text{out}(pc, M_1).\\ \quad\vdots\\ \text{let } P_{n-1} = \alpha_{n-1}; \text{out}(pc, M_{n-1}).\\ \text{let } P_n = \alpha_n.\end{array}$$

where M_i is a term consisting of a tuple containing each bound name in $\alpha_0, \alpha_1, \ldots, \alpha_i$ and the free variables in $\alpha_{i+1}, \alpha_{i+2}, \ldots, \alpha_n$.

Step 2: We replace instance declarations in the main process of the form

$$\text{let } \widetilde{x} = \widetilde{N} \text{ in } P \mid \text{let } \widetilde{x} = \widetilde{N}' \text{ in } P$$

with

$$\begin{array}{l}\text{new } pc_0; \text{new } pc'_0; \text{new } pc_1; \text{new } pc'_1; \ldots; \text{new } pc_{n-1}; \text{new } pc'_{n-1}; (\\ \quad \text{let } \widetilde{x} = \widetilde{N} \text{ in let } pc = pc_0 \text{ in } P_0 |\\ \quad \text{let } \widetilde{x} = \widetilde{N}' \text{ in let } pc = pc'_0 \text{ in } P_0 |\\ \quad \text{in}(pc_0, z_0); \text{in}(pc'_0, z'_0); \textit{(* start strong phase 1 *)} (\\ \quad\quad \text{let } M_0 = z_0 \text{ in let } pc = pc_1 \text{ in } P_1 |\\ \quad\quad \text{let } M_0 = z'_0 \text{ in let } pc = pc'_1 \text{ in } P_1) |\\ \quad\quad\vdots\\ \quad \text{in}(pc_{n-1}, z_{n-1}); \text{in}(pc'_{n-1}, z'_{n-1}); \textit{(* start strong phase n *)} (\\ \quad\quad \text{let } M_{n-1} = z_{n-1} \text{ in } P_n |\\ \quad\quad \text{let } M_{n-1} = z'_{n-1} \text{ in } P_n)\\ )\end{array}$$

If α_0 starts with *(**swap*)*, we further modify the above description, by replacing

$$\begin{array}{lll}\text{let } \widetilde{x} = \widetilde{N} \text{ in} & \textit{with} & \text{let } \widetilde{x} = \text{choice}[\widetilde{N}, \widetilde{N}'] \text{ in}\\ \text{let } \widetilde{x} = \widetilde{N}' \text{ in} & \textit{with} & \text{let } \widetilde{x} = \text{choice}[\widetilde{N}', \widetilde{N}] \text{ in}\end{array}$$

Similarly, if α_i starts with *(**swap*)* and $1 \leq i \leq n$, we further modify the description

$$\begin{array}{lll}\text{let } M_i = z_i \text{ in} & \textit{with} & \text{let } M_i = \text{choice}[z_i, z'_i] \text{ in}\\ \text{let } M_i = z'_i \text{ in} & \textit{with} & \text{let } M_i = \text{choice}[z'_i, z_i] \text{ in}\end{array}$$

Fig. 1 Translator algorithm

3.4 Examples

Example 4. We begin by returning to our trivial observational equivalence:

$$\text{out}(c, a) \mid \text{out}(c, b) \sim \text{out}(c, b) \mid \text{out}(c, a).$$

As the definition of observational equivalence by UUR is too strong, the calculus, and therefore the software tool, are unable to reason about such an equivalence. Using our data swapping syntax, the biprocess encoding the previous equivalence is given below.

```
let P = (**swap*) out(c,x).
process let x = choice[a,b] in P| let x = choice[b,a] in P
```

Our translator gives us the following biprocess, which ProVerif can successfully prove.

```
let P = out(c,x).
process let x = choice[choice[a,b],choice[b,a]] in P|
        let x = choice[choice[b,a],choice[a,b]] in P
```

Example 5. We consider the observational equivalence shown below:

$$\begin{aligned} &\text{out}(c, a); \text{strong phase 1}; \text{out}(c, d) \mid \text{out}(c, b); \text{strong phase 1}; null \\ \sim\ &\text{out}(c, a); \text{strong phase 1}; null \mid \text{out}(c, b); \text{strong phase 1}; \text{out}(c, d) \end{aligned}$$

The pair of processes are both able to output a and b. We then have a synchronisation and obtain the process $\text{out}(c, d) \mid null \sim null \mid \text{out}(c, d)$. To allow ProVerif to prove such an equivalence we provide our translator with the following input:

```
let P =out(c,x); strong phase 1; (**swap*) if y=ok then out(c,d).
process let x = a in let y = choice[ok,ko] in P|
        let x = b in let y = choice[ko,ok] in P
```

Our translator produces the biprocess described below.

```
let P1 = out(c,x); out(pc,y).
let P2 = if y = ok then out(c,c).
process new pc0; new pc1;(
  let x = a in let y = choice[ok,ko] in let pc = pc0 in P1|
  let x = b in let y = choice[ko,ok] in let pc = pc1 in P1|
  in(pc0,y0); in(pc1,y1);(
    let y = choice[y0,y1] in P2|
    let y = choice[y1,y0] in P2))
```

Example 6. As our final example we consider the following equivalence:

$$\begin{aligned} &\text{out}(c, a_1); \text{strong phase 1}; \text{out}(c, a_2) \mid \text{out}(c, b_1); \text{strong phase 1}; \text{out}(c, b_2) \\ \sim\ &\text{out}(c, a_1); \text{strong phase 1}; \text{out}(c, b_2) \mid \text{out}(c, b_1); \text{strong phase 1}; \text{out}(c, a_2) \end{aligned}$$

This is similar to Example 5 with two outputs after the strong phase. Again, thanks to our translator, we are able to conclude on such an example.

4 E-voting protocol due to Fujioka *et al.*

In this section, we study the privacy property of the e-voting protocol due to Fujioka *et al.* [10]. In [12], it is shown that this protocol provides fairness, eligibility and privacy. However, the proof of privacy given in [12] is manual: ProVerif is unable to prove it directly.

4.1 Description

The protocol involves voters, an administrator and a collector. The administrator is responsible for verifying that only eligible voters can cast votes and the collector handles the collecting and publishing of votes. The protocol requires three strong phases.

In the first phase, the voter gets a signature on a commitment to his vote from the administrator, i.e. $m = sign(blind(commit(v, k), r), ska)$ where k is a random key, r is a blinding factor and ska is the private key of the administrator. At the end of this first phase, the voter unblinds m and obtains $y = sign(commit(v, k), ska)$, i.e. the signature of his commitment. The second phase of the protocol is the actual voting phase. The voter sends y to the collector who checks correctness of the signature and, if the test succeeds, enters (ℓ, x, y) onto a list as an ℓ-th item. The last phase of the voting protocol starts, once the collector decides that he received all votes, e.g. after a fixed deadline. In this phase the voters reveal the random key k which allows the collector to open the votes and publish them. The voter verifies that his commitment is in the list and sends ℓ, r to the collector. Hence, the collector opens the ballots.

4.2 Modelling privacy in applied pi

Privacy properties have been successfully studied using equivalences. In the context of voting protocols, the definition of privacy is rather subtle. We recall the definition of privacy for electronic voting protocols given in [12]. A voting protocol guarantees ballot secrecy (privacy) whenever a process where Alice votes for candidate v_1 and Bob votes for candidate v_2 is observationally equivalent to a process where their votes are swapped, i.e. Alice votes v_2 and Bob votes v_1. We denote their secret keys $skva$ and $skvb$ respectively. In [12],

```
let V =
 new k;new r;
 let x = commit(v,k) in
 out(c,(pk(skv),sign(blind(x,r),skv)));
 in(c,m2);
 let y = unblind(m2,r) in
 if checksign(y,pka) = x then
 strong phase 1; (**swap*)
 out(c,y);
 strong phase 2;
 in(c,(l,yprime));
 if yprime = y then out(c,(l,k)).

process
 new ska;new skva;new skvb;
 let pka = pk(ska) in
 out(c,(ska,pka,pk(skva),pk(skvb)));(
  (let (skv,v) = (skva,choice[v1,v2]) in V)|
  (let (skv,v) = (skvb,choice[v2,v1]) in V))
```

Process 1 FOO model (extended syntax)

they rely on hand proof techniques to show privacy on FOO. Our modelling of FOO in the applied pi is similar to the one given in [8]. The underlying equational theory is the same as in [12].

The main process given in Process 1 models the environment and specifies how the other processes are combined. To establish privacy, we do not require that the authorities are honest, so we do not need to model them and we only consider two voter processes in parallel. First, fresh private keys for the voters and the administrator are generated. The corresponding public keys are then made available to the attacker. We also output the secret key of the administrator. This allows the environment to simulate the administrator (even a corrupted one) and hence we show that the privacy property holds even in the presence of a corrupt administrator.

The process V given in Process 1 models the role of a voter. The specification follows directly from our informal description. Note that we use the strong phase command to enforce the synchronisation of the voter processes. As mentioned initially in [12], the separation of the protocol into strong phases is crucial for privacy to hold. We also provide a data swapping hint to allow our translator to produce an output suitable for automatic verification using ProVerif.

4.3 Analysis

We use our translator to remove all instances of strong phases and handle data swapping. Our translator produces Process 2, which is suitable for automatic

```
let V1 =
 new k;new r;
 let x = commit(v,k) in
 out(c,(pk(skv),sign(blind(x,r),skv)));
 in(c,m2);
 let y = unblind(m2,r) in
 if checksign(y,pka) = x then out(pc,(y,k)).

let V2 =
 out(c,y); out(pc,(y,k)).

let V3 =
 in(c,(l,yprime)); if yprime = y then out(c,(l,k)).

process
 new ska;new skva;new skvb;
 let pka = pk(ska) in
 out(c,(ska,pka,pk(skva),pk(skvb)));
 new pc1;new pc2;new pc3;new pc4;(
  (let (skv,v)=(skva,choice[v1,v2]) in let pc=pc1 in V1)|
  (let (skv,v)=(skvb,choice[v2,v1]) in let pc=pc2 in V1)|
  (in(pc1,(y1,k1));in(pc2,(y2,k2));(*strong phase 1*)(*swap*)(
   (let (y,k)=choice[(y1,k1),(y2,k2)] in let pc=pc3 in V2)|
   (let (y,k)=choice[(y2,k2),(y1,k1)] in let pc=pc4 in V2)))|
  (in(pc3,(y3,k3));in(pc4,(y4,k4));(*strong phase 2*)(
    (let (y,k)=(y3,k3) in V3)|
    (let (y,k)=(y4,k4) in V3))))
```

Process 2 Translated FOO model (ProVerif syntax)

verification using ProVerif. Hence, using our approach, we provide the first automatic and complete proof that this protocol satisfies privacy.

5 Direct Anonymous Attestation (DAA)

The Direct Anonymous Attestation (DAA) scheme provides a means for remotely authenticating a trusted platform whilst preserving the user's privacy [4]. In [15], two of the authors have shown that corrupt administrators are able to violate the privacy of the host. Using our extended calculus we are now able to provide a formal and automatic proof that the rectified protocol proposed in [15] satisfies its privacy requirements. We start with a short description of the protocol (for a more complete description, see [4, 15]).

5.1 Description

The protocol can be seen as a group signature scheme without the ability to revoke anonymity and an additional mechanism to detect rogue members. In broad terms the *host* contacts an *issuer* and requests membership to a group. If the issuer wishes to accept the request, it grants the host/TPM an *attestation identity credential*. The host is now able to anonymously authenticate itself as a group member to a *verifier* with respect its credential.

The protocol is initiated when a host wishes to obtain a credential. This is known as the join protocol. The TPM creates a secret f value and a blinding factor v', where $f = hash(hash(\texttt{DAASeed}\|hash(PK'_I))\|\texttt{cnt}\|0)$. The value `DAASeed` is a secret known only to the TPM, `cnt` is a counter used by the TPM to keep track of how many times the Join protocol has been run and PK'_I is the long term public key of the issuer. The inclusion of PK'_I prevents cross issuer linkability [15]. The TPM then constructs the blind message $U := blind(f, v')$ and $N_I := \zeta_I^f$, where $\zeta_I := hash(0\|bsn_I)$ and bsn_I is the basename of the issuer (see [15] for further discussion on DAA basenames). The U and N_I values are submitted to the issuer I. The issuer creates a random nonce value n_e, encrypts it with the public key PK_{EK} of the host's TPM and returns the encrypted value. The TPM decrypts the message, revealing n_e, and returns $hash(U\|n_e)$. The issuer confirms that the hash is correctly formed. The issuer generates a nonce n_i and sends it to the host. The host/TPM constructs a signature proof of knowledge that the messages U and N_I are correctly formed. The issuer verifies the proof and generates a blind signature on the message U. It returns the signature along with a proof that a covert channel has not been used. The host verifies the signature and proof and the TPM unblinds the signature revealing a secret credential v (the signed f).

Once the host has obtained an anonymous attestation credential from the issuer it is able to produce a signature proof of knowledge of attestation on a message m. This is known as the sign/verify protocol. The verifier sends nonce n_v to the host. The host/TPM produce a signature proof of knowledge of attestation on the message $(n_t\|n_v\|b\|m)$, where n_t is a nonce defined by the TPM and b is a parameter. In addition the host computes $N_V := \zeta^f$, where $\zeta := hash(1\|bsn_V)$ and bsn_V is the basename of the verifier. Intuitively if a verifier is presented with such a proof it is convinced that it is communicating with a trusted platform and the message is genuine.

5.2 Modelling privacy in applied pi

The DAA protocol satisfies privacy whenever a process where Alice interacts with the verifier is observationally equivalent to when Bob interacts with the

verifier. For privacy we require that both Alice and Bob have completed the join protocol.

Signature and equational theory. The DAA protocol makes extensive use of signature proofs of knowledge (SPK) to prove knowledge of and relations among discrete logarithms. We will discuss our formalism with an example. The signature proof of knowledge $\mathrm{SPK}\{(\alpha, \beta) : x = g^\alpha \wedge y = h^\beta\}(m)$ denotes a signature proof of knowledge on the message m that x, y were constructed correctly. This leads us to define function `spk/3` to construct an SPK. The first argument contains a tuple of secret values known to the prover α, β. The second argument consists of a tuple of the values on which the prover is claiming to have constructed correctly x, y, such that $x = g^\alpha$ and $y = h^\beta$. Finally the third argument is the message m on which the prover produces a signature on. Verifying the correctness of a SPK is specific to its construction, thus we must require a function `checkspk` for each SPK that the protocol uses. To verify the SPK produced in the aforementioned example the verifier must be in possession of the SPK itself and x, y, g, h, m. We define the equation: $\mathit{checkspk}(\mathit{spk}((\alpha, \beta), (g^\alpha, h^\beta), m), g^\alpha, h^\beta, g, h, m) = \mathit{ok}$. A verifier can now check a SPK using an `if` statement.

Modelling the DAA protocol. As in FOO, the main process (see [9]) models the environment and specifies how the other processes are combined. First, fresh secret keys for the TPMs, the issuer and the verifier are generated using the restriction operator. We also generate two `DAASeed` values. The public keys are then sent on a public channel, i.e. they are made available to the intruder. We also output the secret key of the verifier and issuer since the privacy property should be preserved even if they are corrupt. Next we input the basenames $\mathit{bsn}_I, \mathit{bsn}_V$ of the issuer and verifier. Then we instantiate two instances of the DAA protocol with the necessary parameters.

Our encoding of the DAA protocol (see [9]) follows directly from our informal description. Note that we use the strong phase and data swapping commands introduced by our extension to the calculus to ensure synchronisation. The two instances of the DAA processes must first execute all instructions of DAAJoin before moving onto DAASign. The separation of the protocol into strong phases is crucial for privacy to hold.

5.3 Analysis

We use our translator to remove all instances of strong phases from our encoding and produce code suitable for input to ProVerif. Our translator produces a process (see [9]) which permits the automatic verification of the privacy property using ProVerif. We are also able to detect the vulnerability in the original DAA protocol and prove the optimisation presented in [15].

6 Conclusion

In this paper we have extended the class of equivalences which ProVerif is able to automatically verify. More specifically we are able to reason about processes which require data swapping and/or strong phases. Using the approach developed we are able to automatically verify the privacy properties of the electronic voting protocol FOO and the Direct Anonymous Attestation scheme. In future work, we would like to generalise our translation algorithm and provide a formal proof of the correctness of our translator. Moreover we plan to automate the swapping procedure.

References

1. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: POPL'01: Proc. 28th ACM Symposium on Principles of Programming Languages, pp. 104–115. ACM Press, New York, USA (2001)
2. Backes, M., Maffei, M., Unruh, D.: Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In: IEEE Symposium on Security and Privacy, Proceedings of SSP'08 (2008). To appear
3. Blanchet, B., Abadi, M., Fournet, C.: Automated verification of selected equivalences for security protocols. Journal of Logic and Algebraic Programming **75**(1), 3–51 (2008)
4. Brickell, E., Camenisch, J., Chen, L.: Direct Anonymous Attestation. In: CCS '04: 11th ACM conference on Computer and communications security, pp. 132–145. ACM Press, New York, USA (2004)
5. Chadha, R., Kremer, S., Scedrov, A.: Formal Analysis of Multi-Party Fair Exchange Protocols. In: R. Focardi (ed.) 17th IEEE Computer Security Foundations Workshop, pp. 266–279. IEEE, Asilomar, USA (2004)
6. Clark, J., Jacob, J.: A Survey of Authentication Protocol Literature (1997). URL http://www.cs.york.ac.uk/~jac/papers/drareviewps.ps
7. Delaune, S., Kremer, S., Ryan, M.: Coercion-Resistance and Receipt-Freeness in Electronic Voting. In: CSFW '06: Proc. 19th IEEE workshop on Computer Security Foundations, pp. 28–42. IEEE (2006)
8. Delaune, S., Kremer, S., Ryan, M.D.: Verifying privacy-type properties of electronic voting protocols. Research report, Laboratoire Spécification et Vérification, ENS Cachan, France (2008)
9. Delaune, S., Ryan, M., Smyth, B.: Automatic verification of privacy properties in the applied pi calculus (extended version) (2008). URL http://www.cs.bham.ac.uk/~bas/
10. Fujioka, A., Okamoto, T., Ohta, K.: A Practical Secret Voting Scheme for Large Scale Elections. In: ASIACRYPT '92: Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques, pp. 244–251. Springer, London (1993)
11. Hirt, M., Sako, K.: Efficient receipt-free voting based on homomorphic encryption. In: Eurocrypt, *LNCS*, vol. 1807, pp. 539–556 (2000)
12. Kremer, S., Ryan, M.D.: Analysis of an Electronic Voting Protocol in the Applied Pi Calculus. In: ESOP'05: Proc. of the European Symposium on Programming, *LNCS*, vol. 3444, pp. 186–200 (2005)
13. Lowe, G.: An attack on the Needham-Schroeder public-key authentication protocol. Information Processing Letters **56**(3), 131–133 (1995)
14. Mukhamedov, A., Ryan, M.D.: Fair Multi-party Contract Signing using Private Contract Signatures. Information & Computation (2007). Preprint
15. Smyth, B., Ryan, M., Chen, L.: Direct Anonymous Attestation (DAA): Ensuring privacy with corrupt administrators. In: ESAS'07: 4th European Workshop on Security and Privacy in Ad hoc and Sensor Networks, *LNCS*, vol. 4572, pp. 218–231 (2007)

A method for proving observational equivalence

Véronique Cortier
LORIA, CNRS & INRIA Nancy Grand Est
France
Email: cortier@loria.fr

Stéphanie Delaune
LSV, ENS Cachan & CNRS & INRIA Saclay
France
Email: delaune@lsv.ens-cachan.fr

***Abstract*—Formal methods have proved their usefulness for analyzing the security of protocols. Most existing results focus on trace properties like secrecy (expressed as a reachability property) or authentication. There are however several security properties, which cannot be defined (or cannot be naturally defined) as trace properties and require the notion of *observational equivalence*. Typical examples are anonymity, privacy related properties or statements closer to security properties used in cryptography.**

In this paper, we consider the applied pi calculus and we show that for *determinate* processes, observational equivalence actually coincides with trace equivalence, a notion simpler to reason with. We exhibit a large class of determinate processes, called *simple processes*, that capture most existing protocols and cryptographic primitives. Then, for simple processes without replication nor else branch, we reduce the decidability of trace equivalence to deciding an equivalence relation introduced by M. Baudet. Altogether, this yields the first decidability result of observational equivalence for a general class of equational theories.

I. Introduction

Security protocols are paramount in today's secure transactions through public channels. It is therefore essential to obtain as much confidence as possible in their correctness. Formal methods have proved their usefulness for precisely analyzing the security of protocols. In the case of a bounded number of sessions, secrecy preservation is co-NP-complete [5], [24], [25], and for an unbounded number of sessions, several decidable classes have been identified (e.g. [23]). Many tools have also been developed to automatically verify cryptographic protocols (e.g. [9], [6]).

Most existing results focus on trace properties, that is, statements that something bad never occurs on any execution trace of a protocol. Secrecy and authentication are typical examples of trace properties. There are however several security properties, which cannot be defined (or cannot be naturally defined) as trace properties and require the notion of *observational equivalence*. We focus here on the definition proposed in the context of applied pi-calculus [2], which is well-suited for the analysis of security protocols. Two processes P and Q are observationally equivalent, denoted by $P \approx Q$, if for any process O the processes $P \mid O$ and $Q \mid O$ are equally able to emit on a given channel and are (weakly) bisimilar. This means that the process O cannot observe any difference between the processes P and Q.

This work has been partially supported by the ANR-07-SESU-002 AVOTÉ.

Observational equivalence is crucial when specifying properties like anonymity that states that an observer cannot distinguish the case where A is talking from the case where B is talking (see [3]). Privacy related properties involved in electronic voting protocols (e.g. [17]) also use equivalence as a key notion and cannot be expressed in linear temporal logic. Observational equivalence is also used for defining a stronger notion of secrecy, called "strong secrecy" [10] or even for defining authentication [4]. More generally, it is a notion that allows to express flexible notions of security by requiring observational equivalence between a protocol and an idealized version of it, that magically realizes the desired properties.

Related work.: In contrast to the case of trace properties, there are very few results on automating the analysis of observational equivalence. Decidability results are limited to fixed cryptographic primitives in spi-calculus (e.g. [21], [18]). In applied-pi calculus, an alternative approach has been considered [16], [7], [11] for arbitrary cryptographic primitives. The approach consists in designing stronger notions of equivalences that imply observational equivalence. One of these techniques has been implemented in ProVerif [11]. None of these are however complete, that is, there exist observationally equivalent processes that do not satisfy these stronger notions of equivalences.

Our contributions.: One of the difficulties in proving observational equivalence is the bisimulation property. Although bisimulation-based equivalences may be simpler to check than trace equivalences [22], in the context of cryptographic protocols, it seems easier to simply check *trace equivalence*, that is, equality of the set of execution traces (modulo some equivalence relation between traces). In particular, most decision techniques have been developed for trace properties only. However, it is well-known that this is not sufficient to ensure observational equivalence. J. Engelfriet has shown that observational equivalence and trace equivalence actually coincide in a general model of parallel computation with atomic actions, when processes are *determinate* [20]. Intuitively, a process P is determinate if after the same experiment s, the resulting processes are equivalent, that is, if $P \stackrel{s}{\Rightarrow} P'$ and $P \stackrel{s}{\Rightarrow} P''$ then $P' \approx P''$. Our first contribution is to generalize this result to the applied pi-calculus, which consists in the pi-calculus algebra enriched with terms and equational theories on terms.

Then we show that a large class of processes enjoys the determinacy property. More precisely, we design the class of *simple processes* and show that simple processes are determinate. Simple processes allow replication, `else` branches and arbitrary term algebra modulo an equational theory. Consequently, this class captures most existing security protocols and cryptographic primitives. In addition, our simple processes are close to the fragment considered in [14] for which cryptographic guarantees can be deduced from observational equivalence. The class of processes defined in [14] is however not determinate but we believe that their result could be easily extended to our class of simple processes, yielding to a decision technique for proving indistinguishability in cryptographic models.

Our third contribution is a decidability result for simple processes without replication nor `else` branch and for *convergent subterm theories*. Convergent subterm theories capture a wide array of functions, e.g. pairing, projections, various flavors of encryption and decryption, digital signatures, one-way hash functions, *etc*. We show that trace equivalence of simple processes without replication can be reduced to deciding an equivalence relation introduced by M. Baudet and which has been shown decidable for convergent subterm theories in [7].

Putting our three contributions together, we obtain decidability of observational equivalence for a large and interesting class of processes of the applied pi-calculus. This is the first decidability result for a general class of equational theories. Some of the proofs are omitted but can be found in [15].

II. The applied pi calculus

The *applied pi calculus* [2] is a derivative of the pi calculus that is specialized for modeling cryptographic protocols. Participants in a protocol are modeled as processes, and the communication between them is modeled by means of message passing.

A. Syntax

To describe processes in the applied-pi calculus, one starts with a set of *names* $\mathcal{N} = \{a, b, \ldots, sk, k, n, \ldots\}$, which is split into the set $\mathcal{N}_b$ of names of *basic types* and the set $\mathcal{C}h$ of names of *channel type* (which are used to name communication channels). We also consider a set of *variables* $\mathcal{X} = \{x, y, \ldots\}$, and a *signature* $\mathcal{F}$ consisting of a finite set of *function symbols*. We rely on a sort system for terms. The details of the sort system are unimportant, as long as *base types* differ from *channel types*. We suppose that function symbols only operate on and return terms of base type.

Terms are defined as names, variables, and function symbols applied to other terms. For $\mathsf{N} \subseteq \mathcal{N}$ and $\mathsf{X} \subseteq \mathcal{X}$, the set of terms built from N and X by applying function symbols in $\mathcal{F}$ is denoted by $\mathcal{T}(\mathsf{N}, \mathsf{X})$. Of course function symbol application must respect sorts and arities. We write $fv(T)$ for the set of variables occurring in T. The term T is said to be a *ground* term if $fv(T) = \emptyset$. We shall use $u, v, \ldots$ to denote *metavariables* that range over both names and variables.

Example 1: Consider the following signature

$$\mathcal{F} = \{\mathsf{enc}/2,\ \mathsf{dec}/2,\ \mathsf{pk}/1,\ \langle\,\rangle/2,\ \pi_1/1,\ \pi_2/1\}$$

that contains function symbols for asymmetric encryption, decryption and pairing, each of arity 2, as well as projection symbols and the function symbol pk, each of arity 1. The ground term $\mathsf{pk}(sk)$ represents the public counterpart of the private key sk.

In the applied pi calculus, one has *plain processes*, denoted P, Q, R and *extended processes*, denoted by A, B, C. Plain processes are built up in a similar way to processes in pi calculus except that messages can contain terms rather than just names. Extended processes add *active substitutions* and restriction on variables (see Figure 1).

The substitution $\{^M/_x\}$ is an active substitution that replaces the variable x with the term M. Active substitutions generalize the "let" construct: $\nu x.(\{^M/_x\} \mid P)$ corresponds exactly to

$$\text{"let } x = M \text{ in } P\text{"}.$$

As usual, names and variables have scopes, which are delimited by restrictions and by inputs. We write $fv(A)$, $bv(A)$, $fn(A)$ and $bn(A)$ for the sets of *free* and *bound variables* and *free* and *bound names* of A, respectively. We say that an extended process is *closed* if all its variables are either bound or defined by an active substitution. An *evaluation context* $C[\_]$ is an extended process with a hole instead of an extended process.

Active substitutions are useful because they allow us to map an extended process A to its *frame*, denoted $\phi(A)$, by replacing every plain process in A with 0. Hence, a frame is an extended process built up from 0 and active substitutions by parallel composition and restriction. The frame $\phi(A)$ accounts for the set of terms statically known by the intruder (but does not take into account for A's dynamic behavior). The domain of a frame φ, denoted by $\mathrm{dom}(\varphi)$, is the set of variables for which φ defines a substitution (those variables x for which φ contains a substitution $\{^M/_x\}$ not under a restriction on x).

Example 2: Consider the following process A made up of three components in parallel:

$$\begin{aligned}\nu s, sk, x_1.\quad & (\mathsf{out}(c_1, x_1)\\ & \mid \mathsf{in}(c_1, y).\mathsf{out}(c_2, \mathsf{dec}(y, sk))\\ & \mid \{^{\mathsf{enc}(s,\mathsf{pk}(sk))}/_{x_1}\}).\end{aligned}$$

Its first component publishes the message $\mathsf{enc}(s, \mathsf{pk}(sk))$ stored in x_1 by sending it on c_1. The second receives a message on c_1, uses the secret key sk to decrypt it, and forwards the result on c_2. We have $\phi(A) = \nu s, sk, x_1.\{^{\mathsf{enc}(s,\mathsf{pk}(sk))}/_{x_1}\}$ and $\mathrm{dom}(\phi(A)) = \emptyset$ (since x_1 is under a restriction).

B. Semantics

We briefly recall the operational semantics of the applied pi calculus (see [2] for details). First, we associate an *equational*

$$
\begin{array}{lll}
P, Q, R := 0 & \text{plain processes} & A, B, C := \quad \text{extended processes} \\
\quad P \mid Q & & \quad P \\
\quad !P & & \quad A \mid B \\
\quad \nu n.P & & \quad \nu n.A \\
\quad \texttt{if } M = N \texttt{ then } P \texttt{ else } Q & & \quad \nu x.A \\
\quad \mathrm{in}(u, x).P & & \quad \{^M/_x\} \\
\quad \mathrm{out}(u, N).P & &
\end{array}
$$

where M and N are terms, n is a name, x a variable and u is a metavariable.

Fig. 1. Syntax of processes

theory E to the signature $\mathcal{F}$. The equational theory is defined by a set of equations $M = N$ with $M, N \in \mathcal{T}(\emptyset, \mathsf{X})$ and induces an equivalence relation over terms: $=_{\mathsf{E}}$ is the smallest equivalence relation on terms, which contains all equations $M = N$ in E and that is closed under application of contexts and substitution of terms for variables. Since the equations in E do not contain any names, we have that E is also closed by substitutions of terms for names.

Example 3: Consider the signature $\mathcal{F}$ of Example 1. We define the equational theory $\mathsf{E}_{\mathsf{enc}}$ by the following equations:

$$
\begin{array}{rcl}
\mathsf{dec}(\mathsf{enc}(x, \mathsf{pk}(y)), y) & = & x \\
\pi_i(\langle x_1, x_2 \rangle) & = & x_i \quad \text{for } i \in \{1, 2\}.
\end{array}
$$

We have that $\pi_1(\mathsf{dec}(\mathsf{enc}(\langle n_1, n_2 \rangle, \mathsf{pk}(sk)), sk)) =_{\mathsf{E}_{\mathsf{enc}}} n_1$.

Structural equivalence, noted $\equiv$, is the smallest equivalence relation on extended processes that is closed under α-conversion of names and variables, by application of evaluation contexts, and satisfying some further basic structural rules such as $A \mid 0 \equiv A$, associativity and commutativity of $\mid$, binding-operator-like behavior of ν, and when $M =_{\mathsf{E}} N$ the equivalences:

$$
\nu x.\{^M/_x\} \equiv 0 \qquad \{^M/_x\} \equiv \{^N/_x\}
$$
$$
\{^M/_x\} \mid A \equiv \{^M/_x\} \mid A\{^M/_x\}.
$$

Example 4: Let P be the following process:

$$
\begin{array}{ll}
\nu s, sk. & (\mathrm{out}(c_1, \mathsf{enc}(s, \mathsf{pk}(sk))) \\
& \mid \mathrm{in}(c_1, y).\mathrm{out}(c_2, \mathsf{dec}(y, sk))).
\end{array}
$$

The process P is structurally equivalent to the process A given in Example 2. We have that $\phi(P) = 0 \equiv \phi(A)$.

The operational semantics of processes in the applied pi calculus is defined by structural rules defining two relations: *structural equivalence* (described above) and *internal reduction*, noted $\xrightarrow{\tau}$. Internal reduction is the smallest relation on extended processes closed under structural equivalence and application of evaluation contexts such that:

$$
\begin{array}{rcl}
\mathrm{out}(a, x).P \mid \mathrm{in}(a, x).Q & \xrightarrow{\tau} & P \mid Q \\
\texttt{if } M = M \texttt{ then } P \texttt{ else } Q & \xrightarrow{\tau} & P \\
\texttt{if } M = N \texttt{ then } P \texttt{ else } Q & \xrightarrow{\tau} & Q
\end{array}
$$

where M, N are ground terms such that $M \neq_{\mathsf{E}} N$

The operational semantics is extended by a *labeled* operational semantics enabling us to reason about processes that interact with their environment. Labeled operational semantics defines the relation $\xrightarrow{\ell}$ where ℓ is either an input or an output. We adopt the following rules in addition to the internal reduction rules. Below, the names a and c are channel names whereas x is a variable of base type and y is a variable of any type.

$$
\textsc{In} \qquad \mathrm{in}(a, y).P \xrightarrow{in(a,M)} P\{^M/_y\}
$$

$$
\textsc{Out-Ch} \qquad \mathrm{out}(a, c).P \xrightarrow{out(a,c)} P
$$

$$
\textsc{Open-Ch} \qquad \frac{A \xrightarrow{out(a,c)} A' \qquad c \neq a}{\nu c.A \xrightarrow{\nu c.out(a,c)} A'}
$$

$$
\textsc{Out-T} \qquad \mathrm{out}(a, M).P \xrightarrow{\nu x.out(a,x)} P \mid \{^M/_x\} \quad x \notin fv(P) \cup fv(M)
$$

$$
\textsc{Scope} \qquad \frac{A \xrightarrow{\ell} A' \quad u \text{ does not occur in } \ell}{\nu u.A \xrightarrow{\ell} \nu u.A'}
$$

$$
\textsc{Par} \qquad \frac{A \xrightarrow{\ell} A' \quad bn(\ell) \cap fn(B) = \emptyset \quad bv(\ell) \cap fv(B) = \emptyset}{A \mid B \xrightarrow{\ell} A' \mid B}
$$

$$
\textsc{Struct} \qquad \frac{A \equiv B \quad B \xrightarrow{\ell} B' \quad B' \equiv A'}{A \xrightarrow{\ell} A'}
$$

Note that the labeled transition is not closed under application of evaluation contexts. Moreover the output of a term M needs to be made "by reference" using a restricted variable and an active substitution. The rules differ slightly from those described in [2] but it has been shown in [16] that the two underlying notions of observational equivalence coincide.

III. Trace and observational equivalences

Let $\mathcal{A}$ be the alphabet of actions (in our case this alphabet is infinite) where the special symbol $\tau \in \mathcal{A}$ represents an unobservable action. For every $\alpha \in \mathcal{A}$ the relation $\xrightarrow{\alpha}$ has been defined in Section II. For every $w \in \mathcal{A}^*$ the relation $\xrightarrow{w}$ on extended processes is defined in the usual way. By convention $A \xrightarrow{\epsilon} A$ where ϵ denotes the empty word.

For every $s \in (\mathcal{A} \setminus \{\tau\})^*$, the relation $\stackrel{s}{\Rightarrow}$ on extended processes is defined by: $A \stackrel{s}{\Rightarrow} B$ if, and only if, there exists $w \in \mathcal{A}^*$ such that $A \xrightarrow{w} B$ and s is obtained from w by erasing all occurrences of τ. Intuitively, $A \stackrel{s}{\Rightarrow} B$ means that A transforms into B by experiment s. We also consider the relation $A \stackrel{w}{\mapsto} B$ and $A \stackrel{s}{\Mapsto} B$ that are the restriction of the relations $\xrightarrow{w}$ and $\stackrel{s}{\Rightarrow}$ on closed extended processes.

A. Observational equivalence

Intuitively, two processes are *observationally equivalent* if they cannot be distinguished by any active attacker represented by any context.

We write $A \Downarrow c$ when A can send a message on c, that is, when $A \rightarrow^* C[\mathrm{out}(c, M).P]$ for some evaluation context C that does not bind c.

Definition 1: *Observational equivalence* is the largest symmetric relation $\mathcal{R}$ between closed extended processes with the same domain such that $A \mathcal{R} B$ implies:

1) if $A \Downarrow c$, then $B \Downarrow c$;
2) if $A \rightarrow^* A'$, then $B \rightarrow^* B'$ and $A' \mathcal{R} B'$ for some B';
3) $C[A] \mathcal{R} C[B]$ for all closing evaluation contexts C.

Observational equivalence can be used to formalize many interesting security properties, in particular privacy related properties, such as those studied in [3], [17]. However, proofs of observational equivalences are difficult because of the universal quantification over all contexts. It has been shown that observational equivalence coincides with labeled bisimilarity [2]. This result was first proved in the context of the spi-calculus [12]. Before defining the notion of labeled bisimilarity, we introduce a notion of intruder's knowledge that has been extensively studied (e.g. [1]).

Definition 2 (static equivalence $\sim$): Two terms M and N are *equal in the frame* ϕ, written $(M =_{\mathsf{E}} N)\phi$, if there exists $\tilde{n}$ and a substitution σ such that $\phi \equiv \nu\tilde{n}.\sigma$, $\tilde{n} \cap (fn(M) \cup fn(N)) = \emptyset$, and $M\sigma =_{\mathsf{E}} N\sigma$.

Two closed frames ϕ_1 and ϕ_2 are *statically equivalent*, written $\phi_1 \sim \phi_2$, when:

- $\mathrm{dom}(\phi_1) = \mathrm{dom}(\phi_2)$, and
- for all terms M, N we have that

$$(M =_{\mathsf{E}} N)\phi_1 \text{ if and only if } (M =_{\mathsf{E}} N)\phi_2.$$

Example 5: Consider the theory $\mathsf{E}_{\mathsf{enc}}$ described in Example 3, and the two frames

- $\varphi_a = \{^{\mathsf{enc}(a,\mathsf{pk}(sk))}/_{x_1}\}$, and
- $\varphi_b = \{^{\mathsf{enc}(b,\mathsf{pk}(sk))}/_{x_1}\}$.

We have that $(\mathsf{dec}(x_1, sk) =_{\mathsf{E}_{\mathsf{enc}}} a)\varphi_a$ whereas $(\mathsf{dec}(x_1, sk) \neq_{\mathsf{E}_{\mathsf{enc}}} a)\varphi_b$, thus we have that $\varphi_a \not\sim \varphi_b$.

However, we have that $\nu sk.\varphi \sim \nu sk.\varphi'$. This is a non trivial equivalence. Intuitively, there is no test that allows one to distinguish the two frames since the decryption key and the encryption key are not available.

Definition 3 (labeled bisimilarity $\approx$): *Labeled bisimilarity* is the largest symmetric relation $\mathcal{R}$ on closed extended processes such that $A \mathcal{R} B$ implies

1) $\phi(A) \sim \phi(B)$,
2) if $A \stackrel{\tau}{\mapsto} A'$, then $B \stackrel{\epsilon}{\Mapsto} B'$ and $A' \mathcal{R} B'$ for some B',
3) if $A \stackrel{\ell}{\mapsto} A'$ and $bn(\ell) \cap fn(B) = \emptyset$ then $B \stackrel{\ell}{\Mapsto} B'$ and $A' \mathcal{R} B'$ for some B'.

Example 6: Consider the theory $\mathsf{E}_{\mathsf{enc}}$ and the two processes $P_a = \mathrm{out}(c, \mathsf{enc}(a, \mathsf{pk}(sk)))$ and $P_b = \mathrm{out}(c, \mathsf{enc}(b, \mathsf{pk}(sk)))$. We have that $\nu sk.P_a \approx \nu sk.P_b$ whereas $P_a \not\approx P_b$. These results are direct consequences of the static (in)equivalence relations stated and discussed in Example 5.

B. Trace equivalence

For every closed extended process A we define its set of traces, each trace consisting in a sequence of actions together with the sequence of sent messages:

$$\mathsf{trace}(A) = \{(s, \phi(B)) \mid A \stackrel{s}{\Mapsto} B \text{ for some } B\}.$$

Note that, in the applied pi calculus, the sent messages are exclusively stored in the frame and not in the sequence s (the outputs are made by "reference").

Definition 4 (trace inclusion $\sqsubseteq_t$): Let A and B be two closed extended processes, $A \sqsubseteq_t B$ if for every $(s, \varphi) \in \mathsf{trace}(A)$ such that $bn(s) \cap fn(B) = \emptyset$, there exists $(s', \varphi') \in \mathsf{trace}(B)$ such that $s = s'$ and $\varphi \sim \varphi'$.

Definition 5 (trace equivalence $\approx_t$): Let A and B be two closed extended processes. They are *trace equivalent*, denoted by $A \approx_t B$, if $A \sqsubseteq_t B$ and $B \sqsubseteq_t A$.

It is easy to see that observational equivalence (or labeled bisimilarity) implies trace equivalence while the converse is false in general (see Example 7).

Lemma 1: Let A and B be two closed extended processes: $A \approx B$ implies $A \approx_t B$.

Example 7: Consider the two following processes:

$$A = \nu c'.(\mathrm{out}(c', ok) \mid \mathrm{in}(c', x).\mathrm{out}(c, a).\mathrm{out}(c, b_1) \mid \mathrm{in}(c', x).\mathrm{out}(c, a).\mathrm{out}(c, b_2))$$

$$B = \mathrm{out}(c, a).\nu c'.(\mathrm{out}(c', ok) \mid \mathrm{in}(c', x).\mathrm{out}(c, b_1) \mid \mathrm{in}(c', x).\mathrm{out}(c, b_2)).$$

We have that $A \approx_t B$ whereas $A \not\approx B$. Intuitively, after B's first move, B still has the choice of emitting b_1 or b_2, while A, trying to follow B's first move, is forced to choose between two states from which it can only emit one of the two.

C. Determinacy

J. Engelfriet has shown that observational and trace equivalence coincide for a process algebra with atomic actions, when processes are *determinate* [20]. First, we define this notion in the context of the applied pi calculus.

Definition 6 (determinacy): Let $\cong$ be an equivalence relation on closed extended processes. A closed extended process A is $\cong$*-determinate* if $A \stackrel{s}{\Rightarrow} B$, $A \stackrel{s}{\Rightarrow} B'$ and $\phi(B) \sim \phi(B')$ implies $B \cong B'$.

Fixing the equivalence relation yields to potentially different notions of determinacy. We define two of them: *observation determinacy* (for $\cong \; := \; \approx$) and *trace determinacy* (for $\cong \; := \; \approx_t$). By using the techniques of J. Engelfriet, we can show that these two notions of determinacy actually coincide. So we say that an extended process is *determinate* if it satisfies any of these two notions.

Lemma 2: Let A be a closed extended process. The process A is observation determinate if, and only if, it is trace determinate.

Example 8: Consider for instance the closed extended process A given in Example 7. We have that $A \stackrel{\tau}{\mapsto} A_1$ and $A \stackrel{\tau}{\mapsto} A_2$ for A_1 and A_2 given below:

$$A_1 = \nu c'. \quad (\mathrm{out}(c,a).\mathrm{out}(c,b_1) \mid \mathrm{in}(c',x).\mathrm{out}(c,a).\mathrm{out}(c,b_2))$$

$$A_2 = \nu c'. \quad (\mathrm{in}(c',x).\mathrm{out}(c,a).\mathrm{out}(c,b_1) \mid \mathrm{out}(c,a).\mathrm{out}(c,b_2)).$$

The process A_1 can output the messages a and then b_1 whereas the process A_2 can output a and then b_2. Thus, the process A is neither observation determinate, nor trace determinate.

Our first main contribution is to extend the result of J. Engelfriet [20] to processes of the applied-pi calculus, showing that observational equivalence and trace equivalence coincide when processes are determinate. The proof of this result is relatively simple once the right definition of determinacy has been fixed. In particular, the presence of equational theories and active substitutions do not cause any change in the proof scheme of [20] since the definition of determinacy already captures their impact on processes.

Theorem 1: Let A and B be two closed extended processes that are determinate.

$$A \approx_t B \text{ implies } A \approx B.$$

Proof (sketch). Let A and B be two closed extended processes that are determinate, and assume that $A \approx_t B$. We consider the relation $\mathcal{R}$ defined as follows:

$$A' \; \mathcal{R} \; B' \text{ iff there exists } s \text{ such that } A \stackrel{s}{\Rightarrow} A',\ B \stackrel{s}{\Rightarrow} B', \text{ and } \phi(A') \sim \phi(B').$$

We have that $A \; \mathcal{R} \; B$. It remains to check that $\mathcal{R}$ satisfies the three points of Definition 3. □

IV. An expressive class of determinate processes

In what follows, we consider any signature and equational theory. We do not need the full applied pi-calculus to represent security protocols. For example, when it is assumed that all communications are controlled by the attacker, private channels between processes are not accurate (they should rather be implemented using cryptography). In addition, the attacker schedules the communications between processes thus he knows exactly to whom he is sending messages and from whom he is listening. Thus we assume that each process communicates on a personal channel.

Formally, we consider the fragment of *simple processes* built on *basic processes*. A basic process represents a session of a protocol role where a party waits for a message of a certain form or checks some equalities and outputs a message accordingly. Then the party waits for another message or checks for other equalities and so on.

Intuitively, any protocol whose roles have a deterministic behavior can be modeled as a simple process. Most of the roles are indeed deterministic since an agent should usually exactly know what to do once he has received a message. In particular, all protocols of the Clark and Jacob library [13] can be modeled as simple processes. However, protocols using abstract channels like private or authenticated channels do not fall in our class. This is also the case of some e-voting protocols that are divided in several phases [17]. This feature can not be modeled in the class of simple processes.

Definition 7 (basic process): The set $\mathcal{B}(c,\mathcal{V})$ of *basic processes* built from $c \in \mathcal{C}h$ and $\mathcal{V} \subseteq \mathcal{X}$ (variables of base type) is the least set of processes that contains 0 and such that

- if $B_1, B_2 \in \mathcal{B}(c,\mathcal{V})$, $M, N, s_1, s_2 \in \mathcal{T}(\mathcal{N}_b, \mathcal{V})$, then
  $$\mathsf{if}\ M = N\ \mathsf{then}\ \mathrm{out}(c,s_1).B_1\ \mathsf{else}\ \mathrm{out}(c,s_2).B_2 \in \mathcal{B}(c,\mathcal{V}).$$
- if $B \in \mathcal{B}(c, \mathcal{V} \uplus \{x\})$, x of base type ($x \notin \mathcal{V}$), then
  $$\mathrm{in}(c,x) \cdot B \; \in \; \mathcal{B}(c,\mathcal{V}).$$

Intuitively, in a basic process, depending on the outcome of the test, the process sends on its channel c a message depending on its inputs. A basic process may also input messages on its channel c.

Example 9: We consider a slightly simplified version of a protocol given in [3] designed for transmitting a secret without revealing its identity to other participants. In this protocol, A is willing to engage in communication with B and wants to reveal its identity to B. However, A does not want to compromise its privacy by revealing its identity or the identity of B more broadly. The participants A and B proceed as follows:

$$\begin{aligned} A \to B \quad &: \quad \mathsf{enc}(\langle N_a, \mathsf{pub}(A)\rangle, \mathsf{pub}(B)) \\ B \to A \quad &: \quad \mathsf{enc}(\langle N_a, \langle N_b, \mathsf{pub}(B)\rangle\rangle, \mathsf{pub}(A)) \end{aligned}$$

First A sends to B a nonce N_a and its public key encrypted with the public key of B. If the message is of the expected

form then B sends to A the nonce N_a, a freshly generated nonce N_b and its public key, all of this being encrypted with the public key of A. Otherwise, B sends out a "decoy" message: $\mathsf{enc}(N_b, \mathsf{pub}(B))$. This message should basically look like B's other message from the point of view of an outsider. This is important since the protocol is supposed to protect the identity of the participants.

A session of role A played by agent a with b can be modeled by the following basic process where true denotes a test that is always satisfied and $M = \mathsf{dec}(x, ska)$. Note that A is not given the value skb but is directly given the value $\mathsf{pk}(skb)$, that is the public key corresponding to B's private key.

$$
\begin{array}{l}
A(a,b) \stackrel{\text{def}}{=} \\
\quad \texttt{if true then} \\
\qquad \mathsf{out}(c_A, \mathsf{enc}(\langle n_a, \mathsf{pk}(ska)\rangle, \mathsf{pk}(skb))). \\
\qquad \mathsf{in}(c_A, x). \\
\qquad \texttt{if } \langle \pi_1(M), \pi_2(\pi_2(M))\rangle = \langle n_a, \mathsf{pk}(skb)\rangle \texttt{ then } 0 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{else } 0 \\
\qquad\quad \texttt{else } 0.
\end{array}
$$

Similarly, a session of role B played by agent b with a can be modeled by the basic process $B(b,a)$ where $N = \mathsf{dec}(y, skb)$.

$$
\begin{array}{rl}
B(b,a) \stackrel{\text{def}}{=} & \mathsf{in}(c_B, y). \\
& \quad \texttt{if } \pi_2(N) = \mathsf{pk}(ska) \texttt{ then} \\
& \mathsf{out}(c_B, \mathsf{enc}(\langle \pi_1(N), \langle n_b, \mathsf{pk}(skb)\rangle\rangle, \mathsf{pk}(ska))).0 \\
& \texttt{else } \mathsf{out}(c_B, \mathsf{enc}(nb, \mathsf{pk}(skb))).0.
\end{array}
$$

Intuitively, this protocol preserves anonymity if an attacker cannot distinguish whether b is willing to talk to a (represented by the process $B(b,a)$) or willing to talk to a' (represented by the process $B(b,a')$), provided a, a' and b are honest participants. For illustration purposes, we also consider the process $B'(b,a)$ obtained from $B(b,a)$ by replacing the else branch by else 0. We will see that the "decoy" message plays a crucial role to ensure privacy.

Definition 8 (simple process): A *simple process* is obtained by composing and replicating basic processes and frames, hiding some names:

$$
\nu\tilde{n}.\ \left(\begin{array}{ccc}
\nu\tilde{n}_1.(B_1 \mid \sigma_1) & \mid & !(\nu c'_1, \tilde{m}_1.\mathsf{out}(p_1, c'_1).B'_1) \mid \\
\vdots & & \vdots \\
\nu\tilde{n}_k.(B_k \mid \sigma_k) & \mid & !(\nu c'_n, \tilde{m}_n.\mathsf{out}(p_n, c'_n).B'_n)
\end{array}\right)
$$

where $B_j \in \mathcal{B}(c_j, \emptyset)$, $B'_j \in \mathcal{B}(c'_j, \emptyset)$ and c_j are channel names that are pairwise distinct. The names $p_1, \ldots, p_n$ are distinct channel names that do not appear elsewhere and $\sigma_1, \ldots, \sigma_k$ are frames without restricted names (i.e. substitutions).

Each basic process B'_j first publishes its channel name c'_j on the public channel p_j so that an attacker can communicate with it. Intuitively the public channels $p_1, \ldots, p_n$ indicate from which role the channel name c'_i is emitted. Names of base types may be shared between processes, this is the purpose of $\tilde{n}$.

It is interesting to notice that protocols with deterministic behavior are usually not modeled within our fragment (see e.g. [2]) since a single channel is used for all communications. We think however that using a single channel does not provide enough information to the attacker since he is not able to schedule exactly the messages to the processes and he does not know from which process a message comes from while this information is usually available (via e.g. IP adresses and session ID). For example, a role emitting the constant a twice would be modeled by $P_1 = \mathsf{out}(c,a).\mathsf{out}(c,a).0$ while two roles emitting each the constant a would be modeled by $P_2 = \mathsf{out}(c,a).0 \mid \mathsf{out}(c,a).0$. Then P_1 and P_2 are observationally equivalent while the two protocols could be distinguished in practice, which is reflected in our modeling in simple processes.

Example 10: Continuing Example 9, a simple process representing unbounded number of sessions in which a plays A (with b) and b plays B with a is:

$$
\begin{array}{ll}
\nu ska, skb. & (\ !(\nu n_a, c_A.\mathsf{out}(p_A, c_A).A(a,b)) \\
& \mid\ !(\nu n_b, c_B.\mathsf{out}(p_B, c_B).B(b,a))\ )
\end{array}
$$

For modelling and verification purposes, we may want to disclose the public keys in order to make them available to the attacker. This can be done by means of an additional basic process

$$K(a,b) = \mathsf{out}(c_K, \mathsf{pk}(ska)) \cdot \mathsf{out}(c_K, \mathsf{pk}(skb)).0.$$

Simple processes is a large class of processes that are determinate. Indeed, since each basic process has its own channel to send and receive messages, all the communications are visible to the attacker. Moreover, the attacker knows exactly who is sending a message or from whom he is receiving a message. Actually, given a simple process A a sequence of actions tr, there is a unique process B (up to some internal reduction steps) such that $A \stackrel{\mathsf{tr}}{\Rightarrow} B$.

Theorem 2: Any simple process is determinate.

Applying Theorems 1 and 2, we get that, on simple processes, it is sufficient to check trace equivalence to prove observational equivalence.

Corollary 1: Let A and B be two simple processes: $A \approx_t B$ if, and only if, $A \approx B$.

V. Intermediate calculus

It is well-known that replication leads to undecidability (see e.g. [19]) thus for the remaining of the paper, we consider processes without replication. We also remove else branches since we are only able to provide a decision procedure in this restricted case (see Section VI). Decidability in presence of else branches is left open. The fragment of simple processes without replication nor else branch still allows to analyze all protocols of the Clark and Jacob library [13], for a bounded number of sessions.

Reasoning on processes of the applied-pi calculus is quite involved since it requires one to consider all the rules defining the labeled transition relation $\xrightarrow{\alpha}$. Thus we use a simplified fragment of the class of *intermediate processes*, defined in [16], that are easier to manipulate and such that trace equivalence of simple processes without replication nor else branch is equivalent to trace equivalence of their corresponding intermediate processes.

A. Syntax

The grammar of the *plain intermediate processes* is as follows:

$$\begin{aligned} P, Q, R &:= 0 \\ &\quad \texttt{if } M_1 = M_2 \texttt{ then } P \texttt{ else } Q \\ &\quad \mathsf{in}(c, x).P \\ &\quad \mathsf{out}(c, N).P \end{aligned}$$

where $c \in \mathcal{C}h$ is channel name, M_1, M_2 are terms of base type, x is a variable of base type, and N is a message of base type. Terms M_1, M_2 and N can also use variables.

Definition 9 (intermediate process): An *intermediate process* is a triple $(\mathcal{E}; \mathcal{P}; \Phi)$ where:

- $\mathcal{E}$ is a set of names that represents the names restricted in $\mathcal{P}$;
- $\Phi = \{w_1 \rhd t_1, \ldots, w_n \rhd t_n\}$ where $t_1, \ldots, t_n$ are ground terms, and $w_1, \ldots, w_n$ are variables;
- $\mathcal{P}$ is a multiset of *plain intermediate processes* (defined above) where null processes are removed and such that $fv(\mathcal{P}) \subseteq \{w_1, \ldots, w_m\}$.

Additionally, we require intermediate processes to be *variable distinct*, i.e. any variable is at most bound once.

Given a sequence $\Phi = \{w_1 \rhd t_1, \ldots, w_n \rhd t_n\}$ where $t_1, \ldots, t_n$ are terms, we also denote by Φ its associated frame, i.e. $\{{}^{t_1}/_{w_1}\} \mid \ldots \mid \{{}^{t_n}/_{w_n}\}$.

Given a closed extended process A of the original applied pi without replication, we can easily transform it into an intermediate process $\tilde{A} = (\mathcal{E}; \mathcal{P}; \Phi)$ such that $A \approx \nu\mathcal{E}.(\mathcal{P} \mid \Phi)$. The idea is to rename names and variables to avoid clashes, to apply the active substitutions (SUBST), to remove the restrictions on variables (ALIAS), and finally to push the restrictions on names in front of the process. We can also add some restricted names not appearing in the process in front of it. This will be useful to obtain two intermediate processes with the same set of restricted names.

Example 11: Consider the extended process A described below (M is a term such that $n \notin fn(M)$):

$$\nu sk.\nu x.(\mathsf{out}(c, \mathsf{enc}(x, \mathsf{pk}(sk))).\nu n.\mathsf{out}(c, n) \mid \{{}^{M}/_{x}\}).$$

An intermediate process A' associated to A is:

$$\begin{aligned} A' &= (\mathcal{E}; \mathcal{P}; \Phi) \\ &= (\{sk, n\}; \mathsf{out}(c, \mathsf{enc}(M, \mathsf{pk}(sk))).\mathsf{out}(c, n); \emptyset). \end{aligned}$$

We have that $A \approx \nu\mathcal{E}.(\mathcal{P} \mid \Phi)$. However, note that A and $\nu\mathcal{E}.(\mathcal{P} \mid \Phi)$ are not in structural equivalence. Indeed, structural equivalence does not allow one to push all the restrictions in front of a process.

B. Semantics

From now on, we consider intermediate processes without else branch, that is we assume that any sub-process of the form $\texttt{if } M = N \texttt{ then } P \texttt{ else } Q$ is such that $Q = 0$. The semantics for intermediate processes (without else branch) is given in Figure 2. Let $\mathcal{A}_i$ be the alphabet of actions for the intermediate semantics. For every $w \in \mathcal{A}_i^*$ the relation $\xrightarrow{w}_i$ on intermediate processes is defined in the usual way. For $s \in (\mathcal{A}_i \setminus \{\tau\})^*$, the relation $\overset{s}{\Rightarrow}_i$ on intermediate processes is defined by: $A \overset{s}{\Rightarrow}_i B$ if, and only if there exists $w \in \mathcal{A}_i^*$ such that $A \xrightarrow{w}_i B$ and s is obtained by erasing all occurrences of τ. Note that by definition, intermediate processes are closed.

C. Equivalence

Let $A = (\mathcal{E}_1; \mathcal{P}_1; \Phi_1)$ be an intermediate process. We define the following set:

$$\mathsf{trace}_i(A) = \{(s, \nu\mathcal{E}_2.\Phi_2) \mid (\mathcal{E}_1; \mathcal{P}_1; \Phi_1) \overset{s}{\Rightarrow}_i (\mathcal{E}_2; \mathcal{P}_2; \Phi_2) \text{ for some } (\mathcal{E}_2; \mathcal{P}_2; \Phi_2)\}$$

Definition 10 ($\approx_t$ for intermediate processes): Let A and B be two intermediate processes having the same set of restricted names, i.e. $A = (\mathcal{E}; \mathcal{P}_1; \Phi_1)$ and $B = (\mathcal{E}; \mathcal{P}_2; \Phi_2)$.

The processes A and B are *intermediate trace equivalent*, denoted by $A \approx_t B$, if for every $(s, \varphi) \in \mathsf{trace}_i(A)$ there exists $(s', \varphi') \in \mathsf{trace}_i(B)$ such that $s = s'$ and $\varphi \sim \varphi'$ (and conversely).

Despite the differences between the two semantics, it can be shown that the two notions of trace equivalence coincide [16]. For intermediate processes derived from simple processes, we wish to obtain a similar result for a more detailed notion of trace, called *annotated trace*.

Annotated traces are obtained by replacing the label τ of the rule THEN$_i$ in Figure 2 with test_p where p is the identity of the process, i.e. the name of its channel. If $A_i \xrightarrow{a_1}_i \ldots \xrightarrow{a_n}_i A_i'$, we denote by $\overline{a_1 \cdot \ldots \cdot a_n}$ the trace obtained from $a_1 \cdot \ldots \cdot a_n$ by replacing any test_p by τ, recovering a trace for the previous definition of trace. We can easily adapt the definition of trace and trace equivalence, yielding to annotated trace and annotated trace equivalence.

We show that on simple processes without else branch nor replication, trace equivalence coincides with annotated trace equivalence.

Proposition 1: Let A and B be two simple processes without else branch nor replication. Let $\tilde{A} = (\mathcal{E}; \mathcal{P}_A; \Phi_A)$ and $\tilde{B} = (\mathcal{E}; \mathcal{P}_B; \Phi_B)$ be the two associated intermediate processes.

The processes A and B are trace equivalent (i.e. $A \approx_t B$ in the original applied pi calculus semantics) if, and only if, $\tilde{A}$ and $\tilde{B}$ are annotated trace equivalent.

The proof relies on the result of [16] that states that two processes are trace equivalent if and only if the corresponding

$$(\mathcal{E}; \{\texttt{if } u = v \texttt{ then } P \texttt{ else } Q\} \uplus \mathcal{P}; \Phi) \xrightarrow{\tau}_i (\mathcal{E}; \{P\} \uplus \mathcal{P}; \Phi) \text{ if } u =_{\mathsf{E}} v \quad (\text{Then}_i)$$

$$(\mathcal{E}; \{\mathsf{in}(p, x).P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\mathsf{in}(p,M)}_i (\mathcal{E}; \{P\{x \mapsto u\}\} \uplus \mathcal{P}; \Phi) \quad (\text{In}_i)$$
$$M\Phi = u,\ fv(M) \subseteq \mathrm{dom}(\Phi) \text{ and } fn(M) \cap \mathcal{E} = \emptyset$$

$$(\mathcal{E}; \{\mathsf{out}(p, u).P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\nu w_n.\mathsf{out}(p,w_n)}_i (\mathcal{E}; \{P\} \uplus \mathcal{P}; \Phi \cup \{w_n \rhd u\}) \quad (\text{Out-T}_i)$$
$$w_n \text{ variable such that } n = |\Phi| + 1$$

u, v and x are terms of base type whereas p is a channel name.

Fig. 2. Intermediate semantics of simple processes

intermediate processes are intermediate trace equivalent. We then need to show that traces can be grouped following the annotation, which is due to the determinism of simple processes.

VI. A DECISION PROCEDURE FOR OBSERVATIONAL EQUIVALENCE

The aim of the section is to provide a decision procedure for trace equivalence and for a large class of processes (namely the class of simple processes), for the class of convergent subterm equational theories. Starting from intermediate processes that are obtained from simple processes without else branch nor replication, we reduce trace equivalence to equivalence of constraint systems. We can then conclude by using the decision procedure proposed in [7], [8] for constraint systems for the class of convergent subterm equational theories.

A. Constraint system

Following the notations of [7], we consider a new set $\mathcal{X}^2$ of variables called *second order variables* $X, Y, \ldots$, each variable with an arity, denoted $ar(X)$. We denote by $\mathsf{var}^1(\mathcal{C})$ (resp. $\mathsf{var}^2(\mathcal{C})$) the first order (resp. second order) variables of $\mathcal{C}$, that is $\mathsf{var}^1(\mathcal{C}) = fv(\mathcal{C}) \cap \mathcal{X}$ (resp. $\mathsf{var}^2(\mathcal{C}) = fv(\mathcal{C}) \cap \mathcal{X}^2$).

A constraint system represents the possible executions of a protocol once an interleaving has been fixed.

Definition 11 (constraint system [7]): A *constraint system* is a triple $(\mathcal{E}; \Phi; \mathcal{C})$:

- $\mathcal{E}$ is a set of names (names that are initially unknown to the attacker);
- Φ is a sequence of the form $\{w_1 \rhd t_1, \ldots, w_n \rhd t_n\}$ where t_i are terms and w_i are variables. The t_i represent the terms sent on the network, their variables represent messages sent by the attacker.
- $\mathcal{C}$ is a set of constraints of the form $X \rhd^? x$ with $ar(X) \leq n$, or of the form $s =^?_{\mathsf{E}} s'$ where s, s' are first-order terms. Intuitively, the constraint $X \rhd^? x$ is meant to ensure that x will be replaced by a deducible term.

The *size* of Φ, denoted $|\Phi|$ is its length n.

We also assume the following conditions:

1) for every $x \in \mathsf{var}^1(\mathcal{C})$, there exists a unique X such that $(X \rhd^? x) \in \mathcal{C}$, and each variable X occurs at most once in $\mathcal{C}$.
2) for every $1 \leq k \leq n$, for every $x \in \mathsf{var}^1(t_k)$, there exists $(X \rhd^? x) \in \mathcal{C}$ such that $ar(X) < k$.

Given a term T with variables $w_1, \ldots, w_k$ and $\Phi = \{w_1 \rhd t_1, \ldots, w_n \rhd t_n\}$, $n \geq k$, $T\Phi$ denotes the term T where each w_i has been replaced by t_i. The *structure* of $(\mathcal{E}; \Phi; \mathcal{C})$ is given by $\mathcal{E}$, $|\Phi|$ and $\mathsf{var}^2(\mathcal{C})$ with their arity.

Example 12: The triple $\Sigma_s = (\mathcal{E}_s; \Phi^0_s \cup \{w_4 \rhd t\}; \mathcal{C}_s)$ where

$\mathcal{E}_s = \{ska, ska', skb, n_a, n_b\}$,
$\Phi^0_s = \{w_1 \rhd \mathsf{pk}(ska), w_2 \rhd \mathsf{pk}(ska'), w_3 \rhd \mathsf{pk}(skb)\}$,
$t = \mathsf{enc}(\langle \pi_1(\mathsf{dec}(y, skb)), \langle n_b, \mathsf{pk}(skb)\rangle\rangle, \mathsf{pk}(ska))$,
$\mathcal{C}_s = \{Y \rhd^? y,\ \pi_2(\mathsf{dec}(y, skb)) =^?_{\mathsf{E}} \mathsf{pk}(ska)\}, ar(Y) = 3$

is a constraint system. We will see that it corresponds to the execution of the process $B'(b, a)$ presented in Example 9. We consider three agents (a, a' and b) so that the attacker can try to learn whether b is willing to talk to a or to a'. Their public keys are made available to the attacker.

Definition 12 (solution): A *solution* of a constraint system $\Sigma = (\mathcal{E}; \Phi; \mathcal{C})$ is a substitution θ such that

- $\mathrm{dom}(\theta) = \mathsf{var}^2(\mathcal{C})$, and
- $X\theta \in \mathcal{T}(\mathcal{N}_b \setminus \{\mathcal{E}\}, \mathrm{dom}(\Phi))$ for any $X \in \mathrm{dom}(\theta)$.

Moreover, we require that there exists a closed substitution λ with $\mathrm{dom}(\lambda) = \mathsf{var}^1(\mathcal{C})$ such that:

1) for every $(X \rhd^? x) \in \mathcal{C}$, $(X\theta)(\Phi\lambda) = x\lambda$;
2) for every $(s =^?_{\mathsf{E}} s') \in \mathcal{C}$, $s\lambda =_{\mathsf{E}} s'\lambda$;

The substitution λ is called *first order solution* of Σ associated to θ. The set of solutions of a constraint system Σ is denoted $Sol(\Sigma)$.

Example 13: Continuing Example 12, a solution to $\Sigma_s = (\mathcal{E}_s; \Phi_s; \mathcal{C}_s)$ is θ where $\mathrm{dom}(\theta) = \{Y\}$ and $\theta(Y) = \mathsf{enc}(\langle n_i, w_1\rangle, w_3)$ with n_i a public name (i.e. $n_i \notin \mathcal{E}_s$). The first order-solution λ of Σ_s associated to θ is a substitution whose domain is $\{y\}$ and such that $\lambda(y) = \mathsf{enc}(\langle n_i, \mathsf{pk}(ska)\rangle, \mathsf{pk}(skb))$.

A constraint system Σ is *satisfiable* if $Sol(\Sigma) \neq \emptyset$. Two constraint systems Σ_1 and Σ_2 with the same structures are *equivalent* if and only if $Sol(\Sigma_1) = Sol(\Sigma_2)$. We further define S-equivalence [7] that will be useful to capture static equivalence.

Definition 13 (S-equivalence): Let $\Sigma_1 = (\mathcal{E}; \Phi_1; \mathcal{C}_1)$ and $\Sigma_2 = (\mathcal{E}; \Phi_2; \mathcal{C}_2)$ be two constraint systems with the same structure and consider $x, y \notin \mathsf{var}^1(\mathcal{C}_i)$ and $X, Y \notin \mathsf{var}^2(\mathcal{C}_i)$ for $i = 1, 2$. The two systems Σ_1 and Σ_2 are *S-equivalent* if the constraint systems:

- $(\mathcal{E}; \Phi_1; \mathcal{C}_1 \cup \{X \rhd^? x,\ Y \rhd^? y,\ x =^?_{\mathsf{E}} y\})$, and
- $(\mathcal{E}; \Phi_2; \mathcal{C}_2 \cup \{X \rhd^? x,\ Y \rhd^? y,\ x =^?_{\mathsf{E}} y\})$

are equivalent.

Example 14: Let Σ'_s be the constraint system below:

$$(\mathcal{E}_s; \Phi^0_s \cup \{w_4 \rhd t'\}; Y \rhd^? y, \pi_2(\mathsf{dec}(y, skb)) =^?_{\mathsf{E}} \mathsf{pk}(ska'))$$

where $t' = \mathsf{enc}(\langle \pi_1(\mathsf{dec}(y, skb)), \langle n_b, \mathsf{pk}(skb)\rangle\rangle, \mathsf{pk}(ska'))$, and $\mathcal{E}_s$, Φ^0_s are defined as in Example 12. We will see that this system corresponds to the system obtained after a symbolic execution of the process $B'(b, a')$ presented in Example 9.

The system Σ_s (given in Example 12) is not equivalent to Σ'_s. Indeed, the substitution θ given in Example 13 is such that $\theta \in Sol(\Sigma_s)$ whereas $\theta \notin Sol(\Sigma'_s)$. We conclude that the constraint systems Σ_s and Σ'_s are not equivalent, and thus not in S-equivalence. Actually, this corresponds to the fact that an attacker can distinguish between $B'(b, a)$ and $B'(b, a')$ by sending a message $\mathsf{enc}(\langle n, \mathsf{pk}(ska)\rangle, \mathsf{pk}(skb))$ and see whether b answers or not.

B. Symbolic calculus

Following the approach of [8], we compute from an intermediate process $P = (\mathcal{E}; \mathcal{P}; \Phi)$ the set of constraints systems capturing the possible executions of P, starting from $P_s \stackrel{\text{def}}{=} (\mathcal{E}; \mathcal{P}; \Phi; \emptyset)$ and applying the rules defined in Figure 3.

Definition 14 (symbolic process): A *symbolic process* is a tuple $(\mathcal{E}; \mathcal{P}; \Phi; \mathcal{C})$ where:

- $\mathcal{E}$ is a set of names;
- $\mathcal{P}$ is a multiset of plain intermediate processes where null processes are removed and such that $fv(\mathcal{P}) \subseteq \{x \mid X \rhd^? x \in \mathcal{C}\}$;
- $(\mathcal{E}, \Phi, \mathcal{C})$ is a constraint system.

The rules of Figure 3 define the semantics of symbolic processes. The aim of this symbolic semantics is to avoid the infinite branching due to the inputs of the environment. This is achieved by keeping variables rather than the input terms. The constraint system gives a finite representation of the value that these variables are allowed to take.

The THEN_s (resp. IN_s) rule allows the process to pass a test (resp. an input). The corresponding constraint is added in the set of constraints $\mathcal{C}$. When a process is ready to output a term on a public channel p, the outputted term is added to the frame Φ, which means that this term is made available to the attacker.

Example 15: We consider one session of the protocol presented in Example 9, in which b plays the role B' (with a) and a plays the role A with b. We consider the following process $K(a, a', b)$ that models keys disclosure, i.e.

$$\text{out}(c_K, \mathsf{pk}(ska)).\text{out}(c_K, \mathsf{pk}(ska')).\text{out}(c_K, \mathsf{pk}(skb)).$$

Let $\mathcal{E}$ be the set of names $\{ska, ska', skb, n_a, n_b\}$, and P^s_{ex} the following symbolic process:

$$P^s_{\mathsf{ex}} = (\mathcal{E}; \{A(a, b),\ B'(b, a),\ K(a, a', b)\}; \emptyset; \emptyset).$$

We have that $P^s_{\mathsf{ex}} \stackrel{\mathsf{tr}}{\Rightarrow}_s (\mathcal{E}_s; \mathcal{P}_s; \Phi_s; \mathcal{C}_s)$ where

- $\mathsf{tr} = \nu w_1.\text{out}(c_K, w_1) \cdot \nu w_2.\text{out}(c_K, w_2) \cdot \nu w_3.\text{out}(c_K, w_3) \cdot \text{in}(c_B, y) \cdot \nu w_4.\text{out}(c_B, w_4)$,
- $\mathcal{P}_s = \{A(a, b)\}$, and
- $(\mathcal{E}_s; \Phi_s; \mathcal{C}_s)$ is the constraint system Σ_s defined in Example 12.

We show that the set of symbolic processes obtained from an intermediate process $(\mathcal{E}; \mathcal{P}; \Phi)$ without else branch exactly captures the set of execution traces of $(\mathcal{E}; \mathcal{P}; \Phi)$ though θ-concretization.

Definition 15 (θ-concretization): Consider the symbolic process $(\mathcal{E}_1; \mathcal{P}_1; \Phi_1; \mathcal{C}_1)$ and let θ be a substitution in $Sol((\mathcal{E}_1; \Phi_1; \mathcal{C}_1))$. The intermediate process $(\mathcal{E}_1; \mathcal{P}_1\lambda_\theta; \Phi_1\lambda_\theta)$ is the *θ-concretization of* $(\mathcal{E}_1; \mathcal{P}_1; \Phi_1; \mathcal{C}_1)$ where λ_θ is the first order solution of $(\mathcal{E}_1; \Phi_1; \mathcal{C}_1)$ associated to θ.

We now show soundness of $\xrightarrow{\alpha_s}_s$ w.r.t. $\xrightarrow{\alpha}_i$: whenever this relation holds between two symbolic processes, the relation in the intermediate semantics holds for each θ-concretization. Actually, we need such a result for the more detailed notion of annotated traces (see page 7): the label τ of the rules THEN_s and THEN_i is replaced by test_p where p is the identity of the process, i.e. the name of its channel.

Proposition 2 (soundness): Let $(\mathcal{E}_1; \mathcal{P}_1; \Phi_1; \mathcal{C}_1)$, $(\mathcal{E}_2; \mathcal{P}_2; \Phi_2; \mathcal{C}_2)$ be two symbolic processes such that

- $(\mathcal{E}_1; \mathcal{P}_1; \Phi_1; \mathcal{C}_1) \xrightarrow{\alpha_s}_s (\mathcal{E}_2; \mathcal{P}_2; \Phi_2; \mathcal{C}_2)$, and
- $\theta_2 \in Sol((\mathcal{E}_2; \Phi_2; \mathcal{C}_2))$.

Let $\theta_1 = \theta_2|_{\mathsf{var}^2(\mathcal{C}_1)}$. We have that:

1) $\theta_1 \in Sol((\mathcal{E}_1; \Phi_1; \mathcal{C}_1))$, and
2) $(\mathcal{E}_1; \mathcal{P}'_1; \Phi'_1) \xrightarrow{\alpha_s\theta_2}_i (\mathcal{E}_2; \mathcal{P}'_2; \Phi'_2)$ where $(\mathcal{E}_1; \mathcal{P}'_1; \Phi'_1)$ (resp. $(\mathcal{E}_2; \mathcal{P}'_2; \Phi'_2)$ is the θ_1-concretization (resp. θ_2) of $(\mathcal{E}_1; \mathcal{P}_1; \Phi_1; \mathcal{C}_1)$ (resp. $(\mathcal{E}_2; \mathcal{P}_2; \Phi_2; \mathcal{C}_2)$).

We also show completeness of the symbolic semantics w.r.t. the intermediate one: each time a θ-concretization of a symbolic process reduces to another intermediate process, the symbolic process also reduces to a corresponding symbolic process.

Proposition 3 (completeness): Let $(\mathcal{E}_1; \mathcal{P}_1; \Phi_1; \mathcal{C}_1)$ be a symbolic process, $(\mathcal{E}_1; \mathcal{P}'_1; \Phi'_1)$ its θ_1-concretization where $\theta_1 \in Sol((\mathcal{E}_1; \Phi_1; \mathcal{C}_1))$. Let $(\mathcal{E}; \mathcal{P}; \Phi)$ be an intermediate process such that $(\mathcal{E}_1; \mathcal{P}'_1; \Phi'_1) \xrightarrow{\alpha}_i (\mathcal{E}; \mathcal{P}; \Phi)$. There exist a symbolic process $(\mathcal{E}_2; \mathcal{P}_2; \Phi_2; \mathcal{C}_2)$ and θ_2 such that:

$$\text{THEN}_s \quad (\mathcal{E};\{\texttt{if } u=v \texttt{ then } P \texttt{ else } 0\}\uplus\mathcal{P};\Phi;\mathcal{C}) \xrightarrow{\tau}_s (\mathcal{E};\{P\}\uplus\mathcal{P};\Phi;\mathcal{C}\cup\{u =^?_{\mathsf{E}} v\})$$

$$\text{IN}_s \quad (\mathcal{E};\{\text{in}(p,x).P\}\uplus\mathcal{P};\Phi;\mathcal{C}) \xrightarrow{\text{in}(p,Y)}_s (\mathcal{E};\{P\{x\mapsto y\}\}\uplus\mathcal{P};\Phi;\mathcal{C}\cup\{Y \rhd^? y\}) \text{ where } Y,y \text{ are fresh variables, } ar(Y)=|\Phi|$$

$$\text{OUT-T}_s \quad (\mathcal{E};\{\text{out}(p,u).P\}\uplus\mathcal{P};\Phi;\mathcal{C}) \xrightarrow{\nu w_n.\text{out}(p,w_n)}_s (\mathcal{E};\{P\}\uplus\mathcal{P};\Phi\cup\{w_n \rhd u\};\mathcal{C}) \text{ where } w_n \text{ is a variable such that } n=|\Phi|+1$$

u, v, and x are terms of base type whereas p is a channel name.

Fig. 3. Symbolic execution of simple processes

1) $(\mathcal{E}_1;\mathcal{P}_1;\Phi_1;\mathcal{C}_1) \xrightarrow{\alpha_s}_s (\mathcal{E}_2;\mathcal{P}_2;\Phi_2;\mathcal{C}_2)$;
2) $\theta_2 \in Sol((\mathcal{E}_2;\Phi_2;\mathcal{C}_2))$;
3) the process $(\mathcal{E};\mathcal{P};\Phi)$ is the θ_2-concretization of $(\mathcal{E}_2;\mathcal{P}_2;\Phi_2;\mathcal{C}_2)$; and
4) $\alpha_s\theta_2=\alpha$.

C. Symbolic equivalence

Definition 16 (symbolic trace equivalence): Let A be a simple process without else branch nor replication. We define the set of its symbolic traces as follows:

$$\mathsf{trace}_s(A) = \{(\mathsf{tr},\Sigma) \mid A_s \overset{\mathsf{tr}}{\Rightarrow}_s (\mathcal{E}';\mathcal{P}';\Phi';\mathcal{C}') \text{ and } \Sigma=(\mathcal{E}';\Phi';\mathcal{C}') \text{ satisfiable.}\}$$

Let A and B be two simple processes. They are in *symbolic trace equivalence* if for every $(\mathsf{tr},\Sigma)\in\mathsf{trace}_s(A)$ there exists $(\mathsf{tr}',\Sigma')\in\mathsf{trace}_s(B)$ such that $\mathsf{tr}=\mathsf{tr}'$ and Σ, Σ' are S-equivalent (and conversely).

We show that symbolic trace equivalence exactly captures trace equivalence.

Proposition 4: Let $A=(\mathcal{E};\mathcal{P}_A;\Phi_A)$ and $B=(\mathcal{E};\mathcal{P}_B;\Phi_B)$ be two intermediate processes derived from simple processes without else branch nor replication. We have that A and B are in annotated trace equivalence if, and only if, they are in annotated symbolic trace equivalence.

The proof relies on the fact that, when $A\approx_t B$, execution traces can be grouped in the same way for A and B, forming symbolic traces with S-equivalent constraint systems.

The following proposition is an immediate consequence of Proposition 1 and Proposition 4.

Proposition 5: Let A and B be two simple processes without else branch nor replication: $A\approx_t B$ if, and only if A and B are in annotated symbolic trace equivalence.

Example 16: Relying on our technique, we can now prove that the two following processes P_{ex} and P'_{ex} are not in observational equivalence:

- $P_{\mathsf{ex}}=\nu\tilde{n}.[\,A(a,b)\mid B'(b,a)\mid K(a,a',b)\,]$, and
- $P'_{\mathsf{ex}}=\nu\tilde{n}.[\,A(a',b)\mid B'(b,a')\mid K(a,a',b)\,]$.

Continuing Example 15, we have that $(\mathsf{tr},\Sigma_s)\in\mathsf{trace}_s(P^s_{\mathsf{ex}})$ and Σ_s satisfiable (see Example 13). The only constraint system reachable from

$$P'^s_{\mathsf{ex}} = (\mathcal{E};\{A(a',b),\ B'(b,a'),\ K(a,a',b)\};\emptyset;\emptyset)$$

by the sequence tr is Σ'_s as defined in Example 14. We have seen that Σ'_s is not in S-equivalence with Σ_s. This allows us to conclude that the simple processes P_{ex} and P'_{ex} are not in symbolic trace equivalence, and thanks to Proposition 5, Theorem 1 and Theorem 2, we conclude that $P_{\mathsf{ex}}\not\approx P'_{\mathsf{ex}}$.

D. Decidability result

It remains to show how to decide symbolic trace equivalence. We mainly rely on the result of [7] that ensures that checking whether two constraints systems are S-equivalent is NP-complete, for the class of convergent subterm theories.

An equational theory E is a *convergent subterm theory* if it is generated by a convergent rewriting system $\mathcal{R}$ such that any rule $l\to r\in\mathcal{R}$ satisfies that either r is a strict subterm of l or r is a closed term in normal form w.r.t. $\mathcal{R}$. The equational theory presented in Example 3 is a convergent subterm theory. Many other examples can be found e.g. in [1].

Now, we are able to state our main result.

Theorem 3: Let E be a subterm convergent equational theory. Let A and B be two simple processes without else branch nor replication. The problem whether A and B are observationally equivalent is co-NP-complete.

The decidability of observational equivalence follows from Proposition 5 since there are a finite number of symbolic traces and non S-equivalence of constraint systems is decidable [7]. Actually, since we consider annotated trace, we have that for any simple process P and any annotated trace tr, there is at most one Σ such that $(\mathsf{tr},\Sigma)\in\mathsf{trace}_s(P)$. We show that two simple processes A and B without else branch nor replication are in trace equivalence if, and only if, for any annotated trace $(\mathsf{tr},\Sigma)\in\mathsf{trace}_s(A)$, there exists a (unique) annotated trace $(\mathsf{tr},\Sigma')\in\mathsf{trace}_s(B)$ such that Σ and Σ' are S-equivalent. We show this result in two steps: we go from applied pi to the intermediate calculus (see Proposition 1) and then we go from intermediate calculus to our symbolic calculus (see Proposition 4).

Then the NP-TIME decision procedure for non observational equivalence works as follows:

- Guess a symbolic (annotated) trace tr;
- Compute (in polynomial time) Σ and Σ' such that $(\mathsf{tr},\Sigma)\in\mathsf{trace}_s(A)$ and $(\mathsf{tr},\Sigma')\in\mathsf{trace}_s(B)$;
- check whether Σ and Σ' are not S-equivalent.

Due to [7], we know that the last step can be done in NP-TIME for convergent subterm theories thus we deduce that the overall procedure is NP-TIME. NP-hardness is obtained using the usual encoding [25].

VII. Conclusion

In this paper, we consider the class of *determinate* processes and we show that observational equivalence actually coincides with trace equivalence, a notion simpler to reason with. We exhibit a large class of processes that are determinate and we show how to reduce the decidability of trace equivalence to deciding an equivalence relation introduced by M. Baudet. Altogether, this yields the first decidability result of observational equivalence for a general class of processes.

As future work, it would be interesting to extend this class of processes in different ways. For example, we would like to extend our decision result to else branches. This would require adding disequality tests in set of constraints and adapt the procedure of [7] accordingly. Moreover, some protocols such as e-voting protocols are divided in several phases. It does not seem difficult to add a "phase" operator to the applied pi-calculus and obtain a corresponding decision result for observational equivalence. It would be also interesting to consider larger classes of equational theories such as those considered for e-voting protocols [17].

Our class of simple processes is close to the fragment of processes considered in [14] for proving cryptographic indistinguishability using observational equivalence. However, the fragment of [14] does not enjoy the determinacy property (since it was not designed for it). We plan to extend their result to our class of simple processes, yielding to a decision technique for proving indistinguishability in cryptographic models.

References

[1] M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 387(1-2):2–32, 2006.

[2] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM Press, 2001.

[3] M. Abadi and C. Fournet. Private authentication. *Theoretical Computer Science*, 322(3):427–476, 2004.

[4] M. Abadi and A. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. 4th Conference on Computer and Communications Security (CCS'97)*, pages 36–47. ACM Press, 1997.

[5] R. Amadio, D. Lugiez, and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290:695–740, 2002.

[6] A. Armando et al. The AVISPA Tool for the automated validation of internet security protocols and applications. In *Proc. 17th Int. Conference on Computer Aided Verification (CAV'05)*, volume 3576 of *LNCS*, pages 281–285. Springer, 2005.

[7] M. Baudet. Deciding security of protocols against off-line guessing attacks. In *Proc. 12th Conference on Computer and Communications Security (CCS'05)*, pages 16–25. ACM Press, 2005.

[8] M. Baudet. *Sécurité des protocoles cryptographiques : aspects logiques et calculatoires*. Phd thesis, École Normale Supérieure de Cachan, France, 2007.

[9] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*, pages 82–96. IEEE Comp. Soc. Press, 2001.

[10] B. Blanchet. Automatic proof of strong secrecy for security protocols. In *Proc. Symposium on Security and Privacy*, pages 86–100. IEEE Comp. Soc. Press, 2004.

[11] B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.

[12] M. Boreale, R. D. Nicola, and R. Pugliese. Proof techniques for cryptographic processes. *SIAM Journal on Computing*, 31(3):947–986, 2002.

[13] J. Clark and J. Jacob. A survey of authentication protocol literature. `http://www.cs.york.ac.uk/~jac/papers/drareviewps.ps`, 1997.

[14] H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. In *Proc. 15th Conference on Computer and Communications Security (CCS'08)*, pages 109–118. ACM Press, 2008.

[15] V. Cortier and S. Delaune. A method for proving observational equivalence. Research Report LSV-09-04, Laboratoire Spécification et Vérification, ENS Cachan, France, Feb. 2009. 22 pages.

[16] S. Delaune, S. Kremer, and M. D. Ryan. Symbolic bisimulation for the applied pi-calculus. In *Proc. 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, pages 133–145, 2007.

[17] S. Delaune, S. Kremer, and M. D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 2009. To appear.

[18] L. Durante, R. Sisto, and A. Valenzano. Automatic testing equivalence verification of spi calculus specifications. *ACM Transactions on Software Engineering and Methodology*, 12(2):222–284, 2003.

[19] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. Workshop on Formal Methods and Security Protocols (FMSP'99)*, Trento (Italy), 1999.

[20] J. Engelfriet. Determinacy implies (observation equivalence = trace equivalence). *Theoretical Computer Science*, 36:21–25, 1985.

[21] H. Hüttel. Deciding framed bisimulation. In *Proc. 4th Int. Workshop on Verification of Infinite State Systems (INFINITY'02)*, pages 1–20, 2002.

[22] P. C. Kanellakis and S. A. Smolka. Ccs expressions, finite state processes, and three problems of equivalence. In ACM, editor, *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 228–240, 1983.

[23] G. Lowe. Towards a completeness result for model checking of security protocols. In *Proc. 11th Computer Security Foundations Workshop (CSFW'98)*, 1998.

[24] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. 8th ACM Conference on Computer and Communications Security (CCS'01)*. ACM Press, 2001.

[25] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*, pages 174–190. IEEE Comp. Soc. Press, 2001.

Symbolic Bisimulation for the Applied Pi Calculus *

Stéphanie Delaune[1] Steve Kremer[1]
Mark D. Ryan[2]

[1]LSV, ENS Cachan & CNRS & INRIA, France
[2]School of Computer Science, University of Birmingham, UK

Abstract

We propose a symbolic semantics for the finite applied pi calculus. The applied pi calculus is a variant of the pi calculus with extensions for modelling cryptographic protocols. By treating inputs symbolically, our semantics avoids potentially infinite branching of execution trees due to inputs from the environment. Correctness is maintained by associating with each process a set of constraints on terms. We define a symbolic labelled bisimulation relation, which is shown to be sound but not complete with respect to standard bisimulation. We explore the lack of completeness and demonstrate that the symbolic bisimulation relation is sufficient for many practical examples. This work is an important step towards automation of observational equivalence for the finite applied pi calculus, e.g. for verification of anonymity or strong secrecy properties.

*This work has been partly supported by the EPSRC projects EP/E029833, *Verifying Properties in Electronic Voting Protocols* and EP/E040829/1, *Verifying Anonymity and Privacy Properties of Security Protocols*, the ARA SESUR project AVOTÉ and the ARTIST2 NoE. Preliminary versions of this paper appeared in [13] and [14].

1 Introduction

The *applied pi calculus* [2] is a derivative of the pi calculus [21] that is specialised for modelling cryptographic protocols. Participants in a protocol are modelled as processes, and the communication between them is modelled by means of channels, names and message passing. The main difference with the pi calculus is that the applied pi calculus allows one to manipulate *complex data*, instead of just names. These data are generated by a term algebra and equality is treated modulo an *equational theory*. For instance the equation $\mathsf{dec}(\mathsf{enc}(x, y), y) = x$ models the fact that encryption and decryption with the same key cancel out in the style of the Dolev-Yao model [16]. Such complex data requires the use of a special kind of processes called *active substitutions*. As an example consider the following process and reduction step:

$$\nu a, k.\mathsf{out}(c, \mathsf{enc}(a, k)).P \xrightarrow{\nu x.out(c,x)} \nu a, k.(P \mid \{^{\mathsf{enc}(a,k)}/_{x}\}).$$

The process outputs a secret name a which has been encrypted with the secret key k on a public channel c. The active substitution $\{^{\mathsf{enc}(a,k)}/_{x}\}$ gives the environment the ability to access the term $\mathsf{enc}(a, k)$ via the fresh variable x without revealing a or k. The applied pi calculus also generalizes the *spi calculus* [3] which only allows a fixed set of built-in primitives (symmetric and public-key encryption), while the applied pi calculus allows one to define a variety of primitives by means of an equational theory.

One of the difficulties in automating the proof of properties of systems is the infinite number of possible behaviours of the attacker, even in the case that the process itself is finite. When the process requests an input from the environment, the attacker can give any term which can be constructed from freely available data and the terms it has learned so far in the protocol, and therefore the execution tree of the process is potentially infinite-branching. To address this problem, researchers have proposed *symbolic abstractions* of processes, in which terms input from the environment are represented as symbolic variables, together with some constraints. These constraints describe the knowledge of the attacker (and therefore, the range of possible values of the symbolic variable) at the time the input was performed.

Reachability properties can be verified by deciding satisfiability of constraint systems resulting from symbolic executions of process algebras (e.g. [20, 4]). Similarly, *off-line guessing attacks* coded as *static equivalence* between process states [5] can be decided using such symbolic executions, but this requires one to check the equivalence of constraint systems, rather than satisfiability. Decision procedures for both satisfiability [11] and equivalence [5] of constraint systems exist for significant families of equational theories. *Observational equivalence properties*, which can be characterized as a bisimulation, express the inability of the attacker to distinguish between two processes no matter how it interacts with them. These properties are useful for modelling anonymity and privacy properties (e.g. [12]), as well as strong secrecy. In the spi calculus [3] properties were actually expressed as a testing relation and bisimulation was used as a proof

technique. Symbolic methods have also been used for bisimulation in process algebras [18, 9]. In particular, Borgström *et al.* [10] define a sound symbolic bisimulation for the spi calculus.

In this paper we propose a symbolic semantics for the applied pi calculus together with a sound symbolic bisimulation. To show that a symbolic bisimulation implies the concrete one, we generally need to prove that the symbolic semantics is both sound and complete. The semantics of the applied pi calculus is not well suited for defining such a symbolic semantics. In particular, we argue at the beginning of Part I that defining a symbolic structural equivalence which is both sound and complete seems impossible. The absence of sound and complete symbolic structural equivalence significantly complicates the proof of our main result. We therefore split it into two parts. We define a more restricted semantics which will provide an *intermediate* representation of applied pi calculus processes (Part I). These intermediate processes are a selected (but sufficient) subset of the original processes. One may think of them as being processes in some kind of normal form. We equip these intermediate processes with a labelled bisimulation that coincides with the original one. Then we present a symbolic semantics which is both sound and complete with respect to the intermediate one and give a sound symbolic bisimulation (Part II).

To keep track of the constraints on symbolic variables we associate a constraint system to each symbolic process. Keeping these constraint systems separate from the process allows us to have a clean division between the bisimulation and the constraint solving part. In particular we can directly build on existing work [5] and obtain a decision procedure for our symbolic bisimulation for a significant family of equational theories whenever the constraint system does not contain disequalities. This corresponds to the fragment of the applied pi calculus without else branches in the conditional. For this fragment, one may also notice that our symbolic semantics can be used to verify reachability properties using the constraint solving techniques from [11]. Another side-effect of the separation between the processes and the constraint system is that we forbid α-conversion on symbolic processes as we lose the scope of names in the constraint system, but allow explicit renaming when necessary (using *naming environments*). We believe that the simplicity of our intermediate calculus (especially the structural equivalence) and the absence of α-conversion is appealing in view of an implementation.

Finally, one may note that as in [10, 8], our technique for deciding bisimulation is incomplete. However, we argue that our technique works for many interesting cases. This is the purpose of Part III. Most of the proofs are in Appendix. Those that are omitted can be found in [15].

2 The Applied Pi Calculus

The applied pi calculus [2] is a language for describing concurrent processes and their interactions. It is based on the pi calculus [21], but is intended to be less pure and therefore more convenient to use. In this paper we only consider the

finite applied pi calculus which does not have replication.

2.1 Syntax and Informal Semantics

To describe processes in the applied pi calculus, one starts with a set of *names* (which are used to name communication channels or other constants), a set of *variables*, and a *signature* Σ which consists of the function symbols which will be used to define terms. In the case of security protocols, typical *function symbols* will include enc for encryption, which takes a plaintext and a key and returns the corresponding ciphertext, and dec for decryption, taking a ciphertext and a key and returning the plaintext (if the decryption key matches the encryption). *Terms* are defined as names, variables, and function symbols applied to other terms. We write $vars(T)$ for the set of variables occurring in T. When $vars(T) = \emptyset$ we say that the term T is *ground*.

We rely on a sort system for terms. It includes a universal base type and a channel type. We denote by $\mathcal{N}$ the set of names and among those names we distinguish the set $\mathcal{N}_{ch}$ of channel names. Similarly, we denote by $\mathcal{X}$ the set of variables. Among those variables, we distinghuish two disjoint sets: $\mathcal{X}_b$ the set of variables of base type and $\mathcal{X}_{ch}$ the set of variables of channel type. Of course function symbol application must respect sorts and arities. Function symbols cannot be applied to variables or names of channel sort, and cannot return terms of that sort, so in fact the only terms of channel sort are variables and names of that sort.

We define the equations which hold on terms constructed from the signature as an *equational theory* E. We denote $=_{\mathsf{E}}$ the equivalence relation induced by E.

Example 2.1 *A typical example of an equational theory is defined by the equation* $\mathsf{dec}(\mathsf{enc}(x,k),k) = x$. *Let* $T_1 = \mathsf{dec}(\mathsf{enc}(\mathsf{enc}(n,k_1),k_2),k_2)$ *and* $T_2 = \mathsf{enc}(n,k_1)$. *We have that* $T_1 =_{\mathsf{E}} T_2$ *(while obviously the syntactic equality* $T_1 = T_2$ *does not hold).*

In the applied pi calculus, one has *plain processes*, denoted by P, Q, R and *extended processes*, denoted by A, B, C. Plain processes are built up in a similar way to processes in the pi calculus, except that messages can contain terms (rather than just names). In the grammar described below, M and N are terms, n is a name, x a variable and u is a metavariable, standing either for a name or a variable. Extended processes add *active substitutions*, and restriction on names and variables.

$P, Q, R :=$	plain processes	$A, B, C :=$	extended processes
	0		P
	$P \mid Q$		$A \mid B$
	$\nu n.P$		$\nu n.A$
	if $M = N$ then P else Q		$\nu x.A$
	$\mathsf{in}(u,x).P$		$\{^M/_x\}$
	$\mathsf{out}(u,N).P$		

The substitution $\{^M/_x\}$ replaces the variable x with the term M (x and M have the same sort which is required to be a base sort). Active substitutions generalise "let". The process $\nu x.(\{^M/_x\} \mid P)$ corresponds exactly to the process "let $x = M$ in P". As usual, names and variables have scopes, which are delimited by restrictions and by inputs. We write $fv(A)$, $bv(A)$, $fn(A)$ and $bn(A)$ for the sets of *free* and *bound variables* and *free* and *bound names* of A, respectively. In an extended process, there is at most one substitution for each variable, and there is exactly one when the variable is restricted. We say that an extended process is *closed* if all its variables are either bound or defined by an active substitution. We also allow the usual abuse of notations: we omit trailing 0 processes and "else 0" branches in conditionals and write $\nu u_1, u_2, \ldots, u_n$ instead of $\nu u_1.\nu u_2.\ \ldots\ .\nu u_n$.

Active substitutions are useful because they allow us to map an extended process A to its *frame* $\phi(A)$ by replacing every plain process in A with 0. A frame is an extended process built up from 0 and active substitutions by parallel composition and restriction. The frame $\phi(A)$ can be viewed as an approximation of A that accounts for the static knowledge A exposes to its environment, but not A's dynamic behaviour. The *domain* of a frame φ denoted by $\mathrm{dom}(\varphi)$, is the set of variables for which φ defines a substitution (those variables x for which φ contains a substitution $\{^M/_x\}$ not under a restriction on x). We also define the domain of an extended process A, written $\mathrm{dom}(A)$ to be the domain of its frame, i.e., $\mathrm{dom}(A) = \mathrm{dom}(\phi(A))$.

An *evaluation context* $C[\_]$ is an extended process with a hole instead of an extended process.

2.2 Semantics

The operational semantics of processes in the applied pi calculus is defined by structural rules defining two relations: *structural equivalence* and *internal reduction*.

Structural equivalence, noted $\equiv$, is the smallest equivalence relation on extended processes that is closed under α-conversion on names and variables, application of evaluation contexts, and such that:

Par-0	$A \mid 0$	$\equiv$	A	
Par-A	$A \mid (B \mid C)$	$\equiv$	$(A \mid B) \mid C$	
Par-C	$A \mid B$	$\equiv$	$B \mid A$	
New-0	$\nu n.0$	$\equiv$	0	
New-C	$\nu u.\nu v.A$	$\equiv$	$\nu v.\nu u.A$	
New-Par	$A \mid \nu u.B$	$\equiv$	$\nu u.(A \mid B)$	if $u \notin fn(A) \cup fv(A)$
Alias	$\nu x.\{^M/_x\}$	$\equiv$	0	
Subst	$\{^M/_x\} \mid A$	$\equiv$	$\{^M/_x\} \mid A\{^M/_x\}$	
Rewrite	$\{^M/_x\}$	$\equiv$	$\{^N/_x\}$	if $M =_{\mathsf{E}} N$

We also define $=_\alpha$ for equality closed under α-renaming.

Example 2.2 *Let* $P = \nu s, k.(\mathsf{out}(c_1, \mathsf{enc}(s,k)) \mid \mathsf{in}(c_1, y).\mathsf{out}(c_2, \mathsf{dec}(y,k)))$. *The first component publishes the message* $\mathsf{enc}(s,k)$ *by sending it on* c_1. *The second one receives a message on* c_1, *uses the secret key* k *to decrypt it, and forwards the resulting plaintext on* c_2. *The process* P *is structurally equivalent to the following extended process* A*:*

$$A = \nu s, k, x_1.(\mathsf{out}(c_1, x_1) \mid \mathsf{in}(c_1, y).\mathsf{out}(c_2, \mathsf{dec}(y,k)) \mid \{^{\mathsf{enc}(s,k)}/_{x_1}\})$$

We have $\phi(A) = \nu s, k, x_1.\{^{enc(s,k)}/_{x_1}\} \equiv 0$ *(since* x_1 *is under a restriction).*

As already noted in [2], any closed frame is structurally equivalent to a sequence of active substitutions under some restricted names $\nu\tilde{n}.\{^{M_1}/_{x_1}\} \mid \ldots \mid \{^{M_k}/_{x_k}\}$. Therefore, we sometimes refer to such a frame as $\nu\tilde{n}.\sigma$ where σ is the substitution of terms for variables obtained by taking the union of the active substitutions.

Internal reduction, noted $\rightarrow$, is the smallest relation on extended processes closed under structural equivalence and application of evaluation contexts such that

$$\begin{array}{lrcll}
\textsc{Comm} & \mathsf{out}(a,M).P \mid \mathsf{in}(a,x).Q & \rightarrow & P \mid Q\{^{M}/_{x}\} & \\
\textsc{Then} & \text{if } M = N \text{ then } P \text{ else } Q & \rightarrow & P & \text{where } M =_{\mathsf{E}} N \\
\textsc{Else} & \text{if } M = N \text{ then } P \text{ else } Q & \rightarrow & Q & \\
\multicolumn{5}{r}{\text{for any ground terms } M \text{ and } N \text{ such that } M \neq_{\mathsf{E}} N}
\end{array}$$

Note that the presentation of the communication rule (Comm) slightly differs from the one given in [2], but our presentation is easily shown to be equivalent to theirs. The above presentation is closer to our symbolic semantics and therefore more convenient for the purpose of this paper. Comparisons (Then and Else) are kept unchanged and directly depend on the underlying equational theory; using Else sometimes requires that active substitutions in the context be applied first, to yield ground terms M and N. Terms M and N are required to be ground in the rule Else because disequality is not stable under substitution of terms for variables, unlike equality which explains the absence of this condition in the Then rule.

The operational semantics is extended by a *labelled* operational semantics enabling us to reason about processes that interact with their environment. Labelled operational semantics defines the relation $\xrightarrow{\alpha}$ where α is either $\mathsf{in}(a,M)$ (a is a channel name and M is a term that can contain names and variables), or $\nu x.\mathsf{out}(a,x)$ (x is variable of base type), or $\mathsf{out}(a,c)$ or $\nu c.\mathsf{out}(a,c)$ (c is a channel name). We adopt the following rules in addition to the internal reduction rules:

IN $$\mathsf{in}(a,x).P \xrightarrow{in(a,M)} P\{^M/_x\}$$

OUT-CH $$\mathsf{out}(a,c).P \xrightarrow{out(a,c)} P$$

OPEN-CH $$\frac{A \xrightarrow{out(a,c)} A' \qquad c \neq a}{\nu c.A \xrightarrow{\nu c.out(a,c)} A'}$$

OUT-T $$\mathsf{out}(a,M).P \xrightarrow{\nu x.out(a,x)} P \mid \{^M/_x\} \quad x \notin fv(P) \cup fv(M)$$

SCOPE $$\frac{A \xrightarrow{\alpha} A' \qquad u \text{ does not occur in } \alpha}{\nu u.A \xrightarrow{\alpha} \nu u.A'}$$

PAR $$\frac{A \xrightarrow{\alpha} A' \quad bv(\alpha) \cap fv(B) = bn(\alpha) \cap fn(B) = \emptyset}{A \mid B \xrightarrow{\alpha} A' \mid B}$$

STRUCT $$\frac{A \equiv B \quad B \xrightarrow{\alpha} B' \quad A' \equiv B'}{A \xrightarrow{\alpha} A'}$$

Example 2.3 *Consider the process P defined in Example 2.2. We have*

$$\begin{array}{rl} P & \nu s,k.(\mathsf{in}(c_1,y).\mathsf{out}(c_2,\mathsf{dec}(y,k)) \mid \{\mathsf{enc}(s,k)/x_1\}) \\ \xrightarrow{in(c_1,x_1)} & \nu s,k.(\mathsf{out}(c_2,\mathsf{dec}(x_1,k)) \mid \{\mathsf{enc}(s,k)/x_1\}) \\ \xrightarrow{\nu x_2.out(c_1,x_2)} & \nu s,k.(\{\mathsf{enc}(s,k)/x_1\} \mid \{\mathsf{dec}(x_1,k)/x_2\}) \end{array}$$

Let B be the extended process obtained after this sequence of reduction steps. We have that $\phi(B) \equiv \nu s,k.\{^{\mathsf{enc}(s,k)}/_{x_1}, ^s/_{x_2}\}$.

Our rules differ slightly from those described in [2]. Our rules OUT-CH and OPEN-CH can be used only when c is a channel name, whereas in [2] there are identical rules OUT-ATOM and OPEN-ATOM which can be used to output a channel name or a variable of base type. To handle variables of base types, we have the rule OUT-T instead. OUT-T can easily be derived from the rules in [2], and, conversely, any application of OUT-ATOM or OPEN-ATOM involving a variable of base type can be replaced by an application of OUT-T, though the label $out(c,x)$ will be replaced by a label $\nu y.out(c,y)$ where y is a fresh variable not appearing in the process. Any transition in our semantics is also a transition in [2], but they also allow output of free variables directly. For example, notice that in [2], the process $\mathsf{out}(c,M).P \mid \{^M/_x\}$ can transition by label $\nu y.out(c,y)$ to $P \mid \{^M/_x\} \mid \{^M/_y\}$, or by label $out(c,x)$ to $P \mid \{^M/_x\}$ (since $\mathsf{out}(c,M).P \mid \{^M/_x\} \equiv \mathsf{out}(c,x).P \mid \{^M/_x\}$). Our semantics only allows the former transition. In [15], we prove that labelled bisimulation in our system coincides with labelled bisimulation in [2].

2.3 Equivalences

We can now define what it means for two frames to be statically equivalent [2].

Definition 2.4 (static equivalence ($\sim$)) *Two closed frames φ_1 and φ_2 are* statically equivalent*, written $\varphi_1 \sim \varphi_2$, if and only if for some names $\tilde{n}_1$, $\tilde{n}_2$ and substitutions σ_1, σ_2, such that $\varphi_1 \equiv \nu\tilde{n}_1\sigma_1$, $\varphi_2 \equiv \nu\tilde{n}_2\sigma_2$, and $\mathrm{dom}(\sigma_i) \cap vars(\mathrm{img}(\sigma_i)) = \emptyset$ for $i = 1, 2$, we have*

(i) $\mathrm{dom}(\varphi_1) = \mathrm{dom}(\varphi_2)$,

(ii) for all terms M, N with variables included in $\mathrm{dom}(\varphi_i)$ and using no names occurring in $\tilde{n}_1$ or $\tilde{n}_2$, $M\sigma_1 =_{\mathsf{E}} N\sigma_1$ is equivalent to $M\sigma_2 =_{\mathsf{E}} N\sigma_2$.

Extended processes A and B are *statically equivalent*, noted by $A \sim B$, if we have that $\phi(A) \sim \phi(B)$.

Example 2.5 *Let $\varphi_0 = \nu k.\sigma_0$ and $\varphi_1 = \nu k.\sigma_1$ where $\sigma_0 = \{\mathsf{enc}(s_0, k)/x_1, k/x_2\}$, $\sigma_1 = \{\mathsf{enc}(s_1, k)/x_1, k/x_2\}$ and s_0, s_1 and k are names. Let E be the theory defined by the axiom $\mathsf{dec}(\mathsf{enc}(x, k), k) = x$. We have $\mathsf{dec}(x_1, x_2)\sigma_0 =_{\mathsf{E}} s_0$ but not $\mathsf{dec}(x_1, x_2)\sigma_1 =_{\mathsf{E}} s_0$. Therefore we have $\varphi_0 \not\sim \varphi_1$. However, note that we have $\nu k.\{\mathsf{enc}(s_0, k)/x_1\} \sim \nu k.\{\mathsf{enc}(s_1, k)/x_1\}$.*

Definition 2.6 (labelled bisimilarity ($\approx$)) *Labelled bisimilarity is the largest symmetric relation $\mathcal{R}$ on closed extended processes, such that $A\ \mathcal{R}\ B$ implies*

1. *$A \sim B$,*
2. *if $A \to A'$, then $B \to^* B'$ and $A'\ \mathcal{R}\ B'$ for some B',*
3. *if $A \xrightarrow{\alpha} A'$ and $fv(\alpha) \subseteq \mathrm{dom}(A)$ and $bn(\alpha) \cap fn(B) = \emptyset$, then $B \to^* \xrightarrow{\alpha} \to^* B'$ and $A'\ \mathcal{R}\ B'$ for some B'.*

The definition of labelled bisimilarity is like the usual definition of bisimilarity, except that at each step one additionally requires that the processes are statically equivalent. In [2], it is shown that labelled bisimilarity coincides with *observational equivalence*, which is a relation capturing the fact that two given processes cannot be distinguished by any context. In general, it is easier to work with labelled bisimilarity rather than observational equivalence because of the quantification over all contexts. As mentioned in the introduction, contextually defined equivalences are useful to formalize many security properties, in particular anonymity properties, such as those studied in [12].

— PART I: Intermediate Calculus —

The idea of a symbolic semantics is to have a notion of process in which terms that have been input from the environment are represented as variables. This allows us to reason in a way that abstracts away from the particular term that was input. Given such a process P_s, we can create a concrete process (an "instance") $P_s\sigma$ by applying a substitution σ that maps the input variables to terms. We define the symbolic semantics by means of counterparts $\equiv_s$, $\rightarrow_s$, $\xrightarrow{\alpha}_s$ for the concrete relations $\equiv$, $\rightarrow$, $\xrightarrow{\alpha}$ of applied pi calculus, and aim to show soundness and completeness results relating the symbolic and concrete semantics. Structural equivalence occupies a crucial role in our calculi because the transition relations are closed under structural equivalence; therefore, we would ideally like a notion of symbolic structural equivalence which is sound and complete in the following (informal) sense:

Soundness: $P_s \equiv_s Q_s$ implies for any valid instantiation σ, $P_s\sigma \equiv Q_s\sigma$;
Completeness: $P_s\sigma \equiv Q$ implies there exists Q_s s.t. $P_s \equiv_s Q_s$ and $Q_s\sigma = Q$.

Unfortunately, completeness in this sense appears to be unachievable. To see this, consider the following example:

Example 2.7 *Consider the following process:*

$$P = \mathsf{in}(c, x).\mathsf{in}(c, y).\mathsf{out}(c, f(x)).\mathsf{out}(c, g(y)).$$

The process P can be reduced to $P' = \mathsf{out}(c, f(M_1)).\mathsf{out}(c, g(M_2))$ where M_1 and M_2 are two arbitrary terms provided by the environment. In the case that $f(M_1) =_{\mathsf{E}} g(M_2)$ we have $P' \equiv \nu z.(\mathsf{out}(c, z).\mathsf{out}(c, z) \mid \{^{f(M_1)}/_z\})$, but this structural equivalence does not hold whenever $f(M_1) \neq_{\mathsf{E}} g(M_2)$. The aim of symbolic semantics is to avoid instantiating the variables x and y; the process P would reduce symbolically to $P'_s = \mathsf{out}(c, f(x)).\mathsf{out}(c, g(y))$. In this case we need to keep auxiliary information that allows us to infer that x and y may take arbitrary values. The process P'_s represents the two cases in which x and y are equal or distinct. Hence, the question of whether the symbolic structural equivalence $P'_s \equiv_s \nu z.(\mathsf{out}(c, z).\mathsf{out}(c, z) \mid \{^{f(x)}/_z\})$ is valid cannot be decided, as it depends on the concrete values of x and y.

Therefore, the notion of symbolic structural equivalence that we will introduce is sound but not complete in the sense above (we will give a weaker completeness result). This seems to be an inherent problem and it propagates to internal and labelled reduction, since they are closed under structural equivalence. In Example 2.7, the control flow is not affected by whether $f(x) =_{\mathsf{E}} g(y)$. When control flow is affected by conditions on input variables, we maintain those conditions as a set of constraints. This allows us to give a soundness result for symbolic labelled bisimulation.

Unfortunately, the fact that we are unable to have a notion of symbolic structural equivalence which is both sound and complete in the sense mentioned above

significantly complicates the proof of our main result. We therefore split it into two parts. In this part, we define a more restricted semantics which will provide an *intermediate* representation of applied pi calculus processes (Section 3). These intermediate processes are a selected (but sufficient) subset of the original processes. One may think of them as being processes in some kind of normal form. We equip these intermediate processes with a labelled bisimulation that coincides with the original one (Section 5).

3 Syntax and Semantics

3.1 Syntax

One has *intermediate plain processes* (denoted by P, Q, R), *intermediate framed processes* (F, G, H), and *intermediate extended processes* (A, B, C).

$$
\begin{array}{lll}
P, Q, R := & \text{inter. plain process} \\
\quad 0 & \\
\quad P \mid Q & \\
\quad \text{if } M = N \text{ then } P \text{ else } Q & \\
\quad \mathsf{in}(u, x).P & \\
\quad \mathsf{out}(u, N).P &
\end{array}
\qquad
\begin{array}{lll}
F, G, H := & \text{inter. framed process} \\
\quad P & \\
\quad \{^{M}/_{x}\} & \\
\quad F \mid G & \\
A, B, C := & \text{inter. extended process} \\
\quad F & \\
\quad \nu n.A &
\end{array}
$$

Additionally, we require intermediate extended processes to be

- *name and variable distinct* (nv-distinct): $bn(A) \cap fn(A) = bv(A) \cap fv(A) = \emptyset$ and any name and variable is at most bound once; and
- *applied*, meaning that each variable in $\mathrm{dom}(A)$ occurs only once in A (the occurrence in the substitution is the only one).

Intuitively, an intermediate process is applied if all active substitutions have been applied. Intermediate extended processes are a kind of normal form for extended processes. Because they are applied and nv-distinct, we do not need restriction νx on variables x, and all νn for names n occur at the beginning of the process (which is possible as our language does not have replication).

Example 3.1 *The extended process* $\mathsf{out}(c, x) \mid \{^{M}/_{x}\}$ *is not applied, as* x *occurs twice. The corresponding intermediate process would be* $\mathsf{out}(c, M) \mid \{^{M}/_{x}\}$.

As expected, an *intermediate context* is an intermediate extended process with a hole and similarly an intermediate framed context is an intermediate framed process with a hole. An *intermediate (framed) evaluation context* is a (framed) context whose hole is not under a conditional, an input or an output. We say that such a context $C[\_]$ is a context w.r.t. an extended intermediate process A if and only if $C[A]$ is an extended intermediate process. For instance, the context $\nu n.(B \mid \_)$ would not be a context for any A such that $n \in fn(A) \cup bn(A)$ as $\nu n.(B \mid A)$ would violate nv-distinctness.

As we do not allow α-conversion we explicitly run intermediate extended processes in a *naming environment*.

Definition 3.2 (naming environment) *A* naming environment

$$\mathsf{N} : \mathcal{N} \cup \mathcal{X} \to \{\mathsf{n}, \mathsf{f}, \mathsf{b}\}$$

is a function which maps each name and variable to one of n, f, b *(standing for "new", "free" and "bound" respectively), such that there are infinitely many names and infinitely many variables that are mapped to each of* n, f *and* b. *(More precisely, the sets* $\mathsf{N}^{-1}(\mathsf{n}) \cap \mathcal{X}$, $\mathsf{N}^{-1}(\mathsf{n}) \cap \mathcal{N}$, $\mathsf{N}^{-1}(\mathsf{f}) \cap \mathcal{X}$, $\mathsf{N}^{-1}(\mathsf{f}) \cap \mathcal{N}$, $\mathsf{N}^{-1}(\mathsf{b}) \cap \mathcal{X}$, *and* $\mathsf{N}^{-1}(\mathsf{b}) \cap \mathcal{N}$ *are all infinite.)*

Intuitively, $\mathsf{N}(u) = \mathsf{f}$ if the name or variable u occurs *free* in A, and $\mathsf{N}(u) = \mathsf{b}$ if the name or variable u has been *bound* and will not be used again. $\mathsf{N}(u) = \mathsf{n}$ means that the name or variable is *new* and has not been used before, either as free or bound. This discipline helps us avoid name and variable conflicts. We use standard notation for function updating: if $\mathsf{N}(u) = t$ then the naming environment $\mathsf{N}' = \mathsf{N}[u \mapsto t']$ is defined to be the same as N except that $\mathsf{N}'(u) = t'$; and $\mathsf{N}[U \mapsto t']$ is defined as $\mathsf{N}[u_1 \mapsto t', \ldots, u_n \mapsto t']$ if $U = \{u_1, \ldots, u_n\}$. If U is a set of names and variables then $\mathsf{N}(U) = \{\mathsf{N}(u) \mid u \in U\}$ and we write $\mathsf{N}(U) = t$ if $\mathsf{N}(U) \subseteq \{t\}$.

Definition 3.3 (compatible) *We say that a naming environment* N *is* compatible with an nv-distinct process A *if* $\mathsf{N}(bn(A) \cup bv(A)) = \mathsf{b}$ *and* $\mathsf{N}(fn(A) \cup fv(A)) = \mathsf{f}$. *A naming environment* N *is* compatible with a label α *if* $\mathsf{N}(bn(\alpha) \cup bv(\alpha)) = \mathsf{n}$ *and* $\mathsf{N}(fn(\alpha) \cup fv(\alpha)) = \mathsf{f}$.

Requesting that $\mathsf{N}(bn(\alpha) \cup bv(\alpha)) = \mathsf{n}$ may seem strange with respect to the original semantics. However, the intermediate semantics, that we present in the following subsection, should clarify this point.

We define an *intermediate process* to be a pair $(A \,;\, \mathsf{N})$ where A is an intermediate extended process and N a naming environment, compatible with A. Moreover, $(A \,;\, \mathsf{N})$ is closed if A is closed. We denote by $\psi(A)$ the substitution obtained by taking the union of the active substitutions $\{^{M}/_{x}\}$ occurring in A. Note that $\phi(A)$ denotes the frame of the process including the name restrictions, while $\psi(A)$ only refers to the substitution.

Throughout the paper we always suppose that substitutions are cycle-free and use the following notational conventions for substitution. Given substitutions $\sigma_1 = \{^{M_1}/_{x_1}, \ldots, ^{M_p}/_{x_p}\}$ and $\sigma_2 = \{^{N_1}/_{y_1}, \ldots, ^{N_q}/_{y_q}\}$ we write $\sigma_1 \cup \sigma_2$ for $\{^{M_1}/_{x_1}, \ldots, ^{M_p}/_{x_p}, ^{N_1}/_{y_1}, \ldots, ^{N_q}/_{y_q}\}$ and $\sigma_1\sigma_2$ for $\{^{M_1\sigma_2}/_{x_1}, \ldots, ^{M_p\sigma_2}/_{x_p}\}$. We define $\mathrm{img}(\sigma)$ to be the image of σ, e.g., $\mathrm{img}(\sigma_1) = \{M_1, \ldots, M_p\}$. Moreover, we write $\sigma^\star$ to emphasize that we iterate the substitution until obtaining idempotence. This is needed when $\mathrm{dom}(\sigma) \cap \mathit{vars}(\mathrm{img}(\sigma)) \neq \emptyset$.

We now define the $\downarrow$ operator which transforms an nv-distinct extended processes into an intermediate extended process.

Definition 3.4 ($A\downarrow$) *Given an nv-distinct extended process A, the intermediate extended process $A\downarrow$ is defined inductively as follows:*

$$\begin{array}{ccc} 0\downarrow = 0 & \mathsf{in}(u,x).P\downarrow = \nu\tilde{n}.\mathsf{in}(u,x).P' & (\nu n.A)\downarrow = \nu n.(A\downarrow) \\ \{^M/_x\}\downarrow = \{^M/_x\} & \mathsf{out}(u,N).P\downarrow = \nu\tilde{n}.\mathsf{out}(u,N).P' & (\nu x.A)\downarrow = \tilde{A} \end{array}$$

$$\begin{array}{rcl} \textit{if } M = N \textit{ then } P \textit{ else } Q\downarrow & = & \nu\tilde{n}.\nu\tilde{m}.\textit{if } M = N \textit{then } P' \textit{ else } Q' \\ (A \mid B)\downarrow & = & \nu\tilde{n}.\nu\tilde{m}.(A' \mid B')(\psi(A') \cup \psi(B'))^\star \end{array}$$

where $P\downarrow = \nu\tilde{n}.P'$, $Q\downarrow = \nu\tilde{m}.Q'$, $A\downarrow = \nu\tilde{n}.A'$, $B\downarrow = \nu\tilde{m}.B'$, and $\tilde{A}$ is $A\downarrow$ but with the unique occurrence of $\{^M/_x\}$ replaced by 0.

The transformation $\downarrow$ consists of:

1. applying active substitutions as much as possible and allows us to get rid of restrictions on variables. This operation preserves structural equivalence since the rules SUBST and ALIAS allows us to do this.

2. pushing the restrictions on names in front of the process. This operation does not preserve structural equivalence (see Example 3.5) but only labelled bisimilarity (Lemma 3.6).

This operation is extended as expected to context. If $C[\_]$ is an evaluation context, then $C[\_]\downarrow$ is the intermediate evaluation context obtained by applying the above rules, with the additional rule $\_\downarrow = \_$.

Example 3.5 *Let $A = \nu x.(in(c,y).\nu b_1.out(a,b_1).out(a,x) \mid \{^{f(b_2)}/_x\})$. We have that*

$$A\downarrow = \nu b_1.(in(c,y).out(a,b_1).out(a,f(b_2)) \mid 0$$

Lemma 3.6 *Let A be an nv-distinct extended process. We have that $A\downarrow \approx A$.*

3.2 Semantics

Structural equivalence $\equiv_i$ is the smallest equivalence relation on intermediate processes such that:

$$\begin{array}{lrcl} \textsc{Par}\text{-}0_i & (A \;;\mathsf{N}) & \equiv_i & (A \mid 0 \;;\mathsf{N}) \\ \textsc{Par}\text{-}\mathrm{A}_i & (A \mid (B \mid C) \;;\mathsf{N}) & \equiv_i & ((A \mid B) \mid C \;;\mathsf{N}) \\ \textsc{Par}\text{-}\mathrm{C}_i & (A \mid B \;;\mathsf{N}) & \equiv_i & (B \mid A \;;\mathsf{N}) \\ \textsc{New}\text{-}\mathrm{C}_i & (\nu n.\nu m.A \;;\mathsf{N}) & \equiv_i & (\nu m.\nu n.A \;;\mathsf{N}) \end{array}$$

and that is closed by application of intermediate evaluation context, i.e.

$$\frac{(A \;;\mathsf{N}) \equiv_i (B \;;\mathsf{N})}{(C[A] \;;\mathsf{N}[bn(C[0]) \mapsto \mathsf{b}]) \equiv_i (C[B] \;;\mathsf{N}[bn(C[0]) \mapsto \mathsf{b}])}$$

Note that, in this intermediate semantics, we have removed several structural equivalence rules such as SUBST, REWRITE and we do not allow α-renaming. In particular, Example 2.7 is not problematic anymore.

Internal reduction $\rightarrow_i$ is the smallest relation on intermediate processes closed by structural equivalence ($\equiv_i$), by application of intermediate evaluation contexts and such that:

$$\begin{array}{lrcll}
\textsc{Comm}_i & (\mathsf{out}(a,M).P \mid \mathsf{in}(a,x).Q \;;\; \mathsf{N}) & \rightarrow_i & (P \mid Q\{^M/_x\} \;;\; \mathsf{N}) & \\
\textsc{Then}_i & (\text{if } M = N \text{ then } P \text{ else } Q \;;\; \mathsf{N}) & \rightarrow_i & (P \;;\; \mathsf{N}) & \text{where } M =_{\mathsf{E}} N \\
\textsc{Else}_i & (\text{if } M = N \text{ then } P \text{ else } Q \;;\; \mathsf{N}) & \rightarrow_i & (Q \;;\; \mathsf{N}) & \\
 & & & \text{for any ground terms } M \text{ and } N \text{ such that } M \neq_{\mathsf{E}} N &
\end{array}$$

Labelled transition $\xrightarrow{\alpha}_i$. We also extend our intermediate semantics with the following labelled transition relation.

$$\textsc{In}_i \qquad (\mathsf{in}(a,x).P \;;\; \mathsf{N}) \xrightarrow{in(a,M)}_i (P\{^M/_x\} \;;\; \mathsf{N}) \quad \text{where } \mathsf{N}(fn(M) \cup fv(M)) = \mathsf{f}$$

$$\textsc{Out-Ch}_i \qquad (\mathsf{out}(a,c).P \;;\; \mathsf{N}) \xrightarrow{out(a,c)}_i (P \;;\; \mathsf{N})$$

$$\textsc{Out-T}_i \qquad (\mathsf{out}(a,M).P \;;\; \mathsf{N}) \xrightarrow{\nu x.out(a,x)}_i (P \mid \{^M/_x\}, \mathsf{N}[x \mapsto \mathsf{f}]) \quad \text{where } x \in \mathcal{X}_b \text{ and } \mathsf{N}(x) = \mathsf{n}$$

$$\textsc{Open-Ch}_i \qquad \frac{(A \;;\; \mathsf{N}) \xrightarrow{out(a,c)}_i (A' \;;\; \mathsf{N}') \qquad c \neq a,\ d \in \mathcal{N}_{ch},\ \mathsf{N}(d) = \mathsf{n}}{(\nu c.A, \mathsf{N}[c \mapsto \mathsf{b}]) \xrightarrow{\nu d.out(a,d)}_i (A'\{^d/_c\}, \mathsf{N}'[c \mapsto \mathsf{b}, d \mapsto \mathsf{f}])}$$

$$\textsc{Scope}_i \qquad \frac{(A \;;\; \mathsf{N}) \xrightarrow{\alpha}_i (A', \mathsf{N}') \qquad n \text{ does not occur in } \alpha}{(\nu n.A \;;\; \mathsf{N}[n \mapsto \mathsf{b}]) \xrightarrow{\alpha}_i (\nu n.A', \mathsf{N}'[n \mapsto \mathsf{b}])}$$

$$\textsc{Par}_i \qquad \frac{(A \;;\; \mathsf{N}) \xrightarrow{\alpha\psi(B)}_i (A', \mathsf{N}')}{(A \mid B \;;\; \mathsf{N}) \xrightarrow{\alpha}_i (A' \mid B, \mathsf{N}')}$$

$$\textsc{Struct}_i \qquad \frac{(A \;;\; \mathsf{N}) \equiv_i (B \;;\; \mathsf{N}) \xrightarrow{\alpha}_i (B', \mathsf{N}') \equiv_i (A', \mathsf{N}')}{(A \;;\; \mathsf{N}) \xrightarrow{\alpha}_i (A', \mathsf{N}')}$$

One may note two particularities in this semantics. The $\textsc{Open-Ch}_i$ rule requires an "on-the-fly renaming" at the point that we reveal a bound name. This will be needed in the bisimulation because we require both the left- and right-hand processes to use the same label without allowing α-conversion. The second unusual detail is the $\alpha\psi(B)$ label in the $\textsc{Par}_i$ rule which is needed to keep processes applied. Note that $\psi(B)$ can only affect labels that are of the form $\mathsf{in}(c, M)$ since for the other ones, the variables in $\mathrm{dom}(\psi(B))$ do not occur in the label α.

Example 3.7 *Let* $A = \nu a.(\mathsf{in}(c,x).P(x) \mid \{^a/_y\})$ *and* $B = \nu a.(P(a) \mid \{^a/_y\})$. *We have that* $A \xrightarrow{in(c,y)} B$. *Let* N *be a naming environment compatible with* $A\downarrow$ *and* $in(c,y)$. *We have also that* $(A\downarrow\ ;\mathsf{N}) \xrightarrow{in(c,y)} (B\downarrow\ ;\mathsf{N})$. *The derivation witnessing this reduction uses the fact that the label in the* $\textsc{Par}_i$ *rule can be instantiated along the derivation.*

$$\dfrac{\dfrac{(\mathsf{in}(c,x).P(x)\ ;\mathsf{N}) \xrightarrow{in(c,a)} (P(a)\ ;\mathsf{N})}{(\mathsf{in}(c,x).P(x) \mid \{^a/_y\}\ ;\mathsf{N}) \xrightarrow{in(c,y)} (P(a) \mid \{^a/_y\}\ ;\mathsf{N})}}{(\nu a.(\mathsf{in}(c,x).P(x) \mid \{^a/_y\})\ ;\mathsf{N}[a \mapsto \mathsf{b}]) \xrightarrow{in(c,y)} (\nu a.(P(a) \mid \{^a/_y\})\ ;\mathsf{N}[a \mapsto \mathsf{b}])}$$

In particular we note that if the first label had been $\mathsf{in}(c,y)$ *the obtained process would have been* $(P(y)\ ;\mathsf{N})$ *and the* $\textsc{Par}_i$ *rule could not have been used because* $P(y) \mid \{^a/_y\}$ *would not have been an intermediate process (as it is not applied).*

4 Soundness and Completeness

We now introduce the relation $\approxeq$ on intermediate processes. Intuitively, $\approxeq$ captures the structural equivalences that are "missing" in $\equiv_i$ with respect to $\equiv$. We show completeness of the intermediate semantics up to $\approxeq$. Note that the rule $\textsc{New-C}_i$ is in the relation $\approxeq$ because of some tricky interactions between the transformation $\downarrow$ and the relation $\equiv$ (see Example 4.2 given below).

Definition 4.1 ($\approxeq$) *We define* $\approxeq$ *to be the smallest equivalence relation on intermediate processes closed under bijective renaming of bound names and variables and such that*

$$\begin{array}{lrcll}
\textsc{New-N}_i & (\nu\tilde{n}.\nu m.A\ ;\mathsf{N}) & \approxeq & (\nu\tilde{n}.A\ ;\mathsf{N}) & \text{if } m \notin fn(A) \\
\textsc{Rew-N}_i & (A\{^M/_x\}\ ;\mathsf{N}) & \approxeq & (A\{^N/_x\}\ ;\mathsf{N}) & \text{if } M =_{\mathsf{E}} N \\
\textsc{New-C}_i & (\nu n.\nu m.A\ ;\mathsf{N}) & \approxeq & (\nu m.\nu n.A\ ;\mathsf{N}) &
\end{array}$$

Example 4.2 *Consider the processes:* $A = \mathsf{out}(c,n_1).\nu n_2.\nu n_2'.\mathsf{out}(c,\langle n_2,n_2'\rangle)$ *and* $C = \nu n_2'.\nu n_2.\mathsf{out}(c,n_1).\mathsf{out}(c,\langle n_2,n_2'\rangle)$. *We have that*

$$A\downarrow = \nu n_2.\nu n_2'.(\mathsf{out}(c,n_1).\mathsf{out}(c,\langle n_2,n_2'\rangle)) \ \text{ and } \ A\downarrow \equiv C.$$

Note that there is no B *such that* $A \equiv B$ *and* $B\downarrow = C$. *Nevertheless, because of the rule* $\textsc{New-C}_i$ *there exists* B' *(e.g.* $B' = A$*) such that* $A \equiv B'$ *and* $(B'\downarrow\ ;\mathsf{N}) \approxeq (C\ ;\mathsf{N})$ *for any compatible naming environment* N. *This property is needed in the proof of Proposition 5.5.*

Lemma 4.3 *Let* $(A\ ;\mathsf{N})$ *and* $(A'\ ;\mathsf{N})$ *be two intermediate processes such that* $(A\ ;\mathsf{N}) \approxeq (A'\ ;\mathsf{N})$. *Then we have that* $A \equiv A'$.

We now show that our intermediate semantics is *sound* with respect to the original semantics: any structural equivalence and (labelled) reduction that holds in the intermediate context also holds in the original semantics.

Proposition 4.4 (soundness) *Let* $(A_i \; ; \mathsf{N})$ *and* $(B_i \; ; \mathsf{N}')$ *be two intermediate processes such that* $(A_i \; ; \mathsf{N}) \bowtie_i (B_i \; ; \mathsf{N}')$ *with* $\bowtie \in \{\equiv, \rightarrow, \xrightarrow{\alpha}\}$. *Then we have that* $A_i \bowtie B_i$.

The proofs when $\bowtie \in \{\equiv, \rightarrow\}$ are straightforward and are sketched in Appendix A. The proof for $\xrightarrow{\alpha}_i$ is more involved and detailed in Appendix A.

We also introduce a useful *commutation lemma*. As we show completeness for one step of each of these relations up to $\approxeq$ the commutation lemmas will allow us to lift the result to sequences of steps.

Lemma 4.5 (commutation) *Let* $(A \; ; \mathsf{N})$, $(A' \; ; \mathsf{N})$ *and* $(B \; ; \mathsf{N}')$ *be three intermediate processes such that* $(A' \; ; \mathsf{N}) \approxeq (A \; ; \mathsf{N}) \bowtie (B \; ; \mathsf{N}')$ *with* $\bowtie \in \{\equiv_i, \rightarrow_i, \xrightarrow{\alpha}_i\}$. *Then there exists* $(B' \; ; \mathsf{N}')$ *such that* $(A' \; ; \mathsf{N}) \bowtie (B' \; ; \mathsf{N}') \approxeq (B \; ; \mathsf{N}')$.

Proof.(sketch) This result can be proved by considering proofs in "linear form", i.e., by applying the rules directly under the evaluation context, resulting into a sequence of processes rather than a proof tree. Similarly, the proof of $(A' \; ; \mathsf{N}) \approxeq (A \; ; \mathsf{N})$ can be written as a sequence of steps. Now, it is easy to show, by case analysis on each pair of rules, that each time there is an application of $\approxeq$ occurring immediately to the left of an application of $\equiv_i$, this pair of rule applications can be commuted. □

Next we show completeness of the intermediate semantics: any structural equivalence and (labelled) reduction that holds in the original semantics should also hold for corresponding intermediate processes in the new semantics. As discussed previously completeness seems difficult to achieve. We therefore show completeness up to $\approxeq$.

Proposition 4.6 (completeness) *Let* A *and* B *be two nv-distinct extended processes such that* $A \bowtie B$ *with* $\bowtie \in \{\equiv, \rightarrow, \xrightarrow{\alpha}\}$. *Let* N *be a naming environment compatible with* $A{\downarrow}$ *and also with* α *when* $\bowtie = \xrightarrow{\alpha}$. *Let* N' *be a naming environment compatible with* $B{\downarrow}$ *and such that:*

- $\mathsf{N}' = \mathsf{N}[x \mapsto \mathsf{f}]$ *when* $\bowtie = \xrightarrow{\alpha}$ *and* α *is of the form* $\nu x.out(a, x)$;
- $\mathsf{N}' = \mathsf{N}[d \mapsto \mathsf{f}]$ *when* $\bowtie = \xrightarrow{\alpha}$ *and* α *is of the form* $\nu d.out(a, d)$;
- $\mathsf{N}' = \mathsf{N}$ *otherwise.*

Then there exists an intermediate process $(D_i \; ; \mathsf{N}')$ *such that*

- $(A{\downarrow} \; ; \mathsf{N}) \bowtie_i (D_i \; ; \mathsf{N}')$, *and*
- $(D_i \; ; \mathsf{N}') \approxeq (B{\downarrow} \; ; \mathsf{N}')$.

Note that, when $\bowtie \in \{\equiv, \rightarrow\}$, we have that $\mathsf{N}' = \mathsf{N}$. The proofs are done in Appendix A.

From Propositions 4.4 and 4.6 and the commutation lemma stated above, we derive the following corollaries. (Corollary 4.8 also requires Lemmas A.5 and A.8 in the appendix.)

Corollary 4.7 (soundness of $\rightarrow_i^*$ and $\rightarrow_i^* \xrightarrow{\alpha}_i \rightarrow_i^*$) *Let $(A_i \,;\, \mathsf{N})$ and $(B_i \,;\, \mathsf{N}')$ be two intermediate processes such that $(A_i \,;\, \mathsf{N}) \rightarrow_i^* (B_i \,;\, \mathsf{N}')$ (resp. $(A_i \,;\, \mathsf{N}) \rightarrow_i^* \xrightarrow{\alpha}_i \rightarrow_i^* (B_i \,;\, \mathsf{N}')$). Then we have that $A_i \rightarrow^* B_i$ (resp. $A_i \rightarrow^* \xrightarrow{\alpha} \rightarrow^* B_i$).*

Corollary 4.8 (completeness of $\rightarrow_i^*$ and $\rightarrow_i^* \xrightarrow{\alpha}_i \rightarrow_i^*$) *Let A and B be two nv-distinct extended processes such that $A \rightarrow^* B$ (resp. $A \rightarrow^* \xrightarrow{\alpha} \rightarrow^* B$) and N be a naming environment compatible with $A\downarrow$ (resp., and α). Let N' be a naming environment compatible with $B\downarrow$ and such that:*

- $\mathsf{N}' = \mathsf{N}[x \mapsto \mathsf{f}]$ *in the case $\rightarrow_i^* \xrightarrow{\alpha}_i \rightarrow_i^*$ when α is of the form $\nu x.out(a, x)$;*
- $\mathsf{N}' = \mathsf{N}[d \mapsto \mathsf{f}]$ *in the case $\rightarrow_i^* \xrightarrow{\alpha}_i \rightarrow_i^*$ when α is of the form $\nu d.out(a, d)$;*
- $\mathsf{N}' = \mathsf{N}$ *otherwise.*

Then there exists an intermediate process $(D_i \,;\, \mathsf{N}')$ such that $(A\downarrow \,;\, \mathsf{N}) \rightarrow_i^* (D_i \,;\, \mathsf{N}')$ (resp. $(A\downarrow \,;\, \mathsf{N}) \rightarrow_i^* \xrightarrow{\alpha}_i \rightarrow_i^* (D_i \,;\, \mathsf{N}')$) and $(D_i \,;\, \mathsf{N}') \cong (B\downarrow \,;\, \mathsf{N}')$.

5 Intermediate Bisimulation

We now define the intermediate labelled bisimulation. The definition is similar to the original one, but is stated with respect to our intermediate semantics. Moreover, note that the side condition $bn(\alpha) \cap fn(B) = \emptyset$ has been removed since the fact that both processes are running in the same naming environment ensures this condition.

Definition 5.1 (Intermediate labelled bisimilarity ($\approx_i$)) Intermediate labelled bisimilarity *is the largest symmetric relation $\mathcal{R}$ on closed intermediate processes with same naming environment, such that $(A_i, \mathsf{N})\ \mathcal{R}\ (B_i \,;\, \mathsf{N})$ implies*

1. *$A_i \sim B_i$,*
2. *if $(A_i \,;\, \mathsf{N}) \rightarrow_i (A_i' \,;\, \mathsf{N})$, then $(B_i \,;\, \mathsf{N}) \rightarrow_i^* (B_i' \,;\, \mathsf{N})$ and $(A_i' \,;\, \mathsf{N})\ \mathcal{R}\ (B_i' \,;\, \mathsf{N})$ for some B_i',*
3. *if $(A_i \,;\, \mathsf{N}) \xrightarrow{\alpha}_i (A_i' \,;\, \mathsf{N}')$ and $fv(\alpha) \subseteq \mathrm{dom}(A_i)$, then $(B_i \,;\, \mathsf{N}) \rightarrow_i^* \xrightarrow{\alpha}_i \rightarrow_i^* (B_i' \,;\, \mathsf{N}')$ and $(A_i' \,;\, \mathsf{N}')\ \mathcal{R}\ (B_i' \,;\, \mathsf{N}')$ for some B_i'.*

The following theorem states that the intermediate and the original bisimulations coincide.

Theorem 5.2 *Let A and B be two nv-distinct extended processes and N be a naming environment compatible with $A\downarrow$ and $B\downarrow$. We have that*

$$A \approx B \text{ if and only if } (A\downarrow \,;\, \mathsf{N}) \approx_i (B\downarrow \,;\, \mathsf{N})$$

Both directions of this result are proved separately by the following propositions.

Proposition 5.3 *Let A and B be two nv-distinct extended processes. We have*

$$A \approx B \text{ implies } (A\downarrow ; \mathsf{N}) \approx_i (B\downarrow ; \mathsf{N}).$$

for any naming environment N compatible with $A\downarrow$ and $B\downarrow$.

Proof. To prove this result, first we define a new relation $\mathcal{R}$. Next we will show that $\mathcal{R}$ is an intermediate labelled bisimulation witnessing $(A\downarrow ; \mathsf{N}) \approx_i (B\downarrow ; \mathsf{N})$.

(i) Definition of $\mathcal{R}$.
We define $\mathcal{R}$ as follows: $(A_i ; \mathsf{N})\ \mathcal{R}\ (B_i ; \mathsf{N})$ if $A_i \approx B_i$ and N is a naming environment compatible with A_i and B_i.

(ii) $\mathcal{R}$ is an intermediate bisimulation relation witnessing $(A\downarrow ; \mathsf{N}) \approx_i (B\downarrow ; \mathsf{N})$.
First we have to show that $(A\downarrow ; \mathsf{N})\ \mathcal{R}\ (B\downarrow ; \mathsf{N})$. We have that $A \approx B$ and hence $A\downarrow \approx B\downarrow$ (by Lemma 3.6) and N is a naming environment compatible with $A\downarrow$ and $B\downarrow$. Hence, by definition of $\mathcal{R}$, we easily conclude.

Now, we have to show that $\mathcal{R}$ satisfies the three points of the definition of intermediate labelled bisimilarity. Let $(A_i ; \mathsf{N})$ and $(B_i ; \mathsf{N})$ be two closed intermediate processes such that $(A_i ; \mathsf{N})\ \mathcal{R}\ (B_i ; \mathsf{N})$. By definition of $\mathcal{R}$, we have that $A_i \approx B_i$.
We have to show that:

1. $A_i \sim B_i$. By hypothesis, we have that $A_i \approx B_i$ which implies $A_i \sim B_i$.

2. If $(A_i ; \mathsf{N}) \rightarrow_i (A'_i ; \mathsf{N})$ then there exists $(B'_i ; \mathsf{N})$ with $(B_i ; \mathsf{N}) \rightarrow_i^* (B'_i ; \mathsf{N})$ and $(A'_i ; \mathsf{N})\ \mathcal{R}\ (B'_i ; \mathsf{N})$.
   By Proposition 4.4, we have that $A_i \rightarrow A'_i$. Since $A_i \approx B_i$ and $A_i \rightarrow A'_i$ there exists an extended process B' such that $B_i \rightarrow^* B'$ and $A'_i \approx B'$. In the remainder, we assume w.l.o.g. that B' is nv-distinct and compatible with N. By Corollary 4.8, there exists an intermediate extended process $(D ; \mathsf{N})$ such that

   - $(B_i\downarrow ; \mathsf{N}) \rightarrow_i^* (D ; \mathsf{N})$, and
   - $(D ; \mathsf{N}) \cong (B'\downarrow ; \mathsf{N})$.

   As B_i is an intermediate process we have that $B_i\downarrow = B_i$. Let $B'_i = D$. It remains to show that $(A'_i ; \mathsf{N})\ \mathcal{R}\ (B'_i ; \mathsf{N})$. As $A'_i \approx B'$, $B' \approx B'\downarrow$ (by Lemma 3.6), and $B'_i \equiv B'\downarrow$ (by Lemma 4.3 and the fact that $(B'_i ; \mathsf{N}) = (D ; \mathsf{N}) \cong (B'\downarrow ; \mathsf{N})$) we have that $A'_i \approx B'_i$. By definition of $\mathcal{R}$ we deduce that $(A'_i ; \mathsf{N})\ \mathcal{R}\ (B'_i ; \mathsf{N})$.

3. If $(A_i ; \mathsf{N}) \xrightarrow{\alpha}_i (A'_i ; \mathsf{N}')$ with $fv(\alpha) \subseteq \mathrm{dom}(A_i)$ then there exists $(B'_i ; \mathsf{N}')$ such that $(B_i ; \mathsf{N}) \rightarrow_i^* \xrightarrow{\alpha}_i \rightarrow_i^* (B'_i ; \mathsf{N}')$ and $(A' ; \mathsf{N}')\ \mathcal{R}'\ (B'_i ; \mathsf{N}')$.
   This case is similar to the previous one.

□

To show the converse direction, we need an additional lemma.

Lemma 5.4 *Let* $(A \ ; \mathsf{N})$ *and* $(B \ ; \mathsf{N})$ *be two intermediate processes, and* N' *a naming environment compatible with* A *and* B*. Then:*

1. $(A \ ; \mathsf{N}) \approx_i (B \ ; \mathsf{N})$ *implies* $(A \ ; \mathsf{N}') \approx_i (B \ ; \mathsf{N}')$*; and*
2. $(A \ ; \mathsf{N}) \cong (B \ ; \mathsf{N})$ *implies* $(A \ ; \mathsf{N}') \cong (B \ ; \mathsf{N}')$.

To see that the first part of the lemma holds it is sufficient to note that $(A \ ; \mathsf{N}) \bowtie_i (A' \ ; \mathsf{N}')$ if and only if $(A\rho \ ; \mathsf{N}\rho) \bowtie_i (A'\rho \ ; \mathsf{N}'\rho)$ where ρ is a bijective renaming of names and variables, $\bowtie \in \{\equiv, \rightarrow, \xrightarrow{\alpha}\}$ and $\mathsf{N}\rho(u) = \mathsf{N}(\rho^{-1}(u)))$. The second part is immediate.

Proposition 5.5 *Let* $(A_i \ ; \mathsf{N})$ *and* $(B_i \ ; \mathsf{N})$ *be two intermediate processes. We have that*

$$(A_i \ ; \mathsf{N}) \approx_i (B_i \ ; \mathsf{N}) \text{ implies } A_i \approx B_i$$

Proof. To prove this result, we first define a new relation $\mathcal{R}$ and then we will show that $\mathcal{R}$ witnesses $\approx$.

(i) Definition of $\mathcal{R}$*.*
$A \ \mathcal{R} \ B$ if there exist two intermediate processes $(\hat{A} \ ; \mathsf{N})$ and $(\hat{B} \ ; \mathsf{N})$, and two nv-distinct extended processes A_α and B_α such that

- $(\hat{A} \ ; \mathsf{N}) \approx_i (\hat{B} \ ; \mathsf{N})$,
- $(\hat{A} \ ; \mathsf{N}) \cong (A_\alpha{\downarrow} \ ; \mathsf{N})$ and $(\hat{B} \ ; \mathsf{N}) \cong (B_\alpha{\downarrow} \ ; \mathsf{N})$,
- $A_\alpha =_\alpha A$ and $B_\alpha =_\alpha B$.

(ii) $\mathcal{R}$ *witnesses* $\approx$*.*
First we show that $A_i \ \mathcal{R} \ B_i$. Let $\hat{A} = A_\alpha = A_i$ and $\hat{B} = B_\alpha = B_i$. By definition of $\mathcal{R}$ we easily conclude as $(A_i \ ; \mathsf{N}) \approx_i (B_i \ ; \mathsf{N})$.

Now, we show that $\mathcal{R}$ satisfies the three points of the definition of $\approx$. Let A and B be two closed intermediate extended processes such that $A \ \mathcal{R} \ B$. By definition of $\mathcal{R}$, we know that there exist two intermediate processes $(\hat{A} \ ; \mathsf{N})$ and $(\hat{B} \ ; \mathsf{N})$ such that

- $(\hat{A} \ ; \mathsf{N}) \approx_i (\hat{B} \ ; \mathsf{N})$,
- $(\hat{A} \ ; \mathsf{N}) \cong (A_\alpha{\downarrow} \ ; \mathsf{N})$ and $(\hat{B} \ ; \mathsf{N}) \cong (B_\alpha{\downarrow} \ ; \mathsf{N})$, and
- $A_\alpha =_\alpha A$ and $B_\alpha =_\alpha B$.

We have to show that:

1. $A \sim B$. Since $(\hat{A} \ ; \mathsf{N}) \approx_i (\hat{B} \ ; \mathsf{N})$, we have that $\hat{A} \sim \hat{B}$ by definition of $\approx_i$. As $A_\alpha =_\alpha A$ and $B_\alpha =_\alpha B$, we have that $\hat{A} \sim A$ and $\hat{B} \sim B$. We conclude by transitivity of $\sim$.

2. If $A \to A'$ for some extended process A' then there exists B' such that $B \to^* B'$ and $A' \mathcal{R} B'$.
   If $A \to A'$ we also have that $A_\alpha \to A'_\alpha$ for some A'_α such that $A'_\alpha =_\alpha A'$ and A'_α compatible with N. By Proposition 4.6, there exists an extended process $(D \; ; \mathsf{N})$ such that

   - $(A_\alpha{\downarrow} \; ; \mathsf{N}) \to_i (D \; ; \mathsf{N})$, and
   - $(D \; ; \mathsf{N}) \cong (A'_\alpha{\downarrow} \; ; \mathsf{N})$.

   We have that $(\hat{A} \; ; \mathsf{N}) \cong (A_\alpha{\downarrow} \; ; \mathsf{N}) \to_i (D \; ; \mathsf{N}) \cong (A'_\alpha{\downarrow} \; ; \mathsf{N})$. By Lemma 4.5, there exists (D', N) such that

   $$(\hat{A} \; ; \mathsf{N}) \to_i (D' \; ; \mathsf{N}) \cong (D \; ; \mathsf{N}) \cong (A'_\alpha{\downarrow} \; ; \mathsf{N}).$$

   Since $(\hat{A} \; ; \mathsf{N}) \approx_i (\hat{B} \; ; \mathsf{N})$ and $(\hat{A} \; ; \mathsf{N}) \to_i (D' \; ; \mathsf{N})$, we have that there exists $(B'_i \; ; \mathsf{N})$ such that $(\hat{B} \; ; \mathsf{N}) \to_i^* (B'_i \; ; \mathsf{N})$ and $(D' \; ; \mathsf{N}) \approx_i (B'_i \; ; \mathsf{N})$. By Corollary 4.7, we have that $\hat{B} \to^* B'_i$. Thus, we deduce that $B \to^* B''_i$, for some B''_i such that $(B''_i{\downarrow} \; ; \mathsf{N}) \cong (B'_i \; ; \mathsf{N})$. Let $B' = B''_i$, $B'_\alpha = B'$, $\hat{B'} = B'_i$ and $\hat{A'} = D'$. We have that $B \to^* B'$ and by definition of $\mathcal{R}$, we have that $A' \mathcal{R} B'$. Indeed, we have that:

   - $(\hat{A'} \; ; \mathsf{N}) \approx_i (\hat{B'} \; ; \mathsf{N})$, i.e. $(D' \; ; \mathsf{N}) \approx_i (B'_i \; ; \mathsf{N})$,
   - $(\hat{A'} \; ; \mathsf{N}) \cong (A'_\alpha{\downarrow} \; ; \mathsf{N})$ and $(\hat{B'} \; ; \mathsf{N}) \cong (B'_\alpha{\downarrow} \; ; \mathsf{N})$, and
   - $A'_\alpha =_\alpha A'$ and $B'_\alpha =_\alpha B'$.

3. If $A \xrightarrow{\alpha} A'$ and $fv(\alpha) \subseteq \mathrm{dom}(A)$ and $bn(\alpha) \cap fn(B) = \emptyset$ for some extended process A' then $B \to^* \xrightarrow{\alpha} \to^* B'$ and $A' \mathcal{R} B'$ for some process B'.

   First, we can assume w.l.o.g. that $(fn(\alpha) \cup bn(\alpha)) \cap (bn(A_\alpha) \cup bn(B_\alpha)) = \emptyset$ and $(fn(\alpha) \cup bn(\alpha)) \cap (bn(\hat{A}) \cup bn(\hat{B})) = \emptyset$ (since A_α, B_α $\hat{A}$, $\hat{B}$ can be chosen to make this true). For the same reason, we can assume that $bv(\alpha) \cap (bv(A_\alpha) \cup bv(B_\alpha)) = \emptyset$ and $bv(\alpha) \cap (bv(\hat{A}) \cup bv(\hat{B})) = \emptyset$. Let $\hat{\mathsf{N}} = \mathsf{N}[fn(\alpha) \mapsto \mathsf{f}][bn(\alpha), bv(\alpha) \mapsto \mathsf{n}]$. Note that $\hat{\mathsf{N}}$ is now compatible with A_α, B_α $\hat{A}$, $\hat{B}$ and α. Let A'_α be an nv-distinct extended process such that $A_\alpha \xrightarrow{\alpha} A'_\alpha$ and A'_α compatible with $\hat{\mathsf{N}}'$ where $\hat{\mathsf{N}}'$ is defined as follows:

   $$\hat{\mathsf{N}}' = \begin{cases} \hat{\mathsf{N}}[x \mapsto \mathsf{f}] & \text{when } \alpha \text{ is of the form } \nu x.out(a, x); \\ \hat{\mathsf{N}}[d \mapsto \mathsf{f}] & \text{when } \alpha \text{ is of the form } \nu d.out(a, d); \\ \hat{\mathsf{N}} & \text{otherwise.} \end{cases}$$

   By Proposition 4.6, there exists an extended process $(D \; ; \hat{\mathsf{N}}')$ such that

   - $(A_\alpha{\downarrow} \; ; \hat{\mathsf{N}}) \xrightarrow{\alpha}_i (D \; ; \hat{\mathsf{N}}')$, and
   - $(D \; ; \hat{\mathsf{N}}') \cong (A'_\alpha{\downarrow} \; ; \hat{\mathsf{N}}')$.

We have that $(\hat{A}\ ;\ \mathsf{N}) \cong (A_\alpha\downarrow\ ;\ \mathsf{N})$ and thus $(\hat{A}\ ;\ \hat{\mathsf{N}}) \cong (A_\alpha\downarrow\ ;\ \hat{\mathsf{N}})$ (Lemma 5.4). Moreover, we have that $(A_\alpha\downarrow\ ;\ \hat{\mathsf{N}}) \xrightarrow{\alpha}_i (D\ ;\ \hat{\mathsf{N}}') \cong (A'_\alpha\downarrow\ ;\ \hat{\mathsf{N}}')$. By Lemma 4.5, there exists $(D', \hat{\mathsf{N}}')$ such that

$$(\hat{A}\ ;\ \hat{\mathsf{N}}) \xrightarrow{\alpha}_i (D'\ ;\ \hat{\mathsf{N}}') \cong (D\ ;\ \hat{\mathsf{N}}') \cong (A'_\alpha\downarrow\ ;\ \hat{\mathsf{N}}').$$

Since $(\hat{A}\ ;\ \mathsf{N}) \approx_i (\hat{B}\ ;\ \mathsf{N})$, we have also that $(\hat{A}\ ;\ \hat{\mathsf{N}}) \approx_i (\hat{B}\ ;\ \hat{\mathsf{N}})$ (Lemma 5.4). Moreover, we have that $(\hat{A}\ ;\ \hat{\mathsf{N}}) \xrightarrow{\alpha}_i (D'\ ;\ \hat{\mathsf{N}}')$, thus there exists $(B'_i\ ;\ \hat{\mathsf{N}}')$ such that $(\hat{B}\ ;\ \hat{\mathsf{N}}) \to_i^* \xrightarrow{\alpha}_i \to_i^* (B'_i\ ;\ \hat{\mathsf{N}}')$ and $(D'\ ;\ \hat{\mathsf{N}}') \approx_i (B'_i\ ;\ \hat{\mathsf{N}}')$. By Corollary 4.7, we have that $\hat{B} \to^* \xrightarrow{\alpha} \to^* B'_i$. Thus, we deduce that $B \to^* \xrightarrow{\alpha} \to^* B''_i$ for some B''_i such that $(B''_i\downarrow\ ;\ \hat{\mathsf{N}}') \cong (B'_i\ ;\ \hat{\mathsf{N}}')$. Let $B' = B''_i$, $B'_\alpha = B'$, $\hat{B}' = B'_i$ and $\hat{A}' = D'$. We have that $B \to^* \xrightarrow{\alpha} \to^* B'$ and by definition of $\mathcal{R}$, we have that $A'\ \mathcal{R}\ B'$. Indeed, we have that:

- $(\hat{A}'\ ;\ \hat{\mathsf{N}}') \approx_i (\hat{B}'\ ;\ \hat{\mathsf{N}}')$, i.e. $(D'\ ;\ \hat{\mathsf{N}}') \approx_i (B'_i\ ;\ \hat{\mathsf{N}}')$,
- $(\hat{A}'\ ;\ \hat{\mathsf{N}}') \cong (A'_\alpha\downarrow\ ;\ \hat{\mathsf{N}}')$ and $(\hat{B}'\ ;\ \hat{\mathsf{N}}') \cong (B'_\alpha\downarrow\ ;\ \hat{\mathsf{N}}')$, and
- $A'_\alpha =_\alpha A'$ and $B'_\alpha =_\alpha B'$.

□

— PART II: Symbolic Calculus —

A *symbolic process* is an intermediate process together with a *constraint system*. The aim of a symbolic semantics is to avoid the infinite branching due to the inputs of the environment. This is achieved by keeping variables rather than the input terms. The constraint system gives a finite representation of the value that these variables are allowed to take.

6 Constraint systems

Definition 6.1 (constraint system) *A* constraint system $\mathcal{C}$ *is a set of constraints where every constraint is of one of the following forms:*

- *"*$\varphi \Vdash x$*", where* $\varphi = \nu\tilde{u}.\sigma$ *for some tuple of names and variables* $\tilde{u}$ *and some substitution* σ*, and* x *is a variable which does not appear under a restriction of any frame nor in the domain of any frame;*
- *"*$M = N$*", where* M *and* N *are terms;*
- *"*$M \neq N$*", where* M *and* N *are terms;*
- *"*$\mathsf{gd}(M)$*" where* M *is a term;*

The constraint $\varphi \Vdash x$ is useful for specifying the information φ held by the environment when it supplies an input x. As we will see in the following section, these variables will be taken from a special set of variables. The constraint $\mathsf{gd}(M)$ means that the term M is ground. We denote by $\mathcal{D}ed(\mathcal{C})$ the *deducibility constraints* of $\mathcal{C}$, i.e. $\{\varphi \Vdash x \mid \text{“}\varphi \Vdash x\text{”} \in \mathcal{C}\}$. When $\mathcal{D}ed(\mathcal{C}) = \{\varphi_1 \Vdash x_1, \ldots, \varphi_\ell \Vdash x_\ell\}$, we define $cv(\mathcal{C}) = \{x_1, \ldots, x_\ell\}$ to be the *constraint variables* of $\mathcal{C}$. Moreover, we write $names(\mathcal{C})$ (resp. $vars(\mathcal{C})$) for the names (resp. variables) of $\mathcal{C}$.

The constraint systems that we consider arise while executing symbolic processes. We therefore restrict ourselves to *well-formed* constraint systems, capturing the fact that the knowledge of the environment always increases along the execution: we allow it to use more names and variables (less restrictions) or give it access to more terms (larger substitution). The fact that the constraint system is not arbitrary is useful when solving the constraints such as in [20].

Definition 6.2 (ordering on frames $\preceq$) *Let* $\varphi_1 = \nu\tilde{u}_1.\sigma_1$ *and* $\varphi_2 = \nu\tilde{u}_2.\sigma_2$ *be two frames. The frame* φ_1 *is smaller than or equal to* φ_2*, denoted by* $\varphi_1 \preceq \varphi_2$*, if* $\tilde{u}_1 \supseteq \tilde{u}_2$*,* $\mathrm{dom}(\sigma_1) \subseteq \mathrm{dom}(\sigma_2)$ *and* $y\sigma_1 = y\sigma_2$ *for any* $y \in \mathrm{dom}(\sigma_1)$*.*

Definition 6.3 (well-formed constraint system) *A constraint system* $\mathcal{C}$ *is* well-formed *if* $\mathcal{D}ed(\mathcal{C})$ *can be ordered* $\varphi_1 \Vdash x_1, \ldots, \varphi_\ell \Vdash x_\ell$ *in such a way that:*

1. *(monotonicity) for every* i *such that* $1 \leq i < \ell$*, we have* $\varphi_i \preceq \varphi_{i+1}$*, and*
2. *(origination) for every* i *such that* $1 \leq i \leq \ell$*, we have*

for all $x \in vars(\mathrm{img}(\varphi_i)) \cap cv(\mathcal{C})$*, there exists* $j < i$ *such that* $x = x_j$*.*

Moreover, if "$M \neq N$" $\in \mathcal{C}$ *then* "$\mathsf{gd}(M)$" $\in \mathcal{C}$ *and* "$\mathsf{gd}(N)$" $\in \mathcal{C}$*.*

In the remainder, when we consider a well-formed constraint system $\mathcal{C}$ and we write, for $\mathcal{D}ed(\mathcal{C})$, the sequence $\varphi_1 \Vdash x_1, \ldots, \varphi_\ell \Vdash x_\ell$, this implicitly means that we consider the ordering given by the monotonicity condition.

We say that two well-formed constraint systems $\mathcal{C}$ and $\mathcal{C}'$ have *same basis* if $\mathcal{D}ed(\mathcal{C}) = \{\varphi_1 \Vdash x_1, \ldots, \varphi_\ell \Vdash x_\ell\}$ and $\mathcal{D}ed(\mathcal{C}') = \{\varphi'_1 \Vdash x'_1, \ldots, \varphi'_\ell \Vdash x'_\ell\}$ are such that $x_i = x'_i$ and $\mathrm{dom}(\varphi_i) = \mathrm{dom}(\varphi'_i)$ for $1 \leq i \leq \ell$.

We now define the solutions of a well-formed constraint system. Intuitively, each solution defines an intermediate process which corresponds to a concrete instance of the corresponding symbolic process.

Definition 6.4 (E-solution) *Let* $\mathcal{C}$ *be a well-formed constraint system such that* $\mathcal{D}ed(\mathcal{C}) = \{\varphi_1 \Vdash x_1, \ldots, \varphi_\ell \Vdash x_\ell\}$ *where each* $\varphi_i = \nu\tilde{u}_i.\sigma_i$ *for some* $\tilde{u}_i$ *and some substitution* σ_i*. An* E-solution *of* $\mathcal{C}$ *is a substitution* θ *whose domain is* $cv(\mathcal{C})$ *and such that*

- $vars(x_i\theta) \cap cv(\mathcal{C}) = \emptyset$ *and* $vars(x_i\theta) \cap (\mathrm{dom}(\varphi_\ell) \setminus \mathrm{dom}(\varphi_i)) = \emptyset$*;*
- $names(x_i\theta) \cap \tilde{u}_i = \emptyset$ *and* $vars(x_i\theta) \cap \tilde{u}_i = \emptyset$*;*
- *for every constraint* "$M = N$" $\in \mathcal{C}$*, we have* $M(\theta\sigma_\ell)^\star =_{\mathsf{E}} N(\theta\sigma_\ell)^\star$*;*
- *for every constraint* "$M \neq N$" $\in \mathcal{C}$*, we have* $M(\theta\sigma_\ell)^\star \neq_{\mathsf{E}} N(\theta\sigma_\ell)^\star$*;*
- *for every constraint* "$\mathsf{gd}(M)$" $\in \mathcal{C}$*, the term* $M(\theta\sigma_\ell)^\star$ *is ground.*

We denote by $Sol_{\mathsf{E}}(\mathcal{C})$ *the set of* E*-solutions of* $\mathcal{C}$*. An* E*-solution* θ *of* $\mathcal{C}$ *is* closed *if* $vars(x_i\theta) \subseteq \mathrm{dom}(\varphi_i)$ *for any* $i \in \{1, \ldots, \ell\}$*. We denote by* $Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C})$ *the set of closed* E*-solutions of* $\mathcal{C}$*.*

The condition that $vars(x_i\theta) \cap cv(\mathcal{C}) = \emptyset$ states that the image of θ should not use any variables that are in the domain of θ. The second condition, $vars(x_i\theta) \cap (\mathrm{dom}(\varphi_\ell) \setminus \mathrm{dom}(\varphi_i)) = \emptyset$, ensures that the environment does not use information that will only be revealed "in the future"; it can use only the entries of the frame that have previously been added. The conditions $names(x_i\theta) \cap \tilde{u}_i = \emptyset$ and $vars(x_i\theta) \cap \tilde{u}_i = \emptyset$ dissallow the environment to use restricted names and variables which are supposed to be secret; thus, they ensure that the value $x_i\theta$ can be *deduced* from public data. (These conditions are related to the definition of deduction given in [1].) The meaning of the remaining conditions should be clear.

Example 6.5 *Let* $\mathcal{C} = \{\nu k.\nu s.\{^{\mathsf{enc}(s,k)}/_{y_1}, {}^{k}/_{y_2}\} \Vdash x', \mathsf{gd}(c), x' = s\}$*. Let* E *be the equational theory* $\mathsf{dec}(\mathsf{enc}(x, y), y) = x$ *and* $\theta = \{^{\mathsf{dec}(y_1,y_2)}/_{x'}\}$*. We have that* θ *is a closed* E*-solution of* $\mathcal{C}$*. Note that* $\theta' = \{^{\mathsf{dec}(y_1,k)}/_{x'}\}$ *is not an* E*-solution of* $\mathcal{C}$*. Indeed,* $names(x'\theta') \cap \{k, s\} = \{k\}$ *and thus is not empty.*

We also define what it means to apply an evaluation context on a constraint system. This is needed because we define the semantics in a compositional way.

Definition 6.6 ($C[\mathcal{C}]$) *Let $C = \nu\tilde{n}.(\_ \mid D)$ be an intermediate evaluation context and e be a constraint. We have that*

- $C[e] = e$ *when e is a constraint of the form $M = N$, $M \neq N$ or $\mathsf{gd}(M)$;*
- $C[\nu\tilde{v}.\sigma \Vdash x] = \nu\tilde{n}.\nu\tilde{v}.(\sigma \cup \psi(D)) \Vdash x$ *otherwise.*

Similarly, for a variable $y \in \mathcal{X}$

- $\nu y.e = e$ *when e is a constraint of the form $M = N$, $M \neq N$ or $\mathsf{gd}(M)$;*
- $\nu y.e = \nu y, \tilde{v}.\sigma \Vdash x$ *when $e = \nu\tilde{v}.\sigma \Vdash x$.*

Given a constraint system $\mathcal{C}$, we have that $C[\mathcal{C}] = \{C[e] \mid e \in \mathcal{C}\}$ and $\nu y.\mathcal{C} = \{\nu y.e \mid e \in \mathcal{C}\}$.

7 Syntax and Semantics

7.1 Syntax

We first need to extend naming environments used in the intermediate semantics to the symbolic setting. For this, we introduce an infinite set $\mathcal{Y}$ of variables, to be used as constraint variables, disjoint from the set $\mathcal{X}$. We also distinghuish two disjoint subsets: $\mathcal{Y}_b$ for variables of base type and $\mathcal{Y}_{ch}$ for those of channel type. The functions *vars* and *fv* are updated to also return variables from $\mathcal{Y}$.

Definition 7.1 (Symbolic naming environment) *A* symbolic naming environment $\mathsf{N_s} : \mathcal{N} \cup \mathcal{X} \cup \mathcal{Y} \to \{\mathsf{n}, \mathsf{f}, \mathsf{b}, \mathsf{c}\}$ *is a function which maps each name and variable in $\mathcal{N} \cup \mathcal{X}$ to one of* n, f *and* b *and each variable in $\mathcal{Y}$ to one of* n *or* c. *It extends naming environments with an infinite set $\mathcal{Y}$ of variables that can be mapped to* c *(which stands for "constraint") or* n. *As before, we require that there are infinitely many names and infinitely many variables that are mapped to each of* n, f *and* b.

We say that a symbolic naming environment N_s is compatible *with an intermediate extended process A and a constraint system $\mathcal{C}$ if*

- $\mathsf{N}_s(fn(A)) = \mathsf{f}$
- $\mathsf{N}_s(bn(A) \cup bv(A)) = \mathsf{b}$
- $\mathsf{N}_s(names(\mathcal{C})) \subseteq \{\mathsf{f}, \mathsf{b}\}$
- $\mathsf{N}_s(fv(A)) \subseteq \{\mathsf{f}, \mathsf{c}\}$
- $\mathsf{N}_s(y) = \mathsf{c}$ *iff* $y \in cv(\mathcal{C})$
- $\mathsf{N}_s(vars(\mathcal{C})) \subseteq \{\mathsf{f}, \mathsf{c}, \mathsf{b}\}$

Intuitively, $\mathsf{N}_s(y) = \mathsf{c}$ means that the variable y is a *constraint* variable (i.e., an input from the environment subject to constraints in $\mathcal{C}$).

We are now ready to precisely define a symbolic process.

Definition 7.2 (Symbolic process) *A symbolic process is a triple $(A \; ; \; \mathcal{C} \; ; \; \mathsf{N}_s)$ where A is an intermediate extended process, $\mathcal{C}$ is a constraint system and N_s is a symbolic naming environment. We say that $(A \; ; \; \mathcal{C} \; ; \; \mathsf{N_s})$ is well-formed if*

- N_s *is compatible with* A *and* $\mathcal{C}$*;*
- *If* $\mathcal{D}ed(\mathcal{C}) = \{\varphi_1 \Vdash x_1, \ldots, \varphi_\ell \Vdash x_\ell\} \neq \emptyset$ *then* $\phi(A) \succeq \varphi_\ell$ *and* $bv(\varphi_1) \subseteq \mathrm{dom}(A)$*;*
- *for all* M, N *such that* $M = N$, $M \neq N$ *or* $\mathsf{gd}(M)$ *is in* $\mathcal{C}$ *we have that* $vars(M, N) \cap dom(A) = \emptyset$.

$(A\,;\mathcal{C}\,;\mathsf{N_s})$ *is said to be closed if* $fv(A) \subseteq \mathrm{dom}(A) \cup cv(\mathcal{C})$.

Given a well-formed symbolic process $(A\,;\mathcal{C}\,;\mathsf{N_s})$ we define by $Sol_{\mathsf{E}}(\mathcal{C}\,;\mathsf{N}_s)$ the set of solutions of $\mathcal{C}$ which are compatible with N_s, i.e.

$$Sol_{\mathsf{E}}(\mathcal{C}, \mathsf{N}_s) = \{\theta \mid \theta \in Sol_{\mathsf{E}}(\mathcal{C}), \mathsf{N}_s(names(\mathrm{img}(\theta)) \cup vars(\mathrm{img}(\theta))) = \mathsf{f}\}.$$

We denote by $Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}\ ;\ \mathsf{Ns})$ the subset of $Sol_{\mathsf{E}}(\mathcal{C}, \mathsf{N}_s)$ containing closed E-solutions of $\mathcal{C}$.

Each of these solutions θ defines a corresponding (closed) intermediate process which we call the θ-concretization.

Definition 7.3 (θ-concretization) *Let* $(A_s\ ;\mathcal{C}\ ;\mathsf{N}_s)$ *be a well-formed symbolic process. Let* $\theta \in Sol_{\mathsf{E}}(\mathcal{C}, \mathsf{N}_s)$*. We say that an intermediate process* $(A\ ;\mathsf{N})$ *is the* θ*-concretization of* $(A_s\ ;\mathcal{C}\ ;\mathsf{N}_s)$ *if* $A = A_s(\theta\sigma)^\star$ *where* σ *is the maximal frame of* $\mathcal{C}$ *and* $\mathsf{N} = \mathsf{N}_s|_{\mathcal{N}\cup\mathcal{X}}$.

Example 7.4 *Let* $A_s = \nu b.(\mathsf{out}(c, x) \mid \{^b/_y\})$, $\mathcal{C} = \{\nu a.\nu b.\{^b/_y\} \Vdash x,\ x \neq c,\ \mathsf{gd}(x)\}$ *and* N_s *be a naming environment compatible with* A_s *and* $\mathcal{C}$ *such that* $\mathsf{N}_s(d) = \mathsf{f}$*. Let* $\theta_1 = \{^d/_x\}$, $\theta_2 = \{^y/_x\}$ *and* $\mathsf{N} = \mathsf{N}_s|_{\mathcal{N}\cup\mathcal{X}}$*. We have that* $\theta_1, \theta_2 \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}, \mathsf{N}_s)$*. Hence* $(\nu b.(\mathsf{out}(c, d) \mid \{^b/_y\})\ ;\mathsf{N})$ *(resp.* $(\nu b.(\mathsf{out}(c, b) \mid \{^b/_y\})\ ;\mathsf{N})$*) is the* θ_1 *(resp.* θ_2*) concretization of* $(A_s\ ;\mathcal{C}\ ;\mathsf{N}_s)$*. However,* $\nu b.(\mathsf{out}(c, a) \mid \{^b/_y\})$ *is not a concretization of* $(A_s\ ;\mathcal{C}\ ;\mathsf{N}_s)$ *since no* $\theta \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}, \mathsf{N}_s)$ *can have* a *in its image.*

7.2 Symbolic semantics

Symbolic structural equivalence ($\equiv_s$) is the smallest equivalence relation on well-formed symbolic processes such that:

$$\begin{array}{lrcl}
\textsc{Par-0}_s & (A\,;\mathcal{C}\,;\mathsf{N_s}) & \equiv_s & (A \mid 0\,;\mathcal{C}\,;\mathsf{N_s}) \\
\textsc{Par-A}_s & (A \mid (B \mid D)\,;\mathcal{C}\,;\mathsf{N_s}) & \equiv_s & ((A \mid B) \mid D\,;\mathcal{C}\,;\mathsf{N_s}) \\
\textsc{Par-C}_s & (A \mid B\,;\mathcal{C}\,;\mathsf{N_s}) & \equiv_s & (B \mid A\,;\mathcal{C}\,;\mathsf{N_s}) \\
\textsc{New-C}_s & (\nu n.\nu m.A\,;\mathcal{C}\,;\mathsf{N_s}) & \equiv_s & (\nu m.\nu n.A\,;\mathcal{C}\,;\mathsf{N_s})
\end{array}$$

$$\frac{(A\,;\mathcal{C}_A\,;\mathsf{N_s}) \equiv_s (B\,;\mathcal{C}_B\,;\mathsf{N_s})}{(C[A]\,;C[\mathcal{C}_A]\,;\mathsf{N_s}[bn(C[0]) \mapsto \mathsf{b}]) \equiv_s (C[B]\,;C[\mathcal{C}_B]\,;\mathsf{N_s}[bn(C[0]) \mapsto \mathsf{b}])}$$

Symbolic internal reduction $\rightarrow_s$ is the smallest relation on well-formed symbolic processes such that:

$\textsc{Comm}_s$ $\quad (\mathsf{out}(u,M).P \mid \mathsf{in}(v,x).Q \;;\; \mathcal{C} \;;\; \mathsf{N_s}) \rightarrow_s (P \mid Q\{^M/_x\} \;;\; \mathcal{C} \cup \{u = v \;,\; \mathsf{gd}(u) \;,\; \mathsf{gd}(v)\} \;;\; \mathsf{N_s})$
where $u, v \in \mathcal{N}_{ch} \cup (cv(\mathcal{C}) \cap \mathcal{Y}_{ch})$.

$\textsc{Then}_s$ $\quad (\text{if } M = N \text{ then } P \text{ else } Q \;;\; \mathcal{C} \;;\; \mathsf{N_s}) \rightarrow_s (P \;;\; \mathcal{C} \cup \{M = N\} \;;\; \mathsf{N_s})$

$\textsc{Else}_s$ $\quad (\text{if } M = N \text{ then } P \text{ else } Q \;;\; \mathcal{C} \;;\; \mathsf{N_s}) \rightarrow_s (Q \;;\; \mathcal{C} \cup \{M \neq N \;;\; \mathsf{gd}(M) \;;\; \mathsf{gd}(N)\} \;;\; \mathsf{N_s})$

$$\frac{(A \;;\; \mathcal{C}_A \;;\; \mathsf{N_s}) \rightarrow_s (B \;;\; \mathcal{C}_B \;;\; \mathsf{N_s})}{(C[A] \;;\; C[\mathcal{C}_A] \;;\; \mathsf{N_s}[bn(C[0]) \mapsto \mathsf{b}]) \rightarrow_s (C[B] \;;\; C[\mathcal{C}_B] \;;\; \mathsf{N_s}[bn(C[0]) \mapsto \mathsf{b}])}$$

$$\frac{(A \;;\; \mathcal{C}_A \;;\; \mathsf{N_s}) \equiv_s (B \;;\; \mathcal{C}_B \;;\; \mathsf{N_s}) \rightarrow_s (B' \;;\; \mathcal{C}'_B \;;\; \mathsf{N_s}) \equiv_s (B' \;;\; \mathcal{C}'_B \;;\; \mathsf{N_s})}{(A \;;\; \mathcal{C}_A \;;\; \mathsf{N_s}) \rightarrow_s (A' \;;\; \mathcal{C}'_A \;;\; \mathsf{N_s})}$$

In addition to the rules for symbolic structural equivalence and internal reduction, we adopt the following rules:

$\textsc{In}_s$ $\quad (\mathsf{in}(u,x).P \;;\; \mathcal{C} \;;\; \mathsf{N_s}) \xrightarrow{in(u,y)}_s (P\{^y/_x\} \;;\; \mathcal{C} \cup \{0 \Vdash y, \mathsf{gd}(u)\} \;;\; \mathsf{N_s}[y \mapsto \mathsf{c}])$
where $u \in \mathcal{N}_{ch} \cup (\mathcal{Y}_{ch} \cap cv(\mathcal{C}))$, $y \in \mathcal{Y}$, $\mathsf{N_s}(y) = \mathsf{n}$.

$\textsc{Out-Ch}_s$ $\quad (\mathsf{out}(u,v).P \;;\; \mathcal{C} \;;\; \mathsf{N_s}) \xrightarrow{out(u,v)}_s (P \;;\; \mathcal{C} \cup \{\mathsf{gd}(u), \mathsf{gd}(v)\} \;;\; \mathsf{N_s})$
where $u, v \in \mathcal{N}_{ch} \cup (\mathcal{Y}_{ch} \cap cv(\mathcal{C}))$.

$\textsc{Out-T}_s$
$(\mathsf{out}(u,M).P \;;\; \mathcal{C} \;;\; \mathsf{N_s}) \xrightarrow{\nu x.out(u,x)}_s (P \mid \{^M/_x\} \;;\; \nu x.\mathcal{C} \cup \{\mathsf{gd}(u)\} \;;\; \mathsf{N_s}[x \mapsto \mathsf{f}])$
where $x \in \mathcal{X}_b$, $\mathsf{N_s}(x) = \mathsf{n}$.

$\textsc{Open-Ch}_s$
$$\frac{(A \;;\; \mathcal{C} \;;\; \mathsf{N_s}) \xrightarrow{out(u,c)}_s (A' \;;\; \mathcal{C}' \;;\; \mathsf{N'_s}) \qquad u \neq c,\ d \in \mathcal{N}_{ch},\ \mathsf{N_s}(d) = \mathsf{n}}{(\nu c.A \;;\; \nu c.\mathcal{C} \;;\; \mathsf{N_s}[c \mapsto \mathsf{b}]) \xrightarrow{\nu d.out(u,d)}_s (A'\{^d/_c\} \;;\; \nu d.(\mathcal{C}'\{^d/_c\}) \;;\; \mathsf{N'_s}[c \mapsto \mathsf{b},\, d \mapsto \mathsf{f}])}$$

$\textsc{Scope}_s$ $\quad \dfrac{(A \;;\; \mathcal{C} \;;\; \mathsf{N_s}) \xrightarrow{\alpha}_s (A' \;;\; \mathcal{C}' \;;\; \mathsf{N'_s}) \qquad n \text{ does not occur in } \alpha}{(\nu n.A \;;\; \nu n.\mathcal{C} \;;\; \mathsf{N_s}[n \mapsto \mathsf{b}]) \xrightarrow{\alpha}_s (\nu n.A' \;;\; \nu n.\mathcal{C}' \;;\; \mathsf{N_s}[n \mapsto \mathsf{b}])}$

$\textsc{Par}_s$ $\quad \dfrac{(A \;;\; \mathcal{C} \;;\; \mathsf{N_s}) \xrightarrow{\alpha}_s (A' \;;\; \mathcal{C}' \;;\; \mathsf{N'_s})}{(A \mid B \;;\; \mathcal{C} \mid \psi(B) \;;\; \mathsf{N_s}) \xrightarrow{\alpha}_s (A' \mid B \;;\; \mathcal{C}' \mid \psi(B) \;;\; \mathsf{N'_s})}$

$\textsc{Struct}_s$ $\quad \dfrac{(A \;;\; \mathcal{C}_A \;;\; \mathsf{N_s}) \equiv_s (B \;;\; \mathcal{C}_B \;;\; \mathsf{N_s}) \xrightarrow{\alpha}_s (B' \;;\; \mathcal{C}'_B \;;\; \mathsf{N'_s}) \equiv_s (B' \;;\; \mathcal{C}'_B \;;\; \mathsf{N'_s})}{(A \;;\; \mathcal{C}_A \;;\; \mathsf{N_s}) \xrightarrow{\alpha}_s (A' \;;\; \mathcal{C}'_A \;;\; \mathsf{N'_s})}$

For reasons similar to those cited for $\textsc{Open-Ch}_i$, the rules $\textsc{In}_s$ and $\textsc{Open-Ch}_s$ require on-the-fly renaming. When a transition is executed under a context (by the rules $\textsc{Scope}_s$ and $\textsc{Par}_s$) the constraint system must also be put in the context (according to Definition 6.6). In $\textsc{Par}_s$ we avoid the substitution $\psi(B)$

which appears on a label in PAR_i since here we always have that the variables in $\text{dom}(\psi(B))$ do not occur in the label α. In the rule OPEN-CH$_s$, the restriction $\nu d.(\mathcal{C}'\{^d/_c\})$ is needed to ensure that the name d is not used to instantiate the previous inputs: those that are done before the disclosure of this name.

Example 7.5 *To illustrate our symbolic semantics, consider the process* $(A\ ;\ \emptyset\ ;\ \mathsf{N}_s)$ *where* $A = \nu k.\nu s.(\mathsf{in}(c,x).\textit{if } x = s \textit{ then } \mathsf{out}(c,ok) \mid \{^{\mathsf{enc}(s,k)}/_{y_1}\} \mid \{^k/_{y_2}\})$ *and* N_s *is a symbolic environment compatible with* A*. Let* $x' \in \mathcal{Y}_b$ *be a variable such that* $\mathsf{N}_s(x') = \mathsf{n}$*.*

$$\begin{array}{rll}(A\ ;\ \emptyset\ ;\ \mathsf{N}_s) & \xrightarrow{in(c,x')}_s & (A'\ ;\ \{\nu k.\nu s.\{^{\mathsf{enc}(s,k)}/_{y_1}, {}^k/_{y_2}\} \Vdash x'\ ,\ \mathsf{gd}(c)\}\ ;\ \mathsf{N}_s[x' \mapsto \mathsf{c}]) \\ & \longrightarrow_s & (\nu k.\nu s.(\mathsf{out}(c,ok) \mid \{^{\mathsf{enc}(s,k)}/_{y_1}\} \mid \{^k/_{y_2}\})\ ;\ \mathcal{C}\ ;\ \mathsf{N}_s[x' \mapsto \mathsf{c}])\end{array}$$

where $A' = \nu k.\nu s.(\textit{if } x' = s \textit{ then } \mathsf{out}(c,ok) \mid \{^{\mathsf{enc}(s,k)}/_{y_1}\} \mid \{^k/_{y_2}\})$ *and* $\mathcal{C}$ *is the system*

$$\{\nu k.\nu s.\{^{\mathsf{enc}(s,k)}/_{y_1}, {}^k/_{y_2}\} \Vdash x'\ ,\ \mathsf{gd}(c)\ ,\ x' = s\}$$

Let $\theta = \{^{\mathsf{dec}(y_1,y_2)}/_{x'}\}$*. We have* $\theta \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}\ ;\ \mathsf{N}_s[x' \mapsto \mathsf{c}])$ *(see Example 6.5) and*

$$(A\ ;\ \mathsf{N}) \xrightarrow{in(c,x'\theta)}_i \rightarrow_i (A'\theta'\ ;\ \mathsf{N})$$

where $\mathsf{N} = \mathsf{N}_s|_{\mathcal{N}\cup\mathcal{X}}$*.*

8 Soundness and Completeness

We now show soundness of $\equiv_s$, $\rightarrow_s$ and $\xrightarrow{\alpha}_s$ with respect to their counterparts in the intermediate semantics: whenever one of these relations holds between two symbolic processes, the corresponding relation in the intermediate semantics holds for each θ-concretization. The proofs can be found in Appendix B.1.

Proposition 8.1 (soundness of $\equiv_s$ and $\rightarrow_s$) *Let* $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s)$ *and* $(B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}_s)$ *be two well-formed symbolic processes such that* $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s) \bowtie_s (B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}_s)$ *with* $\bowtie\ \in \{\equiv, \rightarrow\}$*. Let* $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_B\ ;\ \mathsf{N}_s)$*. We have that* $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_A\ ;\ \mathsf{N}_s)$ *and* $(A\ ;\ \mathsf{N}) \bowtie_i (B\ ;\ \mathsf{N})$ *where* $(A\ ;\ \mathsf{N})$ *(resp.* $(B\ ;\ \mathsf{N})$*) is the* θ*-concretization of* $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s)$ *(resp.* $(B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}_s)$*).*

Proposition 8.2 (soundness of $\xrightarrow{\alpha}_s$) *Let* $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s)$ *and* $(B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}'_s)$ *be two well-formed symbolic processes such that* $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s) \xrightarrow{\alpha_s}_s (B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}'_s)$*. Let* $\theta_B \in Sol_{\mathsf{E}}(\mathcal{C}_B\ ;\ \mathsf{N}'_s)$ *and* $\theta_A = \theta_B|_{cv(\mathcal{C}_A)}$*. We have that* $\theta_A \in Sol_{\mathsf{E}}(\mathcal{C}_A\ ;\ \mathsf{N}_s)$ *and* $(A\ ;\ \mathsf{N}) \xrightarrow{\alpha_s\theta_B}_i (B\ ;\ \mathsf{N}')$*, where* $(A\ ;\ \mathsf{N})$ *and* $(B\ ;\ \mathsf{N}')$ *are respectively the* θ_A*-concretization and the* θ_B*-concretization of* $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s)$ *and* $(B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}'_s)$*.*

We also show completeness of the symbolic semantics with respect to the intermediate one: each time a θ-concretization of a symbolic process is structurally equivalent, respectively reduces, to another intermediate process, the

symbolic process too is structurally equivalent, respectively reduces, to a corresponding symbolic process. The proofs of these two following propositions can be found in Appendix B.2.

Proposition 8.3 (completeness of $\equiv_s$ and $\rightarrow_s$) *Let $(A_s \ ; \ \mathcal{C}_A \ ; \ \mathsf{N}_s)$ be a well-formed symbolic process and $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_A \ ; \ \mathsf{N}_s)$. Let $(A \ ; \ \mathsf{N})$ be the θ-concretization of $(A_s \ ; \mathcal{C}_A \ ; \ \mathsf{N}_s)$ and (A', N) be an intermediate process such that $(A \ ; \ \mathsf{N}) \bowtie_i (A' \ ; \ \mathsf{N})$ with $\bowtie \in \{\equiv, \rightarrow\}$. Then there exists a well-formed symbolic process $(A'_s \ ; \mathcal{C}'_A \ ; \ \mathsf{N}_s)$ such that:*

1. $(A_s \ ; \mathcal{C}_A \ ; \ \mathsf{N}_s) \bowtie_s (A'_s \ ; \mathcal{C}'_A \ ; \ \mathsf{N}_s)$,
2. $\theta \in Sol_{\mathsf{E}}(\mathcal{C}'_A \ ; \ \mathsf{N}_s)$,
3. $(A' \ ; \ \mathsf{N})$ *is the* θ*-concretization of* $(A'_s \ ; \mathcal{C}'_A \ ; \ \mathsf{N}_s)$.

Proposition 8.4 (completeness of $\xrightarrow{\alpha}_s$) *Let $(A_s \ ; \mathcal{C}_A \ ; \ \mathsf{N}_s)$ be a well-formed symbolic process and $\theta_A \in Sol_{\mathsf{E}}(\mathcal{C}_A \ ; \ \mathsf{N}_s)$. Let $(A \ ; \ \mathsf{N})$ be the θ_A-concretization of $(A_s \ ; \mathcal{C}_A \ ; \ \mathsf{N}_s)$ and $(A' \ ; \ \mathsf{N}')$ be an intermediate process such that $(A \ ; \ \mathsf{N}) \xrightarrow{\alpha}_i (A' \ ; \ \mathsf{N}')$. Then there exists a well-formed symbolic process $(A'_s \ ; \mathcal{C}'_A \ ; \ \mathsf{N}'_s)$ and a substitution θ'_A such that:*

1. $(A_s \ ; \mathcal{C}_A \ ; \ \mathsf{N}_s) \xrightarrow{\alpha_s}_s (A'_s \ ; \mathcal{C}'_A \ ; \ \mathsf{N}'_s)$,
2. $\theta'_A \in Sol_{\mathsf{E}}(\mathcal{C}'_A \ ; \ \mathsf{N}'_s)$ *and* $\theta'_A|_{cv(\mathcal{C}_A)} = \theta_A$,
3. $(A' \ ; \ \mathsf{N}')$ *is the* θ'_A*-concretization of* $(A'_s \ ; \mathcal{C}'_A \ ; \ \mathsf{N}'_s)$, *and*
4. $\alpha_s\theta'_A = \alpha$.

From the propositions stated above, we easily derive the following corollaries.

Corollary 8.5 (soundness of $\rightarrow_s^*$) *Let $(A_s \ ; \mathcal{C}_A \ ; \ \mathsf{N}_s)$ and $(B_s \ ; \mathcal{C}_B \ ; \ \mathsf{N}_s)$ be two well-formed symbolic processes such that $(A_s \ ; \mathcal{C}_A \ ; \ \mathsf{N}_s) \rightarrow_s^* (B_s \ ; \mathcal{C}_B \ ; \ \mathsf{N}_s)$. Let $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_B \ ; \ \mathsf{N}_s)$. We have that $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_A \ ; \ \mathsf{N}_s)$ and $(A \ ; \ \mathsf{N}) \rightarrow_i^* (B \ ; \ \mathsf{N})$ where $(A \ ; \ \mathsf{N})$ (resp. $(B \ ; \ \mathsf{N})$) is the θ-concretization of $(A_s \ ; \ \mathcal{C}_A \ ; \ \mathsf{N}_s)$ (resp. $(B_s \ ; \mathcal{C}_B \ ; \ \mathsf{N}_s)$).*

Corollary 8.6 (soundness of $\rightarrow_s^* \xrightarrow{\alpha}_s \rightarrow_s^*$) *Let $(A_s \ ; \mathcal{C}_A \ ; \ \mathsf{N}_s)$ and $(B_s \ ; \mathcal{C}_B \ ; \ \mathsf{N}'_s)$ be two well-formed symbolic processes such that $(A_s \ ; \ \mathcal{C}_A \ ; \ \mathsf{N}_s) \rightarrow_s^* \xrightarrow{\alpha_s}_s \rightarrow_s^* (B_s \ ; \ \mathcal{C}_B \ ; \ \mathsf{N}'_s)$. Let $\theta_B \in Sol_{\mathsf{E}}(\mathcal{C}_B \ ; \mathsf{N}'s)$ and $\theta_A = \theta_B|_{cv(\mathcal{C}_A)}$. We have that $\theta_A \in Sol_{\mathsf{E}}(\mathcal{C}_A \ ; \ \mathsf{N}_s)$ and $(A \ ; \ \mathsf{N}) \rightarrow_i^* \xrightarrow{\alpha_s\theta_B}_i \rightarrow_i^* (B \ ; \ \mathsf{N}')$, where $(A \ ; \ \mathsf{N})$ and $(B \ ; \ \mathsf{N}')$ are respectively the θ_A-concretization and the θ_B-concretization of $(A_s \ ; \mathcal{C}_A \ ; \ \mathsf{N}_s)$ and $(B_s \ ; \mathcal{C}_B \ ; \ \mathsf{N}'_s)$.*

9 Symbolic Equivalences

In this section, we define our notion of symbolic static equivalence and our notion of symbolic bisimulation. We also show the soundness of these equivalences w.r.t. their intermediate counterparts. We also show in Section 10 that our symbolic bisimulation is *not* complete.

We define symbolic static equivalence using an encoding similar to the one in [5]. The tests used to distinguish two frames in the definition of static equivalence are encoded by means of two additional deduction constraints on fresh variables x, y and by the equation $x = y$.

Definition 9.1 (symbolic static equivalence) *We say that two closed well-formed symbolic processes* $(A_s \; ; \mathcal{C}_A \; ; \mathsf{N}_s)$ *and* $(B_s \; ; \mathcal{C}_B \; ; \mathsf{N}_s)$ *are symbolically statically equivalent, written* $(A_s \; ; \mathcal{C}_A \; ; \mathsf{N}_s) \sim_s (B_s \; ; \mathcal{C}_B \; ; \mathsf{N}_s)$ *if for some variables* $x, y \in \mathcal{Y}_b$*, the constraint systems* $\mathcal{C}'_A$*,* $\mathcal{C}'_B$ *have the same basis and* $Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_A \; ; \mathsf{N}'_s) = Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_B \; ; \mathsf{N}'_s)$ *where*

- $\mathsf{N}_s(\{x, y\}) = \mathsf{n}$,
- $\mathsf{N}'_s = \mathsf{N}_s[x, y \mapsto \mathsf{c}]$,
- $\mathcal{C}'_A = \mathcal{C}_A \cup \{\phi(A_s) \Vdash x \; , \phi(A_s) \Vdash y \; , x = y\}$, *and*
- $\mathcal{C}'_B = \mathcal{C}_B \cup \{\phi(B_s) \Vdash x \; , \phi(B_s) \Vdash y \; , x = y\}$.

The following proposition states the correctness of the symbolic static equivalence with respect to the concrete one.

Proposition 9.2 (soundness of symbolic static equivalence) *Let* $(A_s \; ; \mathcal{C}_A \; ; \mathsf{N}_s)$ *and* $(B_s \; ; \mathcal{C}_B \; ; \mathsf{N}_s)$ *be two closed and well-formed symbolic processes such that* $(A_s \; ; \mathcal{C}_A \; ; \mathsf{N}_s) \sim_s (B_s \; ; \mathcal{C}_B \; ; \mathsf{N}_s)$*. Then we have that:*

1. $Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}_A \; ; \mathsf{N}_s) = Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}_B \; ; \mathsf{N}_s)$*, and*
2. *for all* $\theta \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}_A \; ; \mathsf{N}_s)$ *we have that* $\phi(A_s(\theta\sigma_A)^\star) \sim \phi(B_s(\theta\sigma_B)^\star)$*, where* σ_A *(resp.* σ_B*) is the substitution corresponding to the maximal frame of* $\mathcal{C}_A$ *(resp.* $\mathcal{C}_B$*).*

Proof.

1. We show one direction, i.e., $Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}_A \; ; \mathsf{N}_s) \subseteq Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}_B \; ; \mathsf{N}_s)$. The other one can be proved in a similar way. Let $\theta \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}_A \; ; \mathsf{N}_s)$. Since $(A_s \; ; \mathcal{C}_A \; ; \mathsf{N}_s) \sim_s (B_s \; ; \mathcal{C}_B \; ; \mathsf{N}_s)$, we know that $\mathcal{C}'_A$, $\mathcal{C}'_B$ have same basis and $Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_A \; ; \mathsf{N}'_s) = Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_B \; ; \mathsf{N}'_s)$ where
   - $x, y \in \mathcal{Y}_b$ and $\mathsf{N}_s(\{x, y\}) = \mathsf{n}$,
   - $\mathsf{N}'_s = \mathsf{N}_s[x, y \mapsto \mathsf{c}]$,
   - $\mathcal{C}'_A = \mathcal{C}_A \cup \{\phi(A_s) \Vdash x \; , \phi(A_s) \Vdash y \; , x = y\}$, and

- $\mathcal{C}'_B = \mathcal{C}_B \cup \{\phi(B_s) \Vdash x\ ,\ \phi(B_s) \Vdash y\ ,\ x = y\}$.

Let $\rho = \{x \mapsto a, y \mapsto a\}$ for some name a such that $\mathsf{N}_s(a) = \mathsf{f}$ and that does not occur in $A_s, \mathcal{C}_A, B_s, \mathcal{C}_B$. It is easy to see that $\theta \cup \rho \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_A\ ;\ \mathsf{N}'_s)$. Since $Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_A\ ;\ \mathsf{N}'_s) = Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_B\ ;\ \mathsf{N}'_s)$, we deduce that $\theta \cup \rho \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_B\ ;\ \mathsf{N}'_s)$. It remains to check that $\theta \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}_B\ ;\ \mathsf{N}_s)$.

Let $\phi(B_s) = \nu\tilde{n}'_B.\sigma'_B$. By hypothesis, we know that the idempotent substitution σ obtained by iterating $(\theta \cup \rho)\sigma'_B$ satisfies the constraints in $\mathcal{C}'_B$. Let $\nu\tilde{n}_A.\sigma_A$ be the maximal frame in $\mathcal{C}_A$ and $\nu\tilde{n}_B.\sigma_B$ be the maximal frame in $\mathcal{C}_B$. Since $\mathcal{C}_A$ and $\mathcal{C}_B$ have same basis, we have that $\mathrm{dom}(\sigma_A) = \mathrm{dom}(\sigma_B)$. Moreover, since $\theta \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}_A\ ;\ \mathsf{N}_s)$, we know that $\mathit{vars}(\mathrm{img}(\theta)) \subseteq \mathrm{dom}(\sigma_A) = \mathrm{dom}(\sigma_B)$. Since $\sigma = ((\theta \cup \rho)\sigma'_B)^\star$, we deduce that $\sigma = ((\theta \cup \rho)\sigma_B)^\star$, and actually $\sigma = (\theta\sigma_B)^\star \cup \rho$. This means that $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_B\ ;\ \mathsf{N}_s)$ since x and y do not appear in $\mathcal{C}_B$.

2. Let $\theta \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}_A\ ;\ \mathsf{N}_s)$. Let σ_A be the substitution corresponding to the maximal frame of $\mathcal{C}_A$. Let $A = A_s(\theta\sigma_A)^\star$. Note that A is a closed intermediate extended process and $\phi(A)$ is a closed frame. Thanks to the previous point, we also have that $\theta \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}_B\ ;\ \mathsf{N}_s)$. Let σ_B be the substitution corresponding to the maximal frame of $\mathcal{C}_B$. Let $B = B_s(\theta\sigma_B)^\star$. We have that B is a closed intermediate extended process and $\phi(B)$ is a closed frame. Assume that $\phi(A) \not\sim \phi(B)$. Since we have $\mathrm{dom}(\phi(A)) = \mathrm{dom}(\phi(B))$, this means that there exist two terms M and N such that

   - $\mathit{fv}(M, N) \subseteq \mathrm{dom}(\phi(A)) = \mathrm{dom}(\phi(A_s))$,
   - $\mathit{fn}(M, N) \cap (\mathit{bn}(A) \cup \mathit{bn}(B)) = \emptyset$, and
   - $(M\sigma'_A)(\theta\sigma_A)^\star =_{\mathsf{E}} (N\sigma'_A)(\theta\sigma_A)^\star$ and $(M\sigma'_B)(\theta\sigma_B)^\star \neq_{\mathsf{E}} (N\sigma'_B)(\theta\sigma_B)^\star$ (or vice-versa), where $\phi(A_s) = \nu\tilde{n}'_A.\sigma'_A$ and $\phi(B_s) = \nu\tilde{n}'_B.\sigma'_B$.

   Hence, we have that $\mathsf{N}'_s(\mathit{fv}(M, N)) = \mathsf{f}$ and we can also assume w.l.o.g. that $\mathsf{N}'_s(\mathit{fn}(M, N)) = \mathsf{f}$. Now, let $\rho = \{x \mapsto M,\ y \mapsto N\}$. First note that $\theta \cup \rho$ is closed w.r.t. $\mathcal{C}'_A$ and $\mathcal{C}'_B$. It remains to show that $\theta \cup \rho \in Sol_{\mathsf{E}}(\mathcal{C}'_A\ ;\ \mathsf{N}'_s)$ whereas $\theta \cup \rho \notin Sol_{\mathsf{E}}(\mathcal{C}'_B\ ;\ \mathsf{N}'_s)$ obtaining in this way a contradiction.

   - $\theta \cup \rho \in Sol_{\mathsf{E}}(\mathcal{C}'_A\ ;\ \mathsf{N}'_s)$.
     We want to show that $((\theta \cup \rho)\sigma'_A)^\star$ satisfies the constraints in $\mathcal{C}'_A$. We have that $((\theta \cup \rho)\sigma'_A)^\star = (\theta\sigma_A)^\star \cup (\rho\sigma'_A)(\theta\sigma_A)^\star$. By hypothesis, we know that $(\theta\sigma_A)^\star$ satisfies the constraints in $\mathcal{C}_A$. Hence, we conclude for the constraint in $\mathcal{C}_A$. We have also that $x((\theta \cup \rho)\sigma'_A)^\star =_{\mathsf{E}} y((\theta \cup \rho)\sigma'_A)^\star$, since we know that $(M\sigma'_A)(\theta\sigma_A)^\star =_{\mathsf{E}} (N\sigma'_A)(\theta\sigma_A)^\star$.
   - $\theta \cup \rho \notin Sol_{\mathsf{E}}(\mathcal{C}'_B\ ;\ \mathsf{N}'_s)$.
     We show that $((\theta \cup \rho)\sigma'_B)^\star$ does not satisfy the constraint $x = y$. We have that $x((\theta \cup \rho)\sigma'_B)^\star = (M\sigma'_B)(\theta\sigma_B))^\star$ and $y((\theta \cup \rho)\sigma'_B)^\star = (N\sigma'_B)(\theta\sigma_B)^\star$. We know that $(M\sigma'_B)(\theta\sigma_B)^\star) \neq_{\mathsf{E}} (N\sigma'_B)(\theta\sigma_B)^\star$. This allows us to conclude.

□

Although we do not need completeness of symbolic static equivalence for our result, we may note that it follows from Baudet's result [5]. By completeness we mean that $A \sim B$ implies that $(A\ ;\ \emptyset\ ;\ \mathsf{N}_s) \sim_s (B\ ;\ \emptyset\ ;\ \mathsf{N}_s)$ for any compatible naming environment N_s.
We now define symbolic labelled bisimulation using our symbolic semantics.

Definition 9.3 (Symbolic labelled bisimilarity ($\approx_s$)) *Symbolic labelled bisimilarity is the largest symmetric relation $\mathcal{R}$ on closed well-formed symbolic processes with same naming environment, such that $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s)\ \mathcal{R}\ (B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}_s)$ implies*

1. $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s) \sim_s (B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}_s)$
2. *if $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s) \rightarrow_s (A'_s\ ;\ \mathcal{C}'_A\ ;\ \mathsf{N}_s)$ with $Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_A\ ;\ \mathsf{N}_s) \neq \emptyset$, then there exists a symbolic process $(B'_s\ ;\ \mathcal{C}'_B\ ;\ \mathsf{N}_s)$ such that*
   - $(B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}_s) \rightarrow^*_s (B'_s\ ;\ \mathcal{C}'_B\ ;\ \mathsf{N}_s)$, *and*
   - $(A'_s\ ;\ \mathcal{C}'_A\ ;\ \mathsf{N}_s)\ \mathcal{R}\ (B'_s\ ;\ \mathcal{C}'_B\ ;\ \mathsf{N}_s)$;
3. *if $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s) \xrightarrow{\alpha_s}_s (A'_s\ ;\ \mathcal{C}'_A\ ;\ \mathsf{N}'_s)$ with $Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_A\ ;\ \mathsf{N}'_s) \neq \emptyset$, then there exists a symbolic process $(B'_s\ ;\ \mathcal{C}'_B\ ;\ \mathsf{N}'_s)$ such that*
   - $(B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}_s) \rightarrow^*_s \xrightarrow{\alpha_s}_s \rightarrow^*_s (B'_s\ ;\ \mathcal{C}'_B\ ;\ \mathsf{N}'_s)$, *and*
   - $(A'_s\ ;\ \mathcal{C}'_A\ ;\ \mathsf{N}'_s)\ \mathcal{R}\ (B'_s\ ;\ \mathcal{C}'_B\ ;\ \mathsf{N}'_s)$.

The side condition $Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_A\ ;\ \mathsf{N}'_s) \neq \emptyset$ ensures that we only consider symbolic executions that correspond to at least one concrete execution. The following theorem states the soundness of the symbolic bisimulation with respect to the intermediate one.

Theorem 9.4 (soundness of symbolic labelled bisimilarity) *Let $(A\ ;\ \mathsf{N})$ and $(B\ ;\ \mathsf{N})$ be two intermediate processes. Let N_s be a symbolic naming environment such that $\mathsf{N}_s|_{\mathcal{N}\cup\mathcal{X}} = \mathsf{N}$ and $\mathsf{N}_s(y) = \mathsf{n}$ for all $y \in \mathcal{Y}$. We have that*

$$(A\ ;\ \emptyset\ ;\ \mathsf{N}_s) \approx_s (B\ ;\ \emptyset\ ;\ \mathsf{N}_s) \Rightarrow (A\ ;\ \mathsf{N}) \approx_i (B\ ;\ \mathsf{N})$$

Proof. To prove this result, first we define a new relation $\mathcal{R}'$ and then we will show that $\mathcal{R}'$ is an intermediate labelled bisimulation witnessing $(A\ ;\ \mathsf{N}) \approx_i (B\ ;\ \mathsf{N})$. Let $\mathcal{R}$ be the relation witnessing $(A\ ;\ \emptyset\ ;\ \mathsf{N}_s) \approx_s (B\ ;\ \emptyset\ ;\ \mathsf{N}_s)$.

(i) Definition of $\mathcal{R}'$.
$(A\ ;\ \mathsf{N})\ \mathcal{R}'\ (B\ ;\ \mathsf{N})$ if there exists two closed well-formed symbolic processes $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s)$ and $(B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}_s)$ such that

- $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s)\ \mathcal{R}\ (B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}_s)$ with $\mathsf{N} = \mathsf{N}_s|_{\mathcal{N}\cup\mathcal{X}}$, and
- there exists $\theta \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}_A\ ;\ \mathsf{N}_s)$ such that $A_s(\theta\sigma_A)^\star = A$ and $B_s(\theta\sigma_B)^\star = B$ where σ_A (resp. σ_B) is the maximal frame of $\mathcal{C}_A$ (resp. $\mathcal{C}_B$).

(ii) $\mathcal{R}'$ is an intermediate bisimulation relation witnessing $(A \,;\, \mathsf{N}) \approx_i (B \,;\, \mathsf{N})$. First we have to show that $(A \,;\, \mathsf{N})\ \mathcal{R}'\ (B \,;\, \mathsf{N})$. To do this, it is sufficient to see that the two well-formed symbolic processes $(A \,;\, \emptyset \,;\, \mathsf{N}_s)$ and $(B \,;\, \emptyset \,;\, \mathsf{N}_s)$ satisfy the required conditions.

Now, we have to show that $\mathcal{R}'$ satisfies the three points of the definition of intermediate labelled bisimilarity. Let $(A \,;\, \mathsf{N})$ and $(B \,;\, \mathsf{N})$ be two closed intermediate processes such that $(A \,;\, \mathsf{N})\ \mathcal{R}'\ (B \,;\, \mathsf{N})$. By definition of $\mathcal{R}'$, we know that there exists two closed well-formed symbolic processes $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ and $(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s)$ such that

- $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)\ \mathcal{R}\ (B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s)$, with $\mathsf{N} = \mathsf{N}_s|_{\mathcal{N} \cup \mathcal{X}}$, and
- there exists $\theta \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}_A \,;\, \mathsf{N}_s)$ such that $A_s(\theta\sigma_A)^\star = A$ and $B_s(\theta\sigma_B)^\star = B$.

We have to show that:

1. $\phi(A) \sim \phi(B)$.
   Thanks to Proposition 9.2, we deduce that
   - $Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}_A \,;\, \mathsf{N}_s) = Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}_B \,;\, \mathsf{N}_s)$, and
   - for all $\theta' \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}_A \,;\, \mathsf{N}_s)$ we have $\phi(A_s(\theta'\sigma_A)^\star) \sim \phi(B_s(\theta'\sigma_B)^\star)$.

   Since $\theta \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}_A \,;\, \mathsf{N}_s)$ we deduce that $\phi(A_s(\theta\sigma_A)^\star) \sim \phi(B_s(\theta\sigma_B)^\star)$, *i.e* $\phi(A) \sim \phi(B)$.

2. If $(A \,;\, \mathsf{N}) \rightarrow_i (A' \,;\, \mathsf{N})$, then there exists $(B' \,;\, \mathsf{N})$ such that $(B \,;\, \mathsf{N}) \rightarrow_i^* (B' \,;\, \mathsf{N})$ and $(A' \,;\, \mathsf{N})\ \mathcal{R}'\ (B' \,;\, \mathsf{N})$.
   By definition of $\mathcal{R}'$, we know that $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ is a closed well-formed symbolic process such that $A_s(\theta\sigma_A)^\star = A$ and $\theta \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}_A \,;\, \mathsf{N}_s)$. Hence, thanks to Proposition 8.3, we know that there exists a well-formed symbolic process $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}_s)$ such that
   - $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s) \rightarrow_s (A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}_s)$,
   - $\theta \in Sol_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}_s)$, and
   - $A'_s(\theta\sigma'_A)^\star = A'$.

   We have that $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)\ \mathcal{R}\ (B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s)$ and $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s) \rightarrow_s (A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}_s)$. Moreover $\theta \in Sol_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}_s)$, and actually $\theta \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}_s)$, thus we know that $Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}_s) \neq \emptyset$. Hence, there exists a closed well-formed symbolic process $(B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}_s)$ such that
   - $(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s) \rightarrow_s^* (B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}_s)$, and
   - $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}_s)\ \mathcal{R}\ (B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}_s)$.

   Since $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}_s)\ \mathcal{R}\ (B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}_s)$, we deduce that $Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}_s) = Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_B \,;\, \mathsf{N}_s)$ by using Proposition 9.2. We have that $\theta \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}_s)$ and hence, we deduce that $\theta \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_B \,;\, \mathsf{N}_s)$. Now, by Corollary 8.5, we deduce that $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_B \,;\, \mathsf{N}_s)$ and $(B_s(\theta\sigma_B)^\star \,;\, \mathsf{N}) \rightarrow^* (B'_s(\theta\sigma'_B)^\star \,;\, \mathsf{N})$. Let

$B' = B'_s(\theta\sigma'_B)^\star$. As $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}_s)$ and $(B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}_s)$ are two closed well-formed symbolic processes such that $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}_s)\ \mathcal{R}\ (B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}_s)$ and $A'_s(\theta\sigma'_A)^\star = A'$ and $B'_s(\theta\sigma'_B)^\star = B'$, we have that $(A' \,;\, \mathsf{N})\ \mathcal{R}'\ (B' \,;\, \mathsf{N})$.

3. If $(A \,;\, \mathsf{N}) \xrightarrow{\alpha}_i (A' \,;\, \mathsf{N}')$ with $fv(\alpha) \subseteq \mathrm{dom}(A)$ then $(B \,;\, \mathsf{N}) \rightarrow_i^* \xrightarrow{\alpha}_i \rightarrow_i^* (B' \,;\, \mathsf{N}')$ and $(A' \,;\, \mathsf{N}')\ \mathcal{R}'\ (B' \,;\, \mathsf{N}')$ for some B'.

   By definition of $\mathcal{R}'$, we know that $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ is a closed well-formed symbolic process such that $A_s(\theta\sigma_A)^\star = A$ and $\theta \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}_A \,;\, \mathsf{N}_s)$. Hence, thanks to Proposition 8.4, we know that there exist a well-formed symbolic process $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s)$, a substitution θ' and a label α_s such that

   - $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s) \xrightarrow{\alpha_s}_s (A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s)$ and $\mathsf{N}' = \mathsf{N}'_s|_{\mathcal{N}\cup\mathcal{X}}$,
   - $\theta' \in Sol_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}'_s)$ and $\theta'|_{cv(\mathcal{C}_A)} = \theta$,
   - $A'_s(\theta'\sigma'_A)^\star = A'$ where σ'_A is the substitution corresponding to the maximal frame in $\mathcal{C}'_A$, and
   - $\alpha_s\theta' = \alpha$.

   Actually, we have that $\theta' \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}'_s)$, i.e. θ' is a closed solution. This is clear when the label α is not an input. In the case of an input, α_s is of the form $in(c, y)$ and we conclude by relying on the fact that $vars(y\theta') = fv(\alpha) \subseteq \mathrm{dom}(A)$.

   We have that $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)\ \mathcal{R}\ (B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s)$ and $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s) \xrightarrow{\alpha}_s (A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s)$. Since $\theta' \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}'_s)$, we know that $Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}'_s) \neq \emptyset$. Hence, there exists a closed well-formed symbolic process $(B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)$ such that

   - $(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s) \rightarrow_s^* \xrightarrow{\alpha_s}_s \rightarrow_s^* (B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)$, and
   - $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s)\ \mathcal{R}\ (B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)$.

   Since $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s)\ \mathcal{R}\ (B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)$, thanks to Proposition 9.2, we have that $Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}'_s) = Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_B \,;\, \mathsf{N}'_s)$. We have that $\theta' \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}'_s)$ and hence, we deduce that $\theta' \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\mathcal{C}'_B \,;\, \mathsf{N}'_s)$. Now, by Corollary 8.6, we deduce that $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_B \,;\, \mathsf{N}_s)$ and $(B_s(\theta\sigma_B)^\star \,;\, \mathsf{N}) \rightarrow^* (B'_s(\theta'\sigma'_B)^\star \,;\, \mathsf{N})$.

   As $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s)$ and $(B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)$ are two closed well-formed symbolic processes such that $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s)\ \mathcal{R}\ (B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)$ and $A'_s(\theta'\sigma'_A)^\star = A'$ and $B'_s(\theta'\sigma'_B)^\star = B'$, we have that $(A' \,;\, \mathsf{N})\ \mathcal{R}'\ (B' \,;\, \mathsf{N})$. This allows us to conclude.

□

PART III
— Soundness of Symbolic Bisimulation —

We now put the results of the previous section together (Theorem 5.2 and Theorem 9.4) to prove our main result.

Theorem 9.5 (Soundness of symbolic bisimulation) *Let A and B be two closed, nv-distinct extended processes. For any symbolic naming environment N_s compatible with $A{\downarrow}$, $B{\downarrow}$ and the empty constraint system we have that*

$$(A{\downarrow}\ ;\ \emptyset\ ;\ \mathsf{N}_s) \approx_s (B{\downarrow}\ ;\ \emptyset\ ;\ \mathsf{N}_s) \text{ implies } A \approx B$$

Note that limiting the theorem to nv-distinct processes is not an onerous restriction. If we want to prove that $A \approx B$, we can construct by α-conversion two nv-distinct processes A', B' such that $A' \equiv A$ and $B' \equiv B$. Showing $A' \approx B'$ implies that $A \approx B$, since $\approx$ is closed under structural equivalence.

10 Discussion

Our techniques suffer from the same sources of incompleteness as the ones described for the spi calculus in [10]. In a symbolic bisimulation the instantiation of input variables is postponed until the point at which they are actually used, leading to a finer relation. We illustrate this point on an example, similar to one given in [10].

Example 10.1 *Consider the two following processes:*

$$\begin{array}{rcl} P_1 & = & \nu c_1.\mathsf{in}(c_2, x).(\mathsf{out}(c_1, b) \mid \mathsf{in}(c_1, y) \mid \textit{if } x = a \textit{ then } \mathsf{in}(c_1, z).\mathsf{out}(c_2, a)) \\ Q_1 & = & \nu c_1.\mathsf{in}(c_2, x).(\mathsf{out}(c_1, b) \mid \mathsf{in}(c_1, y) \mid \mathsf{in}(c_1, z).\textit{if } x = a \textit{ then } \mathsf{out}(c_2, a)) \end{array}$$

We have that $P_1 \approx Q_1$ whereas $(P_1\ ;\ \emptyset\ ;\ \mathsf{N}_s) \not\approx_s (Q_1\ ;\ \emptyset\ ;\ \mathsf{N}_s)$ for any compatible naming environment N_s. To see the latter inequivalence, observe that $(Q_1\ ;\ \emptyset\ ;\ \mathsf{N}_s)$ can make the transition $\xrightarrow{\mathsf{in}(c_2, x')}_s$ and then an internal transition to $(\nu c_1.(\mathsf{in}(c_1, y) \mid \textit{if } x' = a \textit{ then } \mathsf{out}(c_2, a))\ ;\ \mathcal{C}\ ;\ \mathsf{N}'_s)$ with $\mathcal{C} = \{0 \Vdash x'\ ,\ \mathsf{gd}(c_2)\}$; this process is still undecided about whether $x' = a$ or not. $(P_1\ ;\ \emptyset\ ;\ \mathsf{N}_s)$ can make the transition $\xrightarrow{\mathsf{in}(c_2, x')}_s$ but then cannot make the corresponding internal transition without committing either to the constraint $x' = a$ or to the constraint $x' \neq a$. Whichever one it does, Q_1 can do the opposite, showing the inequivalence.

The second example shows that the requirement that the constraint systems must have the same solutions gives rise to some incompleteness.

Example 10.2 *Consider the two following processes:*

$$\begin{array}{rcl} P_2 & = & \mathsf{in}(c,x).\mathsf{out}(c,a) \\ Q_2 & = & \mathsf{in}(c,x).\mathit{if}\ x = a\ \mathit{then}\ \mathsf{out}(c,a)\ \mathit{else}\ \mathsf{out}(c,a) \end{array}$$

We have that $P_2 \approx Q_2$ *whereas* $(P_2 \,;\, \emptyset \,;\, \mathsf{N}_s) \not\approx_s (Q_2 \,;\, \emptyset \,;\, \mathsf{N}_s)$ *for any compatible naming environment* N_s*. Indeed, we have that* $(P_2 \,;\, \emptyset \,;\, \mathsf{N}_s) \xrightarrow{\mathsf{in}(c,x')}_s (\mathsf{out}(c,a) \,;\, \mathcal{C} \,;\, \mathsf{N}'_s)$ *with* $\mathcal{C} = \{0 \Vdash x' \,,\, \mathsf{gd}(c)\}$*) whereas* $(Q_2 \,;\, \emptyset \,;\, \mathsf{N}_s) \xrightarrow{\mathsf{in}(c,x')}_s$ *if* $x = a$ *then* $\mathsf{out}(c,a)$ *else* $\mathsf{out}(c,a)$ *and then can move to either* $(\mathsf{out}(c,a) \,;\, \mathcal{C}_1 \,;\, \mathsf{N}'_s)$ *or* $(\mathsf{out}(c,a) \,;\, \mathcal{C}_2 \,;\, \mathsf{N}'_s)$ *where* $\mathcal{C}_1 = \{0 \Vdash x' \,,\, \mathsf{gd}(c) \,,\, x' = a\}$ *and* $\mathcal{C}_2 = \{0 \Vdash x' \,,\, \mathsf{gd}(c) \,,\, \mathsf{gd}(x') \,,\, x' \neq a\}$*. However, neither* $\mathcal{C}_1$ *nor* $\mathcal{C}_2$ *is equivalent to* $\mathcal{C}$*.*

Although our symbolic bisimulation is not complete, as shown above, we are able to prove labelled bisimulation on interesting examples for which the method implemented in the state-of-the-art ProVerif tool [7] fails. For instance, ProVerif is unable to establish labelled bisimilarity between $\mathsf{out}(c,a) \mid \mathsf{out}(c,b)$ and $\mathsf{out}(c,b) \mid \mathsf{out}(c,a)$ whereas of course we are able to deal with such examples. A more interesting example, for which our symbolic semantics plays an important role is as follows.

Example 10.3 *Consider the following two processes.*

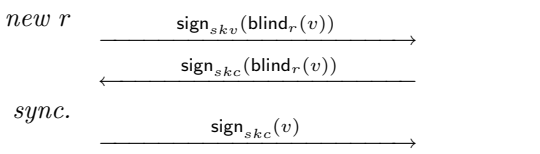
$$\begin{array}{rcl} P & = & \nu c_1.(\mathsf{in}(c_2,x).\mathsf{out}(c_1,x).\mathsf{out}(c_2,a) \mid \mathsf{in}(c_1,y).\mathsf{out}(c_2,y)) \\ Q & = & \nu c_1.(\mathsf{in}(c_2,x).\mathsf{out}(c_1,x).\mathsf{out}(c_2,x) \mid \mathsf{in}(c_1,y).\mathsf{out}(c_2,a)) \end{array}$$

These two processes are labelled bisimilar and our symbolic labelled bisimulation is complete enough to prove this. In particular, let $P' = \nu c_1.(\mathsf{out}(c_1,x').\mathsf{out}(c_2,a) \mid \mathsf{in}(c_1,y).\mathsf{out}(c_2,y))$ *and* $Q' = \nu c_1.(\mathsf{out}(c_1,x').\mathsf{out}(c_2,x') \mid \mathsf{in}(c_1,y).\mathsf{out}(c_2,a))$*. The relation* $\mathcal{R}$*, that witnesses the symbolic bisimulation, includes*

$$\begin{array}{rcl} (P \,;\, \emptyset \,;\, \mathsf{N}_s) & \mathcal{R} & (Q \,;\, \emptyset \,;\, \mathsf{N}_s) \\ (P' \,;\, \{\nu c_1.0 \Vdash x' \,,\, \mathsf{gd}(c_2)\} \,;\, \mathsf{N}'_s) & \mathcal{R} & (Q' \,;\, \{\nu c_1.0 \Vdash x' \,,\, \mathsf{gd}(c_2)\} \,;\, \mathsf{N}'_s) \\ (\nu c_1.(\mathsf{out}(c_2,a) \mid \mathsf{out}(c_2,x')) \,; \\ \{\nu c_1.0 \Vdash x' \,,\, \mathsf{gd}(c_2) \,,\, \mathsf{gd}(c_1)\} \,;\, \mathsf{N}'_s) & \mathcal{R} & (\nu c_1.(\mathsf{out}(c_2,x') \mid \mathsf{out}(c_2,a)) \,; \\ & & \{\nu c_1.0 \Vdash x' \,,\, \mathsf{gd}(c_2) \,,\, \mathsf{gd}(c_1)\} \,;\, \mathsf{N}'_s) \end{array}$$

The example above is inspired by the problems we encountered when we analysed a bisimulation representing the privacy property in an electronic voting protocol [19]. ProVerif is not able to prove this kind of equivalence; its algorithm is limited to cases that the two processes P, Q have the same structure, and differ only in the terms that are output. In this example, the processes differ in their structure, providing the motivation for our methods. Our symbolic bisimulation seems to be "sufficiently" complete to deal with examples of privacy and anonymity properties arising in protocol analysis. We demonstrate that more fully with the next example, which considers the privacy property of voting systems in more detail, and illustrates the equational reasoning aspects of the calculus.

Example 10.4 *We consider a simplified version of the voting protocol due to Fujioka, Okamoto and Ohta (see [17]) that is analysed in [19, 12]. In the simplification we consider here, the voter casts a vote in the first phase of the voting process by blinding his selected candidate v with a random value r, and signing the result with his private key. He sends this signature to the collector. Using this signature, the collector is able to check that the voter is entitled to vote, and the collector sends back his signature on the blinded choice of the voter. The voter now unblinds this value, obtaining the collector's signature on his vote. In the second phase, he anonymously sends this signature to the collector for counting.*



The blinding operation allows signatures to be performed blindly. Here, the collector signs the vote, but is not able to see its value. This helps to achieve the property of vote-privacy for the voter. To avoid traffic analysis attacks, the protocol is in two phases; the voter synchronises with other voters between the two phases (represented by "sync." in the figure). This synchronisation can easily be modelled using private channels, but we prefer to omit that detail to keep the example simple. Although it satisfies vote-privacy, this simple protocol would allow a voter to vote multiple times by repeatedly sending the last message. That problem is easily fixed, but we prefer to keep the protocol simple for the purpose of illustration.

We assume a signature containing the binary functions sign, getmess, blind, unblind, *and the unary function* pk *with the equations:*

$$\begin{aligned} \mathsf{getmess}_{\mathsf{pk}(x)}(\mathsf{sign}_x(y)) &= y \\ \mathsf{unblind}_x(\mathsf{sign}_y(\mathsf{blind}_x(z))) &= \mathsf{sign}_y(z) \end{aligned}$$

Note that key arguments are written as subscripts, to aid readability. A voter with signing key skv casting the vote v for the collector with public key pkc runs the process P described below. (Since all the communications take place over a public channel, we do not mention it for sake of readability.)

$$\begin{aligned} P(skv, v, pkc) = \nu r.\ (&\mathsf{out}(\mathsf{sign}_{skv}(\mathsf{blind}_r(v))).\mathsf{in}(x). \\ &\textit{if } \mathsf{getmess}_{pkc}(x) = \mathsf{blind}_r(v) \textit{ then } \mathsf{out}(\mathsf{unblind}_r(x))) \end{aligned}$$

The anonymity property we want to prove says that an observer (which may include the collector) cannot distinguish a situation in which the voter A votes v_a and the voter B votes v_b, from another one in which they vote the other way around. Roughly speaking, it is the following labelled bisimilarity:

$$\begin{aligned} &\nu ska, skb(\mathsf{out}(\mathsf{pk}(ska)).\mathsf{out}(\mathsf{pk}(skb)).(P(ska, v_a, \mathsf{pk}(skc)) \mid P(skb, v_b, \mathsf{pk}(skc)))) \\ &\qquad\qquad \approx \\ &\nu ska, skb(\mathsf{out}(\mathsf{pk}(ska)).\mathsf{out}(\mathsf{pk}(skb)).(P(ska, v_b, \mathsf{pk}(skc)) \mid P(skb, v_a, \mathsf{pk}(skc)))) \end{aligned} \tag{1}$$

with the proviso that the voters A and B have to synchronise at the "sync." point. Note that we treat the collector's private key skc as a public name; we prove privacy property even in presence of a corrupted collector who disclosed his private key.

Below, we illustrate some of the calculations to establish this equivalence (of course we prove $\approx_s$ to establish $\approx$). We will only consider deducibility and equality constraints and we do not give the naming environment associated to each symbolic process.

intermediate process	*some of the constraints*
$P(ska, v_a, \mathsf{pk}(skc))$	$\emptyset$
$\xrightarrow{\nu x_1.out(x_1)}_s$	
$\nu r.(\mathsf{in}(x).\mathit{if}\ \mathsf{getmess}_{\mathsf{pk}(skc)}(x) = \mathsf{blind}_r(v_a)$ $\mathit{then}\ \mathsf{out}(\mathsf{unblind}_r(x)) \mid \{^{M_1}/_{x_1}\})$	$\emptyset$
$\xrightarrow{in(y)}_s$ $\nu r.(\mathit{if}\ \mathsf{getmess}_{\mathsf{pk}(skc)}(y) = \mathsf{blind}_r(v_a)$ $\mathit{then}\ \mathsf{out}(\mathsf{unblind}_r(y)) \mid \{^{M_1}/_{x_1}\})$	$\nu r.\{^{M_1}/_{x_1}\} \Vdash y$
$\rightarrow_s$ $\nu r.(\mathsf{out}(\mathsf{unblind}_r(y)) \mid \{^{M_1}/_{x_1}\})$	$\nu r.\{^{M_1}/_{x_1}\} \Vdash y$ $\mathsf{getmess}_{\mathsf{pk}(skc)}(y) = \mathsf{blind}_r(v_a)$
$\xrightarrow{\nu x_2.out(x_2)}_s$ $\nu r.(\{^{M_1}/_{x_1}\} \mid \{^{M_2}/_{x_2}\})$	$\nu r.\nu x_2.\{^{M_1}/_{x_1}\} \Vdash y$ $\mathsf{getmess}_{\mathsf{pk}(skc)}(y) = \mathsf{blind}_r(v_a)$

where $M_1 = \mathsf{sign}_{ska}(\mathsf{blind}_r(v_a))$ and $M_2 = \mathsf{unblind}_r(y)$. Note that to derive the first step, we use the rule Out-T_s *and* Scope_s *to add the restriction $\nu r.$ in front of the process. To derive the other steps, for instance $\nu x_2.out(x_2)$ we also use the rule* Par_s *to put $\{^{M_1}/_{x_1}\}$ in parallel. About the naming environment, we have assumed among others that x_1, x_2 and y are marked as new (namely* n*) at the beginning. At the end, the variables x_1, x_2 are marked as* f *whereas y is marked as* c*.*

We can now consider some example evolutions for the two processes in the equivalence (1) we want to establish. One of the expected evolutions of the left hand side is as follows. The process $P(ska, v_a, \mathsf{pk}(skc))$ will first do an action directly followed by the corresponding action of the process $P(skb, v_b, \mathsf{pk}(skc))$. The $\downarrow$ operator allows us to put the restrictions in front of the process to have an intermediate process.

$(\nu ska, skb.(\mathsf{out}(\mathsf{pk}(ska)).\mathsf{out}(\mathsf{pk}(skb)).$
$(P(ska, v_a, \mathsf{pk}(skc)) \mid P(skb, v_b, \mathsf{pk}(skc))))\downarrow \ ; \ \emptyset \ ; \ \mathsf{N}_s)$

$\xrightarrow{\nu x_0.out(x_0)}_s \xrightarrow{\nu x'_0.out(x'_0)}_s$ (* outputs of the public keys *)

$\xrightarrow{\nu x_1.out(x_1)}_s \xrightarrow{\nu x'_1.out(x'_1)}_s$ (* outputs of the first message *)

$\xrightarrow{in(y)}_s \xrightarrow{in(y')}_s$ (* inputs of the second message *)

$\rightarrow_s \rightarrow_s$ (* conditional - then branch *)

$\xrightarrow{\nu x_2.out(x_2)}_s \xrightarrow{\nu x'_2.out(x'_2)}_s \quad (\varphi \ ; \mathcal{C} \ ; \mathsf{N}'_s)$

where $\varphi = \nu ska, skb, ra, rb.(\{^{\mathsf{pk}(ska)}/_{x_0}\} \mid \{^{\mathsf{pk}(skb)}/_{x'_0}\} \mid \{^{\mathsf{sign}_{ska}(\mathsf{blind}_{ra}(v_a))}/_{x_1}\} \mid \{^{\mathsf{sign}_{skb}(\mathsf{blind}_{rb}(v_b))}/_{x'_1}\} \mid \{^{\mathsf{unblind}_{ra}(y)}/_{x_2}\} \mid \{^{\mathsf{unblind}_{rb}(y')}/_{x'_2}\})$.
Among the constraints in $\mathcal{C}$, *we have the following deducibility and equality constraints.*

$$\nu x_2, x'_2.\varphi \Vdash y \qquad \mathsf{getmess}_{\mathsf{pk}(skc)}(y) = \mathsf{blind}_{ra}(v_a)$$
$$\nu x_2.x'_2.\varphi \Vdash y' \qquad \mathsf{getmess}_{\mathsf{pk}(skc)}(y') = \mathsf{blind}_{rb}(v_b)$$

The right hand side of equivalence (1) can evolve in a similar way, i.e. with the same labels, to the symbolic process $(\varphi' \ ; \mathcal{C}' \ ; \mathsf{N}'_s)$ *where*

- $\varphi' = \nu ska, skb, ra, rb.(\{^{\mathsf{pk}(ska)}/_{x_0}\} \mid \{^{\mathsf{pk}(skb)}/_{x'_0}\} \mid \{^{\mathsf{sign}_{ska}(\mathsf{blind}_{ra}(v_b))}/_{x_1}\} \mid \{^{\mathsf{sign}_{skb}(\mathsf{blind}_{rb}(v_a))}/_{x'_1}\} \mid \{^{\mathsf{unblind}_{rb}(y')}/_{x_2}\} \mid \{^{\mathsf{unblind}_{ra}(y)}/_{x'_2}\})$.
- *the system* $\mathcal{C}'$ *contains* $\nu x_2, x'_2.\varphi' \Vdash y, \quad \mathsf{getmess}_{\mathsf{pk}(skc)}(y) = \mathsf{blind}_{ra}(v_b)$
  $\nu x_2, x'_2.\varphi' \Vdash y', \ \mathsf{getmess}_{\mathsf{pk}(skc)}(y') = \mathsf{blind}_{rb}(v_a)$.

To fully show that the two processes are in symbolic bisimulation, we would have to consider other possible evolutions as well. We omit that here. To complete the picture for this path, we illustrate the calculations to show static equivalence for the final processes along the paths. Consider the extended constraint systems $\tilde{\mathcal{C}}$ *and* $\tilde{\mathcal{C}}'$ *as defined in Definition 9.1.*

- $\tilde{\mathcal{C}} = \mathcal{C} \cup \{\varphi \Vdash z_1 \ ; \varphi \Vdash z_2 \ ; z_1 = z_2\}$, *and*
- $\tilde{\mathcal{C}}' = \mathcal{C}' \cup \{\varphi' \Vdash z_1 \ ; \varphi' \Vdash z_2 \ ; z_1 = z_2\}$.

Let $\tilde{\mathsf{N}}_s = \mathsf{N}'_s[z_1, z_2 \mapsto \mathsf{f}]$ *where* z_1, z_2 *are constraint variables that are marked as* n *in* N'_s. *We have to establish that* $Sol^{\mathsf{cl}}_{\mathsf{E}}(\tilde{\mathcal{C}} \ ; \tilde{\mathsf{N}}_s) = Sol^{\mathsf{cl}}_{\mathsf{E}}(\tilde{\mathcal{C}}' \ ; \tilde{\mathsf{N}}_s)$. *We don't prove this fully, but just illustrate with an expected solution. A solution is a map* θ *from the constraint variables* $\{y, y', z_1, z_2\}$ *to terms not including the constraint variables or the restricted names* ska, skb, ra, rb, *and in the case of* y, y', *not including* x_2, x'_2. *Consider for instance the solution*

$$\theta := \left\{ \begin{array}{lcllcl} y & \mapsto & \mathsf{sign}_{skc}(\mathsf{getmess}_{x_0}(x_1)) & z_1 & \mapsto & x_2 \\ y' & \mapsto & \mathsf{sign}_{skc}(\mathsf{getmess}_{x'_0}(x'_1)) & z_2 & \mapsto & v_a \end{array} \right.$$

We have that $\theta \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\tilde{\mathcal{C}} \,;\, \tilde{\mathsf{N}}_s)$ *and also* $\theta \in Sol^{\mathsf{cl}}_{\mathsf{E}}(\tilde{\mathcal{C}}' \,;\, \tilde{\mathsf{N}}_s)$.

ProVerif can handle the equational theory of this example, but it is not able to prove the privacy property. The reason is that ProVerif is not able to construct the bisimulation that is required. In the first phase, the behaviour of voter A must be matched with the behaviour of voter A, and B's behaviour with B's behaviour, so that the signatures respect the static equivalence; while in the second phase, A's behaviour must be matched with B's behaviour, and B's behaviour with A's behaviour, so that the votes output respect the static equivalence. (However, our algorithms are not yet implemented!)

11 Related and Future Work

Pioneering work in symbolic bisimulations has been done by Henessy and Lin [18] for value-passing CCS. However, the result which is most closely related to ours is by Borgström *et al.* [10]: they define a symbolic bisimulation for the spi calculus with the same sources of incompleteness as we have. However, our treatment of general equational theories is non trivial as illustrated by the problems implied for structural equivalence.

For many important equational theories, static equivalence has been shown to be decidable in [1]. More interestingly, some work has also been done to automate observational equivalence. The ProVerif tool [7] automates observational equivalence checking for the applied pi calculus (with process replication), but since the problem is undecidable the technique it uses is necessarily incomplete. The tool aims at proving a finer equivalence relation and relies on easily matching up the execution paths of the two processes [8]. In his thesis, Baudet [6] presents a decision procedure for a similar equivalence, called diff-equivalence, in a simplified process calculus. Examples where this equivalence relation is too fine occur when proving the observational equivalence required to show vote-privacy [19, 12]. Although our symbolic bisimulation is not complete, we are able to conclude on examples where ProVerif fails (see Section 10).

The technique used in ProVerif will generally fail in the case where the two processes take different branches at some point. This is the case in Example 10.3: after a synchronisation (modelled by a communication on the private channel c_1) between the two parallel components of process P (resp. Q), the output action of the left component of P matches the output action of the right component of Q.

Concerning future work, the obvious next step is to study the equivalence of solutions for constraint systems under different equational theories. Promising results have already been shown in [5] for a significant class of equational theories but for constraint systems that do not have disequalities. These results readily apply for deciding our symbolic bisimulation on the fragment without else branches in conditionals. We plan to implement an automated tool for checking observational equivalence. In particular we aim at automating proofs

arising in case studies of electronic voting protocols which currently rely on hand proofs [12].

Another direction for future work is how to include process replication (the "!" operator), which is omitted entirely from this paper. Since we require to put the ν operator in outermost position in intermediate extended processes, one could first try to include replications that do not have ν in their scope. This corresponds to processes that may not terminate, but can only create finitely many names. Including replication having ν in its scope is certainly more challenging.

Acknowledgements. We would like to thank Martín Abadi, Cédric Fournet, Magnus Johansson and Bjorn Victor for interesting discussions. We also warmly thank the anonymous reviewers for their many detailed comments; they helped us significantly to improve the paper. Thanks also to Liu Jia, who asked us several questions that were instrumental in helping us prepare the final version.

References

[1] M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 387(1-2):2–32, 2006.

[2] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th Symposium on Principles of Programming Languages*, pages 104–115. ACM Press, 2001.

[3] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. 4th Conference on Computer and Communications Security*, pages 36–47. ACM, 1997.

[4] R. Amadio, D. Lugiez, and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290:695–740, 2002.

[5] M. Baudet. Deciding security of protocols against off-line guessing attacks. In *Proc. 12th Conference on Computer and Communications Security*, pages 16–25. ACM Press, 2005.

[6] M. Baudet. *Sécurité des protocoles cryptographiques : aspects logiques et calculatoires*. Thèse de doctorat, LSV, ENS Cachan, France, Jan. 2007.

[7] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proc. 14th Computer Security Foundations Workshop*, pages 82–96. IEEE Comp. Soc. Press, 2001.

[8] B. Blanchet, M. Abadi, and C. Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *Proc. 20th Symposium on Logic in Computer Science*, pages 331–340. IEEE Comp. Soc. Press, 2005.

[9] M. Boreale and R. D. Nicola. A symbolic semantics for the pi-calculus. *Information and Computation*, 126(1):34–52, 1996.

[10] J. Borgström, S. Briais, and U. Nestmann. Symbolic bisimulation in the spi calculus. In *Proc. 15th Int. Conference on Concurrency Theory*, volume 3170 of *LNCS*, pages 161–176. Springer, 2004.

[11] S. Delaune and F. Jacquemard. A decision procedure for the verification of security protocols with explicit destructors. In *Proc. 11th ACM Conference on Computer and Communications Security (CCS'04)*, pages 278–287. ACM Press, 2004.

[12] S. Delaune, S. Kremer, and M. D. Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *Proc. 19th Computer Security Foundations Workshop*, pages 28–39. IEEE Comp. Soc. Press, 2006.

[13] S. Delaune, S. Kremer, and M. D. Ryan. Symbolic bisimulation for the applied pi calculus. In *Preliminary Proc. 5th International Workshop on Security Issues in Concurrency (SecCo'07)*, 2007.

[14] S. Delaune, S. Kremer, and M. D. Ryan. Symbolic bisimulation for the applied pi-calculus. In *Proc. 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, volume 4855 of *Lecture Notes in Computer Science*, pages 133–145. Springer, 2007.

[15] S. Delaune, S. Kremer, and M. D. Ryan. Symbolic bisimulation for the applied pi calculus. Research Report LSV-08-32, Laboratoire Spécification et Vérification, ENS Cachan, France, 2008. 73 pages.

[16] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, 1983.

[17] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *Advances in Cryptology – AUSCRYPT '92*, volume 718 of *Lecture Notes in Computer Science*, pages 244–251. Springer, 1992.

[18] M. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138(2):353–389, 1995.

[19] S. Kremer and M. D. Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In *Proc. 14th European Symposium on Programming*, volume 3444 of *LNCS*, pages 186–200. Springer, 2005.

[20] J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. 8th Conference on Computer and Communications Security*, pages 166–175, 2001.

[21] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I. *Information and Computation*, 100(1):1–40, 1992.

A Proofs of Part I – Intermediate Calculus

A.1 Soundness Results

Proposition A.1 (soundness of $\equiv_i$) *Let $(A_i \,;\, \mathsf{N})$ and $(B_i \,;\, \mathsf{N})$ be two intermediate processes such that $(A_i \,;\, \mathsf{N}) \equiv_i (B_i \,;\, \mathsf{N})$. Then we have that $A_i \equiv B_i$.*

Proof. This proof is straightforward and can be done by induction on the proof tree witnessing $(A_i \,;\, \mathsf{N}) \equiv_i (B_i \,;\, \mathsf{N})$. The only point where we have to pay attention is closure by application of intermediate evaluation context. However, to deal with this case, it is sufficient to note that any intermediate evaluation context w.r.t. an intermediate extended process A is also an evaluation context w.r.t. A. □

Proposition A.2 (soundness of $\rightarrow_i$) *Let $(A_i \,;\, \mathsf{N})$ and $(B_i \,;\, \mathsf{N})$ be two intermediate processes such that $(A_i \,;\, \mathsf{N}) \rightarrow_i (B_i \,;\, \mathsf{N})$. Then we have that $A_i \rightarrow B_i$.*

Proof. This proof is straightforward and can be done by induction on the proof tree witnessing $(A_i \,;\, \mathsf{N}) \rightarrow_i (B_i \,;\, \mathsf{N})$. The base case, *i.e.* $\textsc{Comm}_i$, $\textsc{Then}_i$ and $\textsc{Else}_i$ are obvious since corresponding rules exist in the initial semantics. To deal with closure by application of evaluation context, it is sufficient to note that any intermediate evaluation context w.r.t. an intermediate extended process A is also an evaluation context w.r.t. A. Lastly, closure by structural equivalence can be easily done thanks to Proposition A.1. □

Proposition A.3 (soundness of $\xrightarrow{\alpha}_i$) *Let $(A_i \,;\, \mathsf{N})$ and $(B_i \,;\, \mathsf{N}')$ be two intermediate processes such that $(A_i \,;\, \mathsf{N}) \xrightarrow{\alpha}_i (B_i \,;\, \mathsf{N}')$. Then we have that $A_i \xrightarrow{\alpha} B_i$.*

Proof. To prove this result, we distinguish two cases depending on the fact whether α is an output or an input. In both cases, we perform the proof by induction on the prooftree witnessing the fact that $(A_i \,;\, \mathsf{N}) \xrightarrow{\alpha}_i (B_i \,;\, \mathsf{N}')$. However, if α is an input, we need to show an intermediate result in order to be able to deal with the inductive case $\textsc{Par}_i$. Note that in the case where α is a label of the form $\mathsf{out}(a, c)$, $\nu d.\mathsf{out}(a, d)$ or $\nu x.\mathsf{out}(a, x)$, we do not have such a problem since the rule $\textsc{Par}_i$ is equivalent to the rule described below. This is due to the fact that $fv(\alpha) = \emptyset$.

$$\textsc{Par-Out}_i \quad \frac{(A \,;\, \mathsf{N}) \xrightarrow{\alpha}_i (A', \mathsf{N}')}{(A \mid B \,;\, \mathsf{N}) \xrightarrow{\alpha}_i (A' \mid B, \mathsf{N}')}$$

First case: *α is of the form $\mathsf{out}(a, c)$, $\nu d.\mathsf{out}(a, d)$ or $\nu x.\mathsf{out}(a, x)$.* The two base cases, $\textsc{Out-Ch}_i$ and $\textsc{Out-T}_i$, are trivial. We only need to pay attention that the side condition of Out-T is satisfied. This is due to the fact that $\mathsf{N}(x) = \mathsf{n}$ whereas $\mathsf{N}(fv(P) \cup fv(M)) = \mathsf{f}$. Hence, we have that $x \notin fv(P) \cup fv(M)$. Now, we have to deal with the inductive cases.

Case OPEN-CH$_i$. In such a case, we have that the prooftree witnessing the fact that $(A_i \ ; \mathsf{N}) \xrightarrow{\alpha}_i (B_i \ ; \mathsf{N}')$ ends with the following inference rule.

$$\frac{(A' \ ; \mathsf{N}'') \xrightarrow{out(a,c)}_i (B' \ ; \mathsf{N}''') \qquad c \neq a, \mathsf{N}''(d) = \mathsf{n}}{(\nu c.A', \mathsf{N}) \xrightarrow{\nu d.out(a,d)}_i (B'\{^d/_c\}, \mathsf{N}')}$$

By induction hypothesis, we know that $A' \xrightarrow{out(a,c)} B'$ and we deduce that $A'\{^d/_c\} \xrightarrow{out(a,d)} B'\{^d/_c\}$ since A' and B' are nv-distinct and d is fresh (it does not appear in A' nor in B'). By application of the rule OPEN-CH, we obtain $\nu d.A'\{^d/_c\} \xrightarrow{\nu d.out(a,d)} B'\{^d/_c\}$. Since $\nu d.A'\{^d/_c\} \equiv \nu c.A'$, we deduce that $\nu c.A' \xrightarrow{\nu d.out(a,d)} B'\{^d/_c\}$, i.e. exactly what we want.

Case SCOPE$_i$. This case is completely straightforward.

Case PAR-OUT$_i$. The proof tree ends with the following inference rule

$$\textsc{Par-Out}_i \quad \frac{(A' \ ; \mathsf{N}) \xrightarrow{\alpha}_i (B', \mathsf{N}')}{(A' \mid D \ ; \mathsf{N}) \xrightarrow{\alpha}_i (B' \mid D, \mathsf{N}')}$$

In such a case, we need to pay attention that the side condition of the rule PAR is satisfied. Since $\mathsf{N}(bn(\alpha) \cup bv(\alpha)) = \mathsf{n}$ and $\mathsf{N}(fn(D) \cup fv(D)) = \mathsf{f}$, we deduce that $bn(\alpha) \cap fn(D) = bv(\alpha) \cap fv(D) = \emptyset$.

Case STRUCT$_i$. We easily conclude for this case by using Proposition A.1.

Second case: *α is of the form* $\mathsf{in}(a, M)$. To deal with this case, we rely on the claim stated below.

> **Claim:** Let $(A_i \ ; \mathsf{N})$ and $(B_i \ ; \mathsf{N}')$ be two extended processes such that A_i and B_i are intermediate framed processes and $(A_i \ ; \mathsf{N}) \xrightarrow{in(a,M)}_i (B_i \ ; \mathsf{N}')$. Let D_i be an intermediate framed process such that $(A_i \mid D_i \ ; \mathsf{N})$ and $(B_i \mid D_i \ ; \mathsf{N}')$ are also intermediate processes. Then, for any term M' such that $M = M'\psi(D_i)$, we have that $A_i \mid D_i \xrightarrow{in(a,M')} B_i \mid D_i$.

Note that this result will allow us to conclude. Our claim allows us to deal with the case where A_i and B_i are intermediate framed processes. For this it is sufficient to apply the claim above with $D_i = 0$ and $M' = M$. Then it remains to notice that $A_i \mid 0 \equiv A_i$ and $B_i \mid 0 \equiv B_i$ in order to conclude. Now, let us consider the case where $(A_i \ ; \mathsf{N})$, $(B_i \ ; \mathsf{N}')$ are not intermediate framed processes. We show the result by induction on the prooftree witnessing $(A_i \ ; \mathsf{N}) \xrightarrow{in(a,M)}_i (B_i \ ; \mathsf{N}')$. In such a case, this prooftree ends either with an instance of SCOPE$_i$ or an instance of STRUCT$_i$. In both cases, we easily conclude by using the induction hypothesis and applying the corresponding rules, that is either SCOPE or STRUCT and using Proposition A.1.

It remains to establish the claim.

Proof of the claim. We show this result by induction on the prooftree witnessing $(A_i \ ; \mathsf{N}) \xrightarrow{in(a,M)}_i (B_i \ ; \mathsf{N}')$. First, we consider the base case, i.e. the rule IN_i. In such a case, we have that $A_i = \mathsf{in}(a,x).P$, $B_i = P\{^M/_x\}$ and $\mathsf{N}' = \mathsf{N}$. We have that

$$\dfrac{\dfrac{\mathsf{in}(a,x).P \xrightarrow{in(a,M')} P\{^{M'}/_x\}}{\mathsf{in}(a,x).P \mid D_i \xrightarrow{in(a,M')} P\{^{M'}/_x\} \mid D_i \ \equiv P\{^M/_x\} \mid D_i}}{A_i \mid D_i \xrightarrow{in(a,M')} B_i \mid D_i}$$

Now, we have to deal with the inductive cases, that is STRUCT_i and PAR_i since the other rules do not allow us to derive framed processes. For STRUCT_i the result can be easily obtained by applying the induction hypothesis and Proposition A.1. Hence, we focus on PAR_i. In such a case, we have that the proof tree witnessing $(A_i \ ; \mathsf{N}) \xrightarrow{in(a,M)}_i (B_i \ ; \mathsf{N}')$ ends with the following rule:

$$\dfrac{(A_i' \ ; \mathsf{N}) \xrightarrow{in(a,M\psi(D))}_i (B_i' \ ; \mathsf{N}')}{(A_i' \mid D \ ; \mathsf{N}) \xrightarrow{in(a,M)}_i (B_i' \mid D \ ; \mathsf{N}')}$$

Recall that D_i is an intermediate framed process such that $(A_i' \mid D \mid D_i \ ; \mathsf{N})$ and $(B_i' \mid D \mid D_i \ ; \mathsf{N}')$ are also intermediate framed processes. Let M' be a term such that $M = M'\psi(D_i)$. By induction hypothesis and since $M\psi(D) = M'\psi(D|D_i)$, we have that $A_i' \mid (D_i \mid D) \xrightarrow{in(a,M')} B_i' \mid (D_i \mid D)$ and we easily conclude by using the fact that $\xrightarrow{\alpha}$ is closed by structural equivalence. □

A.2 Completeness Results

Given a nv-distinct extended process A containing an active substitution $\{^M/_x\}$. The process $A_{\setminus x}$ is A but with the unique occurrence of $\{^M/_x\}$ replaced by 0. This notation is extended as expected to sequences of variables. Now, we introduce a lemma which allows us to describe the process $C[A]\downarrow$ from $C\downarrow$ and $A\downarrow$.

Lemma A.4 *Let C be an evaluation context which is nv-distinct. Let $\tilde{x}$ be the tuple of variables such that the hole is in the scope of an occurrence of "νx" in C. Then there exists some sequences of names $\tilde{n}_1$, $\tilde{n}_2$ and an intermediate framed evaluation context G such that*

- *$C\downarrow = \nu\tilde{n}_1.\nu\tilde{n}_2.G$, and*
- *for all extended process A such that $C[A]$ is nv-distinct, we have that*

$$C[A]\downarrow = \nu\tilde{n}_1.\nu\tilde{m}.\nu\tilde{n}_2.G[F_{\setminus\tilde{x}}](\psi(G) \cup \psi(F))^\star$$

where $A\downarrow = \nu\tilde{m}.F$ for some sequence of names $\tilde{m}$ and some intermediate framed process F.

Proof. We prove this result by induction on the structure of C. In the base case, i.e. $C = \_$, we can show that $\tilde{n}_1 = \emptyset$, $\tilde{n}_2 = \emptyset$ and $G = \_$ satisfy the requirements. Indeed, let A be an extended process such that $A\downarrow = \nu\tilde{m}.F$, we have $\nu\tilde{n}_1.\nu\tilde{m}.\nu\tilde{n}_2.G[F_{\setminus\tilde{x}}](\psi(G) \cup \psi(F))^\star = \nu\tilde{m}.F\psi(F)^\star = \nu\tilde{m}.F = A\downarrow = C[A]\downarrow$.

The inductive cases are $C = C' \mid B$, $C = B \mid C'$, $C = \nu n.C'$ and $C = \nu x.C'$. Let $\tilde{x}'$ be the tuple of variables x' such that the hole of C' is in the scope of an occurrence of $\nu x'$ in C'. By induction hypothesis, we know that there exists some sequences of names $\tilde{n}'_1$ and $\tilde{n}'_2$ and an intermediate framed evaluation context G' such that

- $C'\downarrow = \nu\tilde{n}'_1.\nu\tilde{n}'_2.G'$, and
- for all extended process A such that $C'[A]$ is nv-distinct, we have that

$$C'[A]\downarrow = \nu\tilde{n}'_1.\nu\tilde{m}.\nu\tilde{n}'_2.G'[F_{\setminus\tilde{x}'}](\psi(G') \cup \psi(F))^\star \text{ where } A\downarrow = \nu\tilde{m}.F.$$

Inductive case 1: $C = C' \mid B$. Let $B\downarrow = \nu\tilde{b}.B'$. In such a case, we have that

$$\begin{array}{rcl} C[A]\downarrow & = & (C'[A] \mid B)\downarrow \\ & = & \nu\tilde{n}'_1.\nu\tilde{m}.\nu\tilde{n}'_2.\nu\tilde{b}.(G'[F_{\setminus\tilde{x}'}](\psi(G') \cup \psi(F))^\star \mid B')(\psi(C'[A]) \cup \psi(B'))^\star \end{array}$$

Let $\tilde{n}_1 = \tilde{n}'_1, \tilde{n}_2 = \tilde{n}'_2, \tilde{b}$ and $G = G' \mid B'$. As $\tilde{x} = \tilde{x}'$ and $\psi(G') \cup \psi(F) = \psi(C'[A])$ we obtain the expected result.

Inductive case 2: $C = B \mid C'$. This case is similar to the previous one.

Inductive case 3: $C = \nu n.C'$. In such a case, we have that

$$\begin{array}{rcl} C[A]\downarrow & = & \nu n.(C'[A]\downarrow) \\ & = & \nu n.\nu\tilde{n}'_1.\nu\tilde{m}.\nu\tilde{n}'_2.G'[F_{\setminus\tilde{x}'}](\psi(G') \cup \psi(F))^\star \end{array}$$

Let $\tilde{n}_1 = n, \tilde{n}'_1$, $\tilde{n}_2 = \tilde{n}'_2$ and $G' = G$. We obtain the expected result.

Inductive case 4: $C = \nu x.C'$. In such a case, we have that

$$\begin{array}{rcl} C[A]\downarrow & = & (C'[A]\downarrow)_{\setminus x} \\ & = & (\nu\tilde{n}'_1.\nu\tilde{m}.\nu\tilde{n}'_2.G'[F_{\setminus\tilde{x}'}](\psi(G') \cup \psi(F))^\star)_{\setminus x} \end{array}$$

Let $\tilde{n}_1 = \tilde{n}'_1$, $\tilde{n}_2 = \tilde{n}'_2$ and $G' = G$. We obtain the expected result since we have that $\tilde{x} = \tilde{x}', x$. □

The following lemma relies on the notion of linear proof defined below. A proof in linear form of $A \equiv B$ is a sequence $A = A_1, \ldots, A_n = B$ such that for every $1 \leq j \leq n$, there exist an evaluation context C_j, two extended processes A'_j and A'_{j+1} such that:

- $A_j = C_j[A'_j]$, $A_{j+1} = C_j[A'_{j+1}]$, and
- either $A'_j \equiv A'_{j+1}$ (or $A'_{j+1} \equiv A'_j$) is an instance of PAR-0, PAR-A, PAR-C, NEW-0, NEW-C, NEW-PAR, ALIAS, SUBST, or REWRITE,

- or $A'_j =_\alpha A'_{j+1}$.

Similarly, a proof in linear form of $A \to^* B$ is a sequence $A = A_1, \ldots, A_n = B$ such that for every $1 \leq j \leq n$, there exist an evaluation context C_j, two extended processes A'_j and A'_{j+1} such that:

- $A_j = C_j[A'_j]$, $A_{j+1} = C_j[A'_{j+1}]$, and
- either $A'_j \equiv A'_{j+1}$ (or $A'_{j+1} \equiv A'_j$) is an instance of PAR-0, PAR-A, PAR-C, NEW-0, NEW-C, NEW-PAR, ALIAS, SUBST, or REWRITE,
- or $A'_j =_\alpha A'_{j+1}$,
- or $A'_j \to A'_{j+1}$ is an instance of COMM, THEN or ELSE.

Moreover, there must exist at least one j such that $A'_j \to A'_{j+1}$ is an instance of COMM, THEN or ELSE.

Lemma A.5 *Let A and B be two nv-distinct extended processes such that $A \bowtie B$ with $\bowtie \in \{\equiv, \to^*\}$ and N be a naming environment compatible with A and B. Then there exists a proof in linear form such that every process in the proof is nv-distinct and compatible with N.*

Proposition A.6 (completeness of $\equiv_i$) *Let A and B be two nv-distinct extended processes such that $A \equiv B$ and N be a naming environment compatible with $A\downarrow$ and $B\downarrow$. Then there exists an intermediate process $(D_i \ ; \mathsf{N})$ such that $(A\downarrow \ ; \mathsf{N}) \equiv_i (D_i \ ; \mathsf{N}) \approx (B\downarrow \ ; \mathsf{N})$.*

Proof. Let A and B be two nv-distinct extended processes such that $A \equiv B$. We consider the proof of structural equivalence in linear form. Thanks to Lemma A.5, we can assume that extended processes involved in this derivation are nv-distinct and compatible with N. We show the result by induction on the length ℓ of the derivation. We first show the result when $\ell = 1$ by considering each rule of structural equivalence in turn. Then, we show the inductive case, i.e. $\ell > 1$. We denote by C the evaluation context under which the structural equivalence rule is applied. We denote by $\tilde{n}_1$, $\tilde{n}_2$, (resp. $\tilde{x}$) and G the sequences of names (resp. variables) and the intermediate framed evaluation context which satisfy the condition stated in Lemma A.4.

Case PAR-0: $C[D] \equiv C[D \mid 0]$.
Let $D_i = B\downarrow$. Clearly, we have that $(D_i \ ; \mathsf{N}) \approx (B\downarrow \ ; \mathsf{N})$. Now, let $\tilde{m}$ be the sequence of names and F be the framed process such that $D\downarrow = \nu\tilde{m}.F$. We have that $(D \mid 0)\downarrow = \nu\tilde{m}.(F \mid 0)$ and thanks to Lemma A.4 we have

- $(A\downarrow \ ; \mathsf{N}) = (C[D]\downarrow \ ; \mathsf{N}) = (\nu\tilde{n}_1.\nu\tilde{m}.\nu\tilde{n}_2.G[F_{\setminus\tilde{x}}](\psi(G) \cup \psi(F))^\star \ ; \mathsf{N})$, and
- $(B\downarrow \ ; \mathsf{N}) = (C[D \mid 0]\downarrow \ ; \mathsf{N}) = (\nu\tilde{n}_1.\nu\tilde{m}.\nu\tilde{n}_2.G[(F \mid 0)_{\setminus\tilde{x}}](\psi(G) \cup \psi(F \mid 0))^\star \ ; \mathsf{N})$.

Hence, we have that $(A\downarrow\ ;\ \mathsf{N}) \equiv_i (B\downarrow\ ;\ \mathsf{N})$.

A similar reasoning allows us to conclude for Par-A, Par-C. For the rule New-C, if the commutation involves two names, we conclude as in the previous case since this rule has a counterpart in the intermediate semantics. Otherwise, we have that $A\downarrow = B\downarrow$ and we easily conclude.

The rule New-0, New-Par, Subst, Alias and Rewrite are also straightforward. Note also that if $A \equiv B$ is a renaming step, then $D_i = A\downarrow$ satisfies the requirement. This concludes the base cases.

Now, it remains to show the inductive case. Let A and A' be two extended processes such that $A \equiv A'$ by a derivation of length $\ell > 1$. Then there exists B such that $A \equiv B$ by a derivation of length 1 and $B \equiv A'$ by a derivation of length $\ell' < \ell$. Firstly, we know that there exists an intermediate extended process $(D_i\ ;\ \mathsf{N})$ such that $(A\downarrow\ ;\ \mathsf{N}) \equiv_i (D_i\ ;\ \mathsf{N})$ and $(D_i\ ;\ \mathsf{N}) \approx (B\downarrow\ ;\ \mathsf{N})$. By using our induction hypothesis, we also know that there exists an intermediate extended process $(D_i'\ ;\ \mathsf{N})$ such that $(B\downarrow\ ;\ \mathsf{N}) \equiv_i (D_i'\ ;\ \mathsf{N})$ and $(D_i'\ ;\ \mathsf{N}) \approx (A'\downarrow\ ;\ \mathsf{N})$. Hence by using Lemma 4.5, we deduce that there exists an intermediate extended process $(D_i''\ ;\ \mathsf{N})$ such that $(D_i\ ;\ \mathsf{N}) \equiv_i (D_i''\ ;\ \mathsf{N})$ and $(D_i''\ ;\ \mathsf{N}) \approx (D_i'\ ;\ \mathsf{N})$. Hence, the process $(D_i''\ ;\ \mathsf{N})$ satisfies the requirements. □

Proposition A.7 (completeness of $\rightarrow_i$) *Let A and B be two nv-distinct extended processes such that $A \rightarrow^* B$ (resp. $A \rightarrow B$) and N be a naming environment compatible with $A\downarrow$ and $B\downarrow$. Then there exists an extended process $(D_i\ ;\ \mathsf{N})$ such that:*

- $(A\downarrow\ ;\ \mathsf{N}) \rightarrow_i^* (D_i\ ;\ \mathsf{N})$ *(resp. $(A\downarrow\ ;\ \mathsf{N}) \rightarrow_i (D_i\ ;\ \mathsf{N})$) and,*
- $(D_i\ ;\ \mathsf{N}) \approx (B\downarrow\ ;\ \mathsf{N})$.

Proof. Let A and B be two nv-distinct extended processes such that $A \rightarrow^* B$. The case where $B = A$ (reflexivity) is trivial. Otherwise we consider the proof of $A \rightarrow^* B$ in linear form. Each step of this proof will be either a single reduction step or a sequence of steps of structural equivalence. Thanks to Lemma A.5, we can assume that extended processes involved in this derivation are nv-distinct and compatible with N. We show the result by induction on the length ℓ of the derivation. We first show the result when $\ell = 1$. In the case of structural equivalence, Proposition A.6 allows us to conclude. Hence, we only consider the three rules of internal reduction in turn. Then, we show the inductive case, i.e. $\ell > 1$. We denote by C the evaluation context under which the rule is applied. We denote by $\tilde{n}_1$, $\tilde{n}_2$, (resp. $\tilde{x}$) and G the sequences of names (resp. variables) and the intermediate framed evaluation context which satisfy the condition stated in Lemma A.4.

Case Comm: $\mathsf{out}(a, M).P \mid \mathsf{in}(a, x).Q \rightarrow P \mid Q\{^M/_x\}$. We have that $A = C[\mathsf{out}(a, M).P \mid \mathsf{in}(a, x).Q]$ and $B = C[P \mid Q\{^M/_x\}]$. Let $P\downarrow = \nu\tilde{n}_p.F_p$ and $Q\downarrow = \nu\tilde{n}_q.F_q$. By using Lemma A.4, we obtain

- $(A\downarrow\ ;\ \mathsf{N}) = (\nu\tilde{n}_1.\nu\tilde{n}_p.\nu\tilde{n}_q.\nu\tilde{n}_2.G[\mathsf{out}(a, M).F_p \mid \mathsf{in}(a, x).F_q]\psi(G)^\star\ ;\ \mathsf{N})$,

- $(B\downarrow\,;\mathsf{N}) = (\nu\tilde{n}_1.\nu\tilde{n}_p.\nu\tilde{n}_q.\nu\tilde{n}_2.G[F_p \mid F_q\{^M/_x\}]\psi(G)^\star\,;\mathsf{N})$.

Note that F_p and F_q are intermediate plain processes (as P, resp. Q, are prefixed by an input, resp. output) and hence $\psi(F_p)$ and $\psi(F_q\{^M/_x\})$ are empty. Let $D_i = B\downarrow$. It is easy to see that D_i satisfies the requirements.

We deal with the rules Then and Else in a similar way.

Now, it remains to show the inductive case. Let A and A' be two extended processes such that $A \rightarrow^* A'$ by a derivation of length $\ell > 1$. Then there exists B such that $A \equiv B$ (or $A \rightarrow B$) and $B \rightarrow^* A'$ by a derivation of length $\ell' < \ell$. In both case we conclude thanks to Lemma 4.5 and the induction hypothesis on $B \rightarrow^* A'$. □

Lemma A.8 *Let A and B be two nv-distinct extended processes such that $A \xrightarrow{\alpha} B$ and N be a naming environment compatible with A and α. Let N' be a naming environment compatible with B and such that:*

- $\mathsf{N}' = \mathsf{N}[x \mapsto \mathsf{f}]$ *when α is of the form $\nu x.out(a,x)$;*
- $\mathsf{N}' = \mathsf{N}[d \mapsto \mathsf{f}]$ *when α if of the form $\nu d.out(a,d)$;*
- $\mathsf{N}' = \mathsf{N}$ *otherwise.*

Then there exist two nv-distinct extended processes A' and B' such that:

- $A \equiv A' \xrightarrow{\alpha} B' \equiv B$ *(where $A' \xrightarrow{\alpha} B'$ does not rely on some structural equivalence steps); and*
- N *is compatible with A', and N' is compatible with B'.*

Proposition A.9 (completeness of $\xrightarrow{\alpha}_i$) *Let A and B be two nv-distinct extended processes such that $A \xrightarrow{\alpha} B$ and N be a naming environment compatible with $A\downarrow$ and α. Let N' be a naming environment compatible with $B\downarrow$ such that:*

- $\mathsf{N}' = \mathsf{N}[x \mapsto \mathsf{f}]$ *when α is of the form $\nu x.out(a,x)$;*
- $\mathsf{N}' = \mathsf{N}[d \mapsto \mathsf{f}]$ *when α if of the form $\nu d.out(a,d)$;*
- $\mathsf{N}' = \mathsf{N}$ *otherwise.*

Then there exists an intermediate process $(D_i\,;\mathsf{N}')$ such that

$$(A\downarrow\,;\mathsf{N}) \xrightarrow{\alpha}_i (D_i\,;\mathsf{N}') \approxeq (B\downarrow\,;\mathsf{N}').$$

Proof. Thanks to Lemma A.8, we can assume that A and B are two nv-distinct extended processes such that $A \xrightarrow{\alpha} B$ without involving any structural equivalence step. Otherwise, we will have that $A \equiv A' \xrightarrow{\alpha} B' \equiv B$ and we can easily conclude, thanks to Lemma 4.5, by applying the result on $A' \xrightarrow{\alpha} B'$ and by using Proposition A.6 on $A \equiv A'$ and $B \equiv B'$. We consider the different kind of labels in turn: $out(a,c)$, $\nu x.out(a,x)$, $in(a,M)$ and $\nu c.out(a,c)$.

We denote by C the evaluation context (constructed by successive applications of the rules $\textsc{Par}_i$ and $\textsc{Scope}_i$) under which the rule is applied. We denote by $\tilde{n}_1$, $\tilde{n}_2$, (resp. $\tilde{x}$) and G the sequences of names (resp. variables) and the intermediate framed evaluation context which satisfy the condition stated in Lemma A.4.

Case Out-Ch: $\mathsf{out}(a,c).P \xrightarrow{\mathsf{out}(a,c)} P$.
We have that $A = C[\mathsf{out}(a,c).P]$ and $B = C[P]$. Note that $a, c \notin bn(C[\mathsf{out}(a,c).P])$. Let $P{\downarrow} = \nu\tilde{n}_p.F_p$. By using Lemma A.4, we obtain

- $(A{\downarrow}\ ;\ \mathsf{N}) = (\nu\tilde{n}_1.\nu\tilde{n}_p.\nu\tilde{n}_2.G[\mathsf{out}(a,c).F_p]\psi(G)^\star\ ;\ \mathsf{N})$,
- $(B{\downarrow}\ ;\ \mathsf{N}') = (\nu\tilde{n}_1.\nu\tilde{n}_p.\nu\tilde{n}_2.G[F_p]\psi(G)^\star\ ;\ \mathsf{N}')$.

Let $D_i = B{\downarrow}$. Obviously, $(D_i\ ;\ \mathsf{N}') \approx (B{\downarrow}\ ;\ \mathsf{N}')$. Moreover, we see that $(\mathsf{out}(a,c).F_p)\psi(G)^\star \xrightarrow{out(a,c)}_i F_p\psi(G)^\star$. By successive applications of rules $\textsc{Par}_i$ and $\textsc{Scope}_i$ we obtain that $(A{\downarrow}\ ;\ \mathsf{N}) \xrightarrow{out(a,c)}_i (D_i\ ;\ \mathsf{N}')$.

Case Out-T: $\mathsf{out}(a,M).P \xrightarrow{\nu x.out(a,x)} P \mid \{^M/_x\}$ $\qquad x \notin fv(P) \cup fv(M)$.
We have that $A = C[\mathsf{out}(a,M).P]$ and $B = C[P \mid \{^M/_x\}]$. Let $P{\downarrow} = \nu\tilde{n}_p.F_p$. By using Lemma A.4, we obtain

- $(A{\downarrow}\ ;\ \mathsf{N}) = (\nu\tilde{n}_1.\nu\tilde{n}_p.\nu\tilde{n}_2.G[\mathsf{out}(a,M).F_p]\psi(G)^\star\ ;\ \mathsf{N})$
- $(B{\downarrow}\ ;\ \mathsf{N}') = (\nu\tilde{n}_1.\nu\tilde{n}_p.\nu\tilde{n}_2.G[F_p \mid \{^M/_x\}]\psi(G)^\star\ ;\ \mathsf{N}')$.

Note that applying the substitution $\psi(G)$ is sufficient as $\psi(F_p)$ is empty and x is a fresh variable. Let $D_i = B{\downarrow}$. Obviously, we have that $(D_i\ ;\ \mathsf{N}') \approx (B{\downarrow}\ ;\ \mathsf{N}')$. Similarly to the previous case we show that $(A{\downarrow}\ ;\ \mathsf{N}) \xrightarrow{\nu x.out(a,x)}_i (D_i\ ;\ \mathsf{N}')$.

Case In: $\mathsf{in}(a,x).P \xrightarrow{in(a,M)} P\{^M/_x\}$.
We have that $A = C[\mathsf{in}(a,x).P]$ and $B = C[P\{^M/_x\}]$. Let $P{\downarrow} = \nu\tilde{n}_p.F_p$. By using Lemma A.4, we obtain

- $(A{\downarrow}\ ;\ \mathsf{N}) = (\nu\tilde{n}_1.\nu\tilde{n}_p.\nu\tilde{n}_2.G[\mathsf{in}(a,x).F_p]\psi(G)^\star\ ;\ \mathsf{N})$
- $(B{\downarrow}\ ;\ \mathsf{N}') = (\nu\tilde{n}_1.\nu\tilde{n}_p.\nu\tilde{n}_2.G[F_p\{^M/_x\}]\psi(G)^\star\ ;\ \mathsf{N}')$.

Let $D_i = B{\downarrow}$. Obviously, we have that $(D_i\ ;\ \mathsf{N}') \approx (B{\downarrow}\ ;\ \mathsf{N}')$. Moreover, we see that (we omit the naming environment for the moment)

$$(\mathsf{in}(a,x).F_p)\psi(G)^\star \xrightarrow{in(a,M\psi(G)^\star)}_i (F_p\{^M/_x\})\psi(G)^\star.$$

By application of the rule $\textsc{Par}_i$ we obtain

$$(\mathsf{in}(a,x).F_p)\psi(G)^\star \mid G\psi(G)^\star \xrightarrow{in(a,M)}_i (F_p\{^M/_x\})\psi(G)^\star \mid G\psi(G)^\star, \text{i.e.}$$

$$G[\mathsf{in}(a,x).F_p]\psi(G)^\star \xrightarrow{in(a,M)}_i G[F_p\{^M/_x\}]\psi(G)^\star.$$

Note that $G\psi(G)^\star$ is an intermediate framed process and $\psi(G\psi(G)^\star) = \psi(G)^\star$. By successive applications of $\textsc{Scope}_i$, we obtain that $(A\downarrow\ ;\ \mathsf{N}) \xrightarrow{in(a,M)} (B\downarrow\ ;\ \mathsf{N})$. Note that since N is compatible with $A\downarrow$ and $\alpha = in(a, M)$, we have that $\tilde{n}_1$, $\tilde{n}_2$ and $\tilde{n}_p$ are marked as bound, i.e. b, whereas names that occur in M are marked as f, and thus $\tilde{n}_1$, $\tilde{n}_2$ and $\tilde{n}$ do not occur in α.

Case Open-Ch. Here, we assume that $A = \nu d.C[\mathsf{out}(a, d).P]$ and $B = C[P]$. Otherwise this can be obtained using structural equivalence. These structural equivalence steps are handled as explained above. Let $P\downarrow = \nu\tilde{n}_p.F_p$. By using Lemma A.4, we obtain

- $(A\downarrow\ ;\ \mathsf{N}) = (\nu d.\nu\tilde{n}_1.\nu\tilde{n}_p.\nu\tilde{n}_2.G[\mathsf{out}(a, d).F_p]\psi(G)^\star\ ;\ \mathsf{N})$,
- $(B\downarrow\ ;\ \mathsf{N}') = (\nu\tilde{n}_1.\nu\tilde{n}_p.\nu\tilde{n}_2.G[F_p]\psi(G)^\star\ ;\ \mathsf{N}')$.

Let $D_i = B\downarrow$. Obviously, we have that $(D_i\ ;\ \mathsf{N}') \approx (B\downarrow\ ;\ \mathsf{N}')$. As above we show that $(A\downarrow\ ;\ \mathsf{N}) \xrightarrow{\nu d.out(a,d)}_i (D_i\ ;\ \mathsf{N}')$. This allows us to conclude. □

B Proofs of Part II – Symbolic Calculus

We first show a useful lemma which allows us to transfer solutions of symbolic processes when we apply evaluation contexts to these processes.

Lemma B.1 *Let $(A\ ;\ \mathcal{C}\ ;\ \mathsf{N}_s)$ be a symbolic process and $C = \nu\tilde{u}.(\_ \mid D)$ an intermediate evaluation context such that $(C[A]\ ;\ C[\mathcal{C}]\ ;\ \mathsf{N}_s[bn(C[0]) \mapsto \mathsf{b}])$ is a symbolic process. We have that*

$$\theta \in Sol_{\mathsf{E}}(C[\mathcal{C}], \mathsf{N}_s[bn(C[0]) \mapsto \mathsf{b}])$$
$$\textit{iff}$$
$$(\theta\psi(C[0]))^\star \in Sol_{\mathsf{E}}(\mathcal{C}, \mathsf{N}_s) \textit{ and } bn(C[0]) \cap names(\mathrm{img}(\theta)) = \emptyset$$

Proof. Let $\mathsf{N}'_s = \mathsf{N}_s[bn(C[0]) \mapsto \mathsf{b}]$.

($\Rightarrow$) We need to consider two cases.

Case $C[\_] = \nu\tilde{n}.\_$.

We have that $\theta \in Sol_{\mathsf{E}}(\nu\tilde{n}.\mathcal{C}, \mathsf{N}'_s)$. Let $\mathcal{D}ed(\mathcal{C}) = \{\phi_i \Vdash x_i \mid 1 \le i \le \ell\}$ with $\phi_i = \nu\tilde{u}_i.\sigma_i$. We have that $(\theta\psi(\nu n.0))^\star = \theta^\star = \theta$ (as $vars(x_i\theta) \cap cv(\mathcal{C}) = \emptyset$ and $\mathrm{dom}(\theta) = cv(\mathcal{C})$). It is easy to check that θ is an E-solution of $\mathcal{C}$. As $\mathsf{N}'_s(names(\mathrm{img}(\theta))) = \mathsf{f}$ and $\mathsf{N}'_s(vars(\mathrm{img}(\theta))) = \mathsf{f}$ we also have that $\mathsf{N}_s(names(\mathrm{img}(\theta))) = \mathsf{f}$ and $\mathsf{N}_s(vars(\mathrm{img}(\theta))) = \mathsf{f}$. Hence, $\theta \in Sol_{\mathsf{E}}(\mathcal{C}, \mathsf{N}_s)$ and $\tilde{n} \cap names(\mathrm{img}(\theta)) = \emptyset$ (as $\mathsf{N}'_s(\tilde{n}) = \mathsf{b}$).

Case $C[\_] = \_ \mid D$.

We have that $\theta \in Sol_{\mathsf{E}}(\mathcal{C} \mid D, \mathsf{N}'_s)$. Let $\mathcal{D}ed(\mathcal{C}) = \{\phi_i \Vdash x_i \mid 1 \le i \le \ell\}$ with $\phi_i = \nu\tilde{u}_i.\sigma_i$ and let $\theta' = \theta\psi(D)$. We have to show that $\theta'^\star \in Sol_{\mathsf{E}}(\mathcal{C}, \mathsf{N}_s)$. For this, we need to show that:

- $vars(x_i\theta'^\star) \cap cv(\mathcal{C}) = \emptyset$. Actually we have that $\mathrm{dom}(\theta') = \mathrm{dom}(\theta) = cv(\mathcal{C}) = cv(\mathcal{C} \mid D) = \{x_1, \ldots, x_\ell\}$. Hence the result.

- $vars(x_i\theta'^\star) \cap (\mathrm{dom}(\phi_\ell) \setminus \mathrm{dom}(\phi_i)) = \emptyset$. By hypothesis we have that $vars(x_i\theta) \cap (\mathrm{dom}(\phi_\ell \cup \psi(D)) \setminus \mathrm{dom}(\phi_i \cup \psi(D))) = \emptyset$. We have that $\mathrm{dom}(\phi_\ell \cup \psi(D)) \setminus \mathrm{dom}(\phi_i \cup \psi(D)) = \mathrm{dom}(\phi_\ell) \setminus \mathrm{dom}(\phi_i)$ and $vars(x_i\theta'^\star) \subseteq vars(x_i\theta) \cup vars(\mathrm{img}(\psi(D)))$.

  As $A \mid D$ is applied we have that $vars(\mathrm{img}(\psi(D))) \cap \mathrm{dom}(\phi_\ell) = \emptyset$. Hence $vars(x_i\theta'^\star) \cap (\mathrm{dom}(\phi_\ell) \setminus \mathrm{dom}(\phi_i)) = \emptyset$.

- $names(x_i\theta'^\star) \cap \tilde{u}_i = \emptyset$. By definition of an intermediate process we have that $bn(0 \mid D) = \emptyset$. Hence, $\tilde{u}_i$ is a sequence of variables and this condition trivially holds.

- $vars(x_i\theta'^\star) \cap \tilde{u}_i = \emptyset$. As the process $A \mid D$ is applied we have that $vars(\mathrm{img}(\psi(D))) \cap \mathrm{dom}(A) = \emptyset$. As for all $x \in \tilde{u}_i$ we have that $x \in \mathrm{dom}(A)$ we conclude that $vars(x_i\theta'^\star) \cap \tilde{u}_i = \emptyset$.

- For any constraint $\mathsf{gd}(M) \in \mathcal{C}$ we need to show that $M(\theta'^\star\sigma_\ell)^\star$ is ground. By hypothesis we have that $M(\theta\sigma_\ell)^\star$ is ground. Hence, as $\mathrm{dom}(\psi(D)) \cap \mathrm{dom}(\sigma_\ell) = \emptyset$ we have that $M(\theta\psi(D))^\star = M\theta$ which allows us to conclude.

- For any constraint $M = N \in \mathcal{C}$ we need to show that $M(\theta'^\star\sigma_\ell)^\star =_{\mathsf{E}} N(\theta'^\star\sigma_\ell)^\star$. By hypothesis $M(\theta\sigma_\ell)^\star =_{\mathsf{E}} N(\theta\sigma_\ell)^\star$. As E is closed under substitution of terms for variables we conclude.

- For any constraint $M \neq N \in \mathcal{C}$ we need to show that $M(\theta'^\star\sigma_\ell)^\star \neq_{\mathsf{E}} N(\theta'^\star\sigma_\ell)^\star$. By hypothesis $M(\theta\sigma_\ell)^\star \neq_{\mathsf{E}} N(\theta\sigma_\ell)^\star$. Moreover, we have that $\mathsf{gd}(M) \in \mathcal{C}$ and $\mathsf{gd}(N) \in \mathcal{C}$. Hence, we have that $M\theta'^\star = M\theta$ and $N\theta'^\star = N\theta$ which allows us to conclude.

- $\mathsf{N}_s(names(\mathrm{img}(\theta')) \cup vars(\mathrm{img}(\theta'))) = \mathsf{f}$. By hypothesis we have that $\mathsf{N}_s(names(\mathrm{img}(\theta)) \cup vars(\mathrm{img}(\theta))) = \mathsf{f}$. Moreover, as $bn(D) = \emptyset$ we have that $\mathsf{N}'_s(names(\mathrm{img}(\psi(D)))) = \mathsf{f}$ which implies that $\mathsf{N}_s(names(\mathrm{img}(\psi(D)))) = \mathsf{f}$. Hence we conclude that $\mathsf{N}_s(names(\mathrm{img}(\theta'^\star))) = \mathsf{f}$. We similarly conclude that $\mathsf{N}_s(vars(\mathrm{img}(\theta'^\star))) = \mathsf{f}$.

In order to conclude, it remains to show that $bn(D) \cap names(\mathrm{img}(\theta)) = \emptyset$. This trivially holds since $bn(D) = \emptyset$.

($\Leftarrow$) We again consider two cases

Case $C[\_] = \nu\tilde{n}.\_$.

By hypothesis, $(\theta\psi(C[0]))^\star = \theta \in Sol_{\mathsf{E}}(\mathcal{C}, \mathsf{N}_s)$. We need to show that $\theta \in Sol_{\mathsf{E}}(\nu\tilde{n}.\mathcal{C}, \mathsf{N}'_s)$. The only tricky case is to show that $\tilde{n} \cap names(\mathrm{img}(\theta)) = \emptyset$. However this is directly implied by the additional hypothesis, i.e. $bn(C[0]) \cap names(\mathrm{img}(\theta)) = \emptyset$.

Case $C[\_] = \_ \mid D$.

By hypothesis we have that $(\theta\psi(D))^\star \in Sol_{\mathsf{E}}(\mathcal{C}, \mathsf{N}_s)$. As $C[A]$ is an extended intermediate process, we have that $bn(C[0]) = \emptyset$. Hence, $\mathsf{N}_s = \mathsf{N}'_s$. Let $\mathcal{D}ed(\mathcal{C}) = \{\phi_i \Vdash x_i \mid 1 \leq i \leq \ell\}$ with $\phi_i = \nu\tilde{u}_i.\sigma_i$. Then $\mathcal{D}ed(\mathcal{C} \mid D) = \{\nu\tilde{u}_i.\sigma_i \cup \psi(D) \Vdash x_i \mid 1 \leq i \leq \ell\}$. We have to show that $\theta \in Sol_{\mathsf{E}}(C[\mathcal{C}], \mathsf{N}'_s)$. For this, we need to show that:

- $vars(x_i\theta) \cap cv(\mathcal{C} \mid D) = \emptyset$. By hypothesis $vars(x_i(\theta\psi(D))^\star) \cap cv(\mathcal{C}) = \emptyset$. As $\mathrm{dom}(\psi(D)) \cap cv(\mathcal{C}) = \emptyset$ and $cv(\mathcal{C} \mid D) = cv(\mathcal{C})$ we conclude.

- $vars(x_i\theta) \cap (\mathrm{dom}(\phi_\ell \cup \psi(D)) \setminus \mathrm{dom}(\phi_i \cup \psi(D))) = \emptyset$. By hypothesis we have that $vars(x_i(\theta\psi(D))^\star) \cap (\mathrm{dom}(\phi_\ell) \setminus \mathrm{dom}(\phi_i)) = \emptyset$. Moreover, $\mathrm{dom}(\phi_\ell) \setminus \mathrm{dom}(\phi_i) = \mathrm{dom}(\phi_\ell \cup \psi(D)) \setminus \mathrm{dom}(\phi_i \cup \psi(D))$. Hence it is sufficient to show that $(vars(x_i\theta) \setminus vars(x_i(\theta\psi(D))^\star)) \cap (\mathrm{dom}(\phi_\ell) \setminus \mathrm{dom}(\phi_i)) = \emptyset$. We have that $(vars(x_i\theta) \setminus vars(x_i(\theta\psi(D))^\star)) \subseteq \mathrm{dom}(\psi(D))$. As $\mathrm{dom}(\psi(D)) \cap \mathrm{dom}(\phi_\ell) = \emptyset$ and $\mathrm{dom}(\phi_\ell) \supseteq \mathrm{dom}(\phi_\ell) \setminus \mathrm{dom}(\phi_i)$ we conclude.

- $names(x_i\theta) \cap \tilde{u}_i = \emptyset$. By hypothesis we have that $names(x_i(\theta\psi(D))^\star) \cap \tilde{u}_i = \emptyset$. As $names(x_i\theta) \subseteq names(x_i(\theta\psi(D))^\star)$ we conclude.

- $vars(x_i\theta) \cap \tilde{u}_i = \emptyset$. By hypothesis we have that $vars(x_i(\theta\ \psi(D))^\star) \cap \tilde{u}_i = \emptyset$. We also have that $vars(x_i\theta) \setminus vars(x_i(\theta\psi(D))^\star) \subseteq vars(\mathrm{img}(\psi(D)))$. We have that $\tilde{u}_i \subseteq \mathrm{dom}(A \mid D)$ and, as $(A \mid D)$ is applied, $vars(\mathrm{img}(\psi(D))) \cap \mathrm{dom}(A \mid D) = \emptyset$. Hence $vars(\mathrm{img}(\psi(D))) \cap \tilde{u}_i = \emptyset$ and we conclude.

- For any constraint $\mathsf{gd}(M) \in C[\mathcal{C}]$ we need to show that $M(\theta(\sigma_\ell \cup \psi(D)))^\star$ is ground. Note that $\mathsf{gd}(M) \in \mathcal{C}$. By hypothesis we have that $M((\theta\psi(D))^\star\sigma_\ell)^\star$ is ground. As $C[A]$ is applied we have that $\mathrm{dom}(\sigma_\ell) \cap vars(\mathrm{img}(\psi(D))) = \mathrm{dom}(\psi(D)) \cap vars(\mathrm{img}(\sigma_\ell)) = \emptyset$. Hence, $M((\theta\psi(D))^\star\sigma_\ell)^\star = M(\theta(\sigma_\ell \cup \psi(D)))^\star$ and we conclude. The cases for $M = N$ and $M \neq N$ are similar.

Finally, as $bn(C[0]) = \emptyset$ we obtain that $\theta \in Sol_{\mathsf{E}}(\mathcal{C} \mid D, \mathsf{N}'_s)$. This allows us to conclude the proof. □

B.1 Soundness Results

Proposition B.2 (soundness of $\equiv_s$) *Let $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s)$ and $(B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}_s)$ be two well-formed symbolic processes such that $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s) \equiv_s (B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}_s)$. Then $\mathcal{C}_A = \mathcal{C}_B$ and for all $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_A\ ;\ \mathsf{N}_s)$, we have that $(A\ ;\ \mathsf{N}) \equiv_i (B\ ;\ \mathsf{N})$ where $(A\ ;\ \mathsf{N})$ (resp. $(B\ ;\ \mathsf{N})$) is the θ-concretization of $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s)$ (resp. $(B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}_s)$).*

Proof. We show this result by induction on the proof tree witnessing the fact that $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s) \equiv_s (B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}_s)$. First we need to consider the following base cases:

Case Par-0_s: $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s) \equiv_s (A_s \mid 0\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s)$.
Trivially, $\mathcal{C}_A = \mathcal{C}_B$. Let $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_A\ ;\ \mathsf{N}_s)$ and $(A\ ;\ \mathsf{N})$ (resp. $(B\ ;\ \mathsf{N})$) be the θ-concretization of $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s)$ (resp. $(A_s \mid 0\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s)$). Let σ

be the substitution corresponding to the maximal frame in $\mathcal{C}_A$. We have that $(A\ ;\ \mathsf{N}) = (A_s(\theta\sigma)^\star\ ;\ \mathsf{N}) \equiv_i (A_s(\theta\sigma)^\star \mid 0\ ;\ \mathsf{N}) = ((A_s \mid 0)(\theta\sigma)^\star\ ;\ \mathsf{N}) = (B\ ;\ \mathsf{N})$.

We can deal with the rules PAR-A)$_s$, PAR-C$_s$, and NEW-C$_s$ in a similar way.

We now consider the inductive case, i.e. application of an evaluation context. The proof tree witnessing the fact that $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s) \equiv_s (B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}_s)$ ends with an application of the following inference rule.

$$\frac{(A'_s\ ;\ \mathcal{C}'_A\ ;\ \mathsf{N}'_s) \equiv_s (B'_s\ ;\ \mathcal{C}'_B\ ;\ \mathsf{N}'_s)}{(C[A'_s]\ ;\ C[\mathcal{C}'_A]\ ;\ \mathsf{N}_s) \equiv_s (C[B'_s]\ ;\ C[\mathcal{C}'_B]\ ;\ \mathsf{N}_s)}$$

We have that $\mathsf{N}_s = \mathsf{N}'_s[bn(C[0]) \mapsto \mathsf{b}]$. As $\mathcal{C}'_A = \mathcal{C}'_B$ we directly have that $C[\mathcal{C}'_A] = C[\mathcal{C}'_B]$. Let $\theta \in Sol_{\mathsf{E}}(C[\mathcal{C}'_A]\ ;\ \mathsf{N}_s)$ and $(A\ ;\ \mathsf{N})$ (resp. $(B\ ;\ \mathsf{N})$) be the θ-concretization of $(C[A'_s]\ ;\ C[\mathcal{C}'_A]\ ;\ \mathsf{N}_s)$ (resp. $(C[B'_s]\ ;\ C[\mathcal{C}'_B]\ ;\ \mathsf{N}_s)$). Let σ be the substitution corresponding to the maximal frame in $\mathcal{C}_A = C[\mathcal{C}'_A]$. We have to show that $(A\ ;\ \mathsf{N}) \equiv_i (B\ ;\ \mathsf{N})$, i.e. $(C(\theta\sigma)^\star[A'_s(\theta\sigma)^\star]\ ;\ \mathsf{N}) \equiv_i (C(\theta\sigma)^\star[B'_s(\theta\sigma)^\star]\ ;\ \mathsf{N})$. Since $\equiv_i$ is closed under application of evaluation context, it is sufficient to show that $(A'_s(\theta\sigma)^\star\ ;\ \mathsf{N}') \equiv_i (B'_s(\theta\sigma)^\star\ ;\ \mathsf{N}')$ where $\mathsf{N}' = \mathsf{N}'_s|_{\mathcal{N}\cup\mathcal{X}}$.

Let $\theta' = (\theta\psi(C[0]))^\star$. By Lemma B.1 we have that $\theta' \in Sol_{\mathsf{E}}(\mathcal{C}'_A\ ;\ \mathsf{N}'_s)$ and $(A'\ ;\ \mathsf{N}') \equiv_i (B'\ ;\ \mathsf{N}')$ where $(A'\ ;\ \mathsf{N}')$ (resp. $(B'\ ;\ \mathsf{N}')$) is the θ'-concretization of $(A'_s\ ;\ \mathcal{C}'_A\ ;\ \mathsf{N}'_s)$ (resp. $(B'_s\ ;\ \mathcal{C}'_B\ ;\ \mathsf{N}'_s)$). We have that $A' = A'_s(\theta'\sigma')^\star$ and $B' = B'_s(\theta'\sigma')^\star$ where σ' is the maximal frame of $\mathcal{C}'_A$. This allows us to conclude since $(\theta'\sigma')^\star = ((\theta\psi(C[0]))^\star\sigma')^\star = (\theta(\psi(C[0])\sigma'))^\star = (\theta\sigma)^\star$. □

Proposition B.3 (soundness of $\rightarrow_s$) *Let $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s)$ and $(B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}_s)$ be two well-formed symbolic processes such that $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s) \rightarrow_s (B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}_s)$. Let $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_B\ ;\ \mathsf{N}_s)$. We have that $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_A\ ;\ \mathsf{N}_s)$ and $(A\ ;\ \mathsf{N}) \rightarrow_i (B\ ;\ \mathsf{N})$ where $(A\ ;\ \mathsf{N})$ (resp. $(B\ ;\ \mathsf{N})$) is the θ-concretization of $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s)$ (resp. $(B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}_s)$).*

Proof. The proof is done by induction on the proof witnessing $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s) \rightarrow_s (B_s\ ;\ \mathcal{C}_B\ ;\ \mathsf{N}_s)$. We first consider the three base cases.

Case COMM$_s$. We have that $A_s = \mathsf{out}(u, M).P_s \mid \mathsf{in}(v, x).Q_s$, $B_s = P_s \mid Q_s\{^M/_x\}$ and $\mathcal{C}_B = \mathcal{C}_A \cup \{u = v, \mathsf{gd}(u), \mathsf{gd}(v)\}$. Let $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_A \cup \{u = v\ ,\ \mathsf{gd}(u)\ ,\ \mathsf{gd}(v)\}\ ;\ \mathsf{N}_s)$. We also have that $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_A\ ;\ \mathsf{N}_s)$. Let σ_A (resp. σ_B) be the substitution corresponding to the maximal frame of $\mathcal{C}_A$ (resp. $\mathcal{C}_B$). Trivially, we have that $\sigma_A = \sigma_B$. Let $\rho = (\theta\sigma_A)^\star$.

$$\begin{array}{rcll}
(A\ ;\ \mathsf{N}) & = & (A_s\rho\ ;\ \mathsf{N}) & \\
& = & (\mathsf{out}(u\rho, M\rho).P_s\rho \mid \mathsf{in}(v\rho, x).Q_s\rho\ ;\ \mathsf{N}) & \text{as } x \notin \mathrm{dom}(\rho) \\
& = & (\mathsf{out}(u\rho, M\rho).P_s\rho \mid \mathsf{in}(u\rho, x).Q_s\rho\ ;\ \mathsf{N}) & \text{as } \theta \in Sol_{\mathsf{E}}(\mathcal{C}_B\ ;\ \mathsf{N}_s) \\
& & & u, v \text{ are of channel type} \\
& \rightarrow_i & (P_s\rho \mid Q_s\rho\{^{M\rho}/_x\}\ ;\ \mathsf{N}) & \text{as } u\rho \text{ is a channel name} \\
& & & \text{as } \mathsf{gd}(u) \in \mathcal{C}_B \\
& = & ((P_s \mid Q_s\{^M/_x\})\rho\ ;\ \mathsf{N}) & \\
& = & (B_s\rho\ ;\ \mathsf{N}) & \\
& = & (B\ ;\ \mathsf{N}) & \text{as } \sigma_A = \sigma_B
\end{array}$$

Again, the rules THEN_s and ELSE_s are similar to the previous case. We now consider the two inductive cases.

The case of the structural equivalence rule is straightforward.

Case APPLICATION OF AN EVALUATION CONTEXT

The proof witnessing the fact that $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s) \rightarrow_s (B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s)$ ends with an application of the following inference rule.

$$\frac{(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s) \rightarrow_s (B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)}{(C[A'_s] \,;\, C[\mathcal{C}'_A] \,;\, \mathsf{N}_s) \rightarrow_s (C[B'_s] \,;\, C[\mathcal{C}'_B] \,;\, \mathsf{N}_s)}$$

We have that $\mathsf{N}_s = \mathsf{N}'_s[bn(C[0]) \mapsto \mathsf{b}]$. Let $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_B \,;\, \mathsf{N}_s)$ and $(A \,;\, \mathsf{N})$ (resp. $(B \,;\, \mathsf{N})$) be the θ-concretization of $(C[A'_s] \,;\, C[\mathcal{C}'_A] \,;\, \mathsf{N}_s)$ (resp. $(C[B'_s] \,;\, C[\mathcal{C}'_B] \,;\, \mathsf{N}_s)$). Let σ'_A (resp. σ'_B) be the substitution corresponding to the maximal frame in $\mathcal{C}'_A$ and σ_A (resp. σ_B) be the substitution corresponding to the maximal frame in $\mathcal{C}_A = C[\mathcal{C}'_A]$ (resp. $\mathcal{C}_B = C[\mathcal{C}'_B]$). Note that since $\rightarrow_s$ does never add deduction constraints we have that $\sigma'_A = \sigma'_B$ and hence $\sigma_A = \sigma_B$. We have to show that $(A \,;\, \mathsf{N}) \rightarrow_i (B \,;\, \mathsf{N})$, i.e. $(C(\theta\sigma_A)^\star[A'_s(\theta\sigma_A)^\star] \,;\, \mathsf{N}) \rightarrow_i (C(\theta\sigma_B)^\star[B'_s(\theta\sigma_B)^\star] \,;\, \mathsf{N})$. Since $\rightarrow_i$ is closed under application of evaluation context, it is sufficient to show that $(A'_s(\theta\sigma_A)^\star \,;\, \mathsf{N}') \rightarrow_i (B'_s(\theta\sigma_B)^\star \,;\, \mathsf{N}')$ where $\mathsf{N}' = \mathsf{N}'_s|_{\mathcal{N}\cup\mathcal{X}}$.

Let $\theta' = (\theta\psi(C[0]))^\star$. By Lemma B.1 we have that $\theta' \in Sol_{\mathsf{E}}(\mathcal{C}'_B \,;\, \mathsf{N}'_s)$ and hence by induction hypothesis, we deduce that $\theta' \in Sol_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}'_s)$ and $(A' \,;\, \mathsf{N}) \rightarrow_i (B' \,;\, \mathsf{N})$ where $(A' \,;\, \mathsf{N}')$ (resp. $(B' \,;\, \mathsf{N}')$) is the θ'-concretization of $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s)$ (resp. $(B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)$). Hence, by Lemma B.1 we deduce that $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_A \,;\, \mathsf{N}_s) = Sol_{\mathsf{E}}(C[\mathcal{C}'_A] \,;\, \mathsf{N}_s)$. We have that $A' = A'_s(\theta'\sigma'_A)^\star$ and $B' = B'_s(\theta'\sigma'_B)^\star$. This allows us to conclude since $(\theta'\sigma'_A)^\star = (\theta\sigma_A)^\star$ and $(\theta'\sigma'_B)^\star = (\theta\sigma_B)^\star$. □

Proposition 8.2 (soundness of $\xrightarrow{\alpha}_s$) *Let $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ and $(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}'_s)$ be two well-formed symbolic processes such that $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s) \xrightarrow{\alpha_s}_s (B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}'_s)$. Let $\theta_B \in Sol_{\mathsf{E}}(\mathcal{C}_B \,;\, \mathsf{N}'_s)$ and $\theta_A = \theta_B|_{cv(\mathcal{C}_A)}$. We have that $\theta_A \in Sol_{\mathsf{E}}(\mathcal{C}_A \,;\, \mathsf{N}_s)$ and $(A \,;\, \mathsf{N}) \xrightarrow{\alpha_s\theta_B}_i (B \,;\, \mathsf{N}')$, where $(A \,;\, \mathsf{N})$ and $(B \,;\, \mathsf{N}')$ are respectively the θ_A-concretization and the θ_B-concretization of $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ and $(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}'_s)$.*

Proof. The proof is done by induction on the proof tree witnessing the following reduction step $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s) \xrightarrow{\alpha_s}_s (B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}'_s)$. We first consider the three base cases.

Case IN_s. We have that $A_s = \mathsf{in}(u, x).P_s$, $B_s = P_s\{^y/_x\}$, $\alpha_s = in(u, y)$ for some $y \in \mathcal{Y}$ such that $\mathsf{N}_s(y) = \mathsf{n}$ and $\mathcal{C}_B = \mathcal{C}_A \cup \{0 \Vdash y, \mathsf{gd}(u)\})$. Moreover, we have that $\mathsf{N}'_s = \mathsf{N}_s[y \mapsto \mathsf{c}]$. Note that we have that $\mathsf{N} = \mathsf{N}'$. Let $\theta_B \in Sol_{\mathsf{E}}(\mathcal{C}_B \,;\, \mathsf{N}'_s)$ and $\theta_A = \theta_B|_{cv(\mathcal{C}_A)}$. As $\mathcal{C}_A \subset \mathcal{C}_B$ we have that $\theta_A \in Sol_{\mathsf{E}}(\mathcal{C}_A \,;\, \mathsf{N}_s)$. Let σ_A (resp. σ_B) be the substitution corresponding to the maximal frame of $\mathcal{C}_A$ (resp. $\mathcal{C}_B$). Trivially, we have that $\mathrm{dom}(\sigma_A) = \mathrm{dom}(\sigma_B) = \emptyset$ since $\phi(P_s) = 0$ and $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ is well-formed.

$$
\begin{array}{lcll}
(A\ ;\ \mathsf{N}) & = & (A_s\theta_A\ ;\ \mathsf{N}) & \\
& = & (\mathsf{in}(u\theta_A, x).P_s\theta_A\ ;\ \mathsf{N}) & \text{as } x \notin dom(\theta_A) \\
& \xrightarrow{in(u,y)\theta_B}_i & (P_s\theta_A\{^{y\theta_B}/_x\}\ ;\ \mathsf{N}') & \text{as } u\theta_B \in \mathcal{N}_{ch},\ \mathsf{N}(fv(y\theta_B) \cup fn(y\theta_B)) = \mathsf{f} \\
& = & (P_s\theta_A\{^{y}/_x\}\{^{y\theta_B}/_y\}\ ;\ \mathsf{N}') & \\
& = & (B_s\theta_B\ ;\ \mathsf{N}') & \text{as } \theta_B = \theta_A \cup \{y \mapsto y\theta_B\} \\
& = & (B\ ;\ \mathsf{N}') &
\end{array}
$$

Case OUT-CH$_s$. This case is similar to the previous one.
Case OUT-T$_s$. We have that $A_s = \mathsf{out}(u, M).P_s$, $B_s = P_s \mid \{^M/_x\}$, $\alpha_s = \nu x.out(u, x)$ where $x \in \mathcal{X}_b$ and $\mathsf{N}_s(x) = \mathsf{n}$. We have also that $\mathcal{C}_B = \nu x.\mathcal{C}_A \cup \{\mathsf{gd}(u)\}$ and $\mathsf{N}'_s = \mathsf{N}_s[x \mapsto \mathsf{f}]$. Let $\theta_B \in Sol_{\mathsf{E}}(\mathcal{C}_B\ ;\ \mathsf{N}'_s)$ and $\theta_A = \theta_B|_{cv(\mathcal{C}_A)}$, i.e. $\theta_B = \theta_A$. As $\nu x.\mathcal{C}_A \subset \mathcal{C}_B$ we have that $\theta_A \in Sol_{\mathsf{E}}(\mathcal{C}_A\ ;\ \mathsf{N}_s)$. Let σ_A (resp. σ_B) be the substitution corresponding to the maximal frame of $\mathcal{C}_A$ (resp. $\mathcal{C}_B$). Trivially, we have that $\mathrm{dom}(\sigma_A) = \mathrm{dom}(\sigma_B) = \emptyset$ since $\phi(P_s) = 0$ and $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s)$ is well-formed.

$$
\begin{array}{lcll}
(A\ ;\ \mathsf{N}) & = & (A_s\theta_A\ ;\ \mathsf{N}) & \\
& = & (\mathsf{out}(u\theta_A, M\theta_A).P_s\theta_A\ ;\ \mathsf{N}) & \\
& \xrightarrow{\nu x.out(u,x)\theta_B}_i & (P_s\theta_A \mid \{^{M\theta_A}/_x\}\ ;\ \mathsf{N}') & \text{as } x \notin \mathrm{dom}(\theta_B) \\
& = & (P_s\theta_B \mid \{^{M\theta_B}/_x\}\ ;\ \mathsf{N}') & \text{as } \theta_A = \theta_B \\
& = & (B_s\theta_B\ ;\ \mathsf{N}) & \\
& = & (B\ ;\ \mathsf{N}) &
\end{array}
$$

Moreover, as $\theta_B \in Sol_{\mathsf{E}}(\nu x.\mathcal{C}_A)$ we have that $x \notin \mathrm{img}(\theta_B)$ and hence, x occurs only once in B.

We now consider the inductive cases.

Case OPEN-CH$_s$.

$$
\frac{(A'_s\ ;\ \mathcal{C}'_A\ ;\ \mathsf{N}''_s) \xrightarrow{out(u,c)}_s (B'_s\ ;\ \mathcal{C}'_B\ ;\ \mathsf{N}'''_s) \quad u \neq c,\ \mathsf{N}''_s(d) = \mathsf{n},\ d \in \mathcal{N}_{ch}}{(\nu c.A'_s\ ;\ \nu c.\mathcal{C}'_A\ ;\ \mathsf{N}_s) \xrightarrow{\nu d.out(u,d)}_s (B'_s\{^d/_c\}\ ;\ \nu d.(\mathcal{C}'_B\{^d/_c\})\ ;\ \mathsf{N}'_s)}
$$

We have that $A_s = \nu c.A'_s$, $B_s = B'_s\{^d/_c\}$, $\alpha_s = \nu d.out(u, d)$, $\mathcal{C}_A = \nu c.\mathcal{C}'_A$ and $\mathcal{C}_B = \nu d.\mathcal{C}'_B\{^d/_c\}$. Moreover, we have that

- $\mathsf{N}_s = \mathsf{N}''_s[c \mapsto \mathsf{b}]$, and
- $\mathsf{N}'_s = \mathsf{N}'''_s[c \mapsto \mathsf{b},\ d \mapsto \mathsf{f}]$.

Let $\theta_B \in Sol_{\mathsf{E}}(\mathcal{C}_B\ ;\ \mathsf{N}'_s)$. We also have that $\theta_B \in Sol_{\mathsf{E}}(\mathcal{C}'_B\ ;\ \mathsf{N}'''_s)$ and $c, d \notin names(\mathrm{img}(\theta_B))$ and $u\theta_B \neq c$. Let $\theta_A = \theta_B|_{cv(\mathcal{C}_A)}$, i.e. $\theta_B = \theta_A$. By induction hypothesis we deduce that $(A'\ ;\ \mathsf{N}'') \xrightarrow{out(u,c)\theta_B}_i (B'\ ;\ \mathsf{N}''')$ where $(A'\ ;\ \mathsf{N}'')$ (resp. $(B'\ ;\ \mathsf{N}''')$) are the θ_B-concretization of $(A'_s\ ;\ \mathcal{C}'_A\ ;\ \mathsf{N}''_s)$ (resp. $(B'_s\ ;\ \mathcal{C}'_B\ ;\ \mathsf{N}'''_s)$) and $\theta_B \in Sol_{\mathsf{E}}(\mathcal{C}'_A\ ;\ \mathsf{N}''_s)$. As $c \notin names(\mathrm{img}(\theta_B))$, we also have that

$c \notin names(\text{img}(\theta_A))$ and hence $\theta_A \in Sol_{\mathsf{E}}(\mathcal{C}_A \,;\, \mathsf{N}_s)$. Since $(A' \,;\, \mathsf{N}'') \xrightarrow{out(u,c)\theta_B}_i (B' \,;\, \mathsf{N}''')$, we deduce that $(\nu c.A' \,;\, \mathsf{N}) \xrightarrow{\nu d.out(u\theta_B,d)}_i (B'\{^d/_c\} \,;\, \mathsf{N}''')$. Note that $c \neq u\theta_B$ and $d \in \mathcal{N}_{ch}$ and $\mathsf{N}''(d) = \mathsf{n}$.

Case SCOPE$_s$.

$$\frac{(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}''_s) \xrightarrow{\alpha}_s (B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'''_s) \quad n \text{ does not occur in } \alpha}{(\nu n.A'_s \,;\, \nu n.\mathcal{C}'_A \,;\, \mathsf{N}_s) \xrightarrow{\alpha}_s (\nu n.B'_s \,;\, \nu n.\mathcal{C}'_B \,;\, \mathsf{N}'_s)}$$

We have that $A_s = \nu n.A'_s$, $B_s = \nu n.B'_s$, $\mathcal{C}_A = \nu n.\mathcal{C}'_A$ and $\mathcal{C}_B = \nu n.\mathcal{C}'_B$. Moreover, we have that $\mathsf{N}_s = \mathsf{N}''_s[n \mapsto \mathsf{b}]$ and $\mathsf{N}'_s = \mathsf{N}'''_s[n \mapsto \mathsf{b}]$. Let $\theta_B \in Sol_{\mathsf{E}}(\mathcal{C}_B \,;\, \mathsf{N}'_s)$. We have that $n \notin names(\text{img}(\theta_B))$. Let $\theta'_B = \theta_B$. By Lemma B.1 we have that $\theta'_B \in Sol_{\mathsf{E}}(\mathcal{C}'_B \,;\, \mathsf{N}'''_s)$. Let $\theta'_A = \theta'_B|_{cv(\mathcal{C}'_A)}$. By induction hypothesis, we have that $(A' \,;\, \mathsf{N}'') \xrightarrow{\alpha\theta'_B}_i (B' \,;\, \mathsf{N}''')$ where $(A' \,;\, \mathsf{N}'')$ and $(B' \,;\, \mathsf{N}''')$ are respectively the θ'_A and the θ'_B-concretization of $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}''_s)$ and $(B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'''_s)$. As $n \notin names(\text{img}(\theta_B))$, n does not occur in $\alpha\theta'_B$ and $\theta_A = \theta'_A \in Sol_{\mathsf{E}}(\mathcal{C}_A \,;\, \mathsf{N}_s)$ by Lemma B.1. Since $(A' \,;\, \mathsf{N}'') \xrightarrow{\alpha\theta'_B}_i (B' \,;\, \mathsf{N}''')$, $\theta_B = \theta'_B$ and n does not occur in $\alpha\theta_B$, we deduce that $(\nu n.A' \,;\, \mathsf{N}) \xrightarrow{\alpha\theta_B}_i (\nu n.B' \,;\, \mathsf{N}')$

Case PAR$_s$.

$$\frac{(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}_s) \xrightarrow{\alpha}_s (B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)}{(A'_s \mid D_s \,;\, \mathcal{C}'_A \mid \psi(D_s) \,;\, \mathsf{N}_s) \xrightarrow{\alpha}_s (B'_s \mid D_s \,;\, \mathcal{C}'_B \mid \psi(D_s) \,;\, \mathsf{N}'_s)}$$

We have that $A_s = A'_s \mid D_s$, $B_s = B'_s \mid D_s$, $\mathcal{C}_A = \mathcal{C}'_A \mid \psi(D_s)$ and $\mathcal{C}_B = \mathcal{C}'_B \mid \psi(D_s)$. Let $\theta_B \in Sol_{\mathsf{E}}(\mathcal{C}_B \,;\, \mathsf{N}'_s)$. Then, by Lemma B.1 we also have that $\theta'_B = (\theta_B\psi(D_s))^\star \in Sol_{\mathsf{E}}(\mathcal{C}'_B \,;\, \mathsf{N}'_s)$. Let $\theta'_A = \theta'_B|_{cv(\mathcal{C}'_A)}$. By induction hypothesis we have that $\theta'_A \in Sol_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}_s)$ and $(A' \,;\, \mathsf{N}) \xrightarrow{\alpha\theta_B\psi(D_s)}_i (B' \,;\, \mathsf{N}')$ where $(A' \,;\, \mathsf{N})$ and $(B' \,;\, \mathsf{N}')$ are respectively the θ'_A and the θ'_B concretization of $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}_s)$ and $(B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)$.

Let $\theta_A = \theta_B|_{cv(\mathcal{C}_A)}$. We have $\theta'_A = \theta_B\psi(D_s)|_{cv(\mathcal{C}_A)}$ and $\theta'_A \in Sol_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}_s)$. Hence by Lemma B.1 we have that $\theta_B|_{cv(\mathcal{C}_A)} \in Sol_{\mathsf{E}}(\mathcal{C}'_A \mid \psi(D_s) \,;\, \mathsf{N}_s)$, i.e. $\theta_A \in Sol_{\mathsf{E}}(\mathcal{C}_A \,;\, \mathsf{N}_s)$. Let σ_A (resp. σ_B, σ'_A and σ'_B) be the substitution corresponding to the maximal frame of $\mathcal{C}_A$ (resp. $\mathcal{C}_B$, $\mathcal{C}'_A$ and $\mathcal{C}'_B$), We also have that $\sigma_A = \sigma'_A \cup \psi(D_s)$ and $\sigma_B = \sigma'_B \cup \psi(D_s)$. As $\mathsf{N}'_s(\text{dom}(\psi(D_s))) = \mathsf{f}$ and $\mathsf{N}'_s(fv(\alpha)) = \mathsf{c}$, we have that $\alpha\theta_B\psi(D_s) = \alpha\theta_B$. Hence, we have that $(A \,;\, \mathsf{N}) \xrightarrow{\alpha\theta_B}_i (B \,;\, \mathsf{N}')$ where $(A \,;\, \mathsf{N})$ and $(B \,;\, \mathsf{N}')$ are respectively the θ_A and the θ_B concretization of $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ and $(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}'_s)$.

Case STRUCT$_s$. This case is straightforward by relying on Proposition B.2.

□

B.2 Completeness Results

Proposition B.4 (completeness of $\equiv_s$) *Let $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ be a well-formed symbolic process and $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_A, \mathsf{N}_s)$. Let $(A \,;\, \mathsf{N})$ be the θ-concretization of $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ and B be a process such that $(A \,;\, \mathsf{N}) \equiv_i (B \,;\, \mathsf{N})$. Then there exists a well-formed symbolic process $(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s)$ such that:*

1. $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s) \equiv_s (B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s)$,
2. $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_B \,;\, \mathsf{N}_s)$, *and*
3. $(B \,;\, \mathsf{N})$ *is the θ-concretization of* $(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s)$.

Proof. We show this result by induction on the proof tree witnessing the fact that $(A, \mathsf{N}) \equiv_i (B, \mathsf{N})$. First we need to consider the following base cases:

Case PAR-0_i: $(D, \mathsf{N}) \equiv_i (D \mid 0, \mathsf{N})$. In such a case, we have that $A = D$ and $B = D \mid 0$. Moreover, since $(A \,;\, \mathsf{N})$ is the θ-concretization of $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ we have that $A = A_s(\theta\sigma_A)^\star$. Hence, we know that $A_s = D_s$ for some process D_s such that $D_s(\theta\sigma_A)^\star = D$. Let $B_s = D_s \mid 0$ and $\mathcal{C}_B = \mathcal{C}_A$. The symbolic process $(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s)$ is well-formed. Moreover, we have

1. $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s) \equiv_s (B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s)$,
2. $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_B \,;\, \mathsf{N}_s)$,
3. $B_s(\theta\sigma_B)^\star = (D_s \mid 0)(\theta\sigma_A)^\star = D \mid 0 = B$, i.e., $(B \,;\, \mathsf{N})$ is the θ-concretization of $(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s)$.

Symmetrically, we have to consider the case where $A = D \mid 0$ and $B = D$. We know that $A = A_s(\theta\sigma_A)^\star$ and we deduce that $A_s = D_s|0$ for some process D_s such that $D_s(\theta\sigma_A)^\star = D$. Let $B_s = D_s$ and $\mathcal{C}_B = \mathcal{C}_A$. We easily conclude.

We can deal with the rules PAR-A, PAR-C and NEW-C in a similar way.

Now, we show the inductive case, i.e. application of an evaluation context. In such a case, we have that the proof tree witnessing the fact that $(A \,;\, \mathsf{N}) \equiv_i (B \,;\, \mathsf{N})$ ends with an application of the following inference rule.

$$\frac{(A' \,;\, \mathsf{N}') \equiv_i (B' \,;\, \mathsf{N}')}{(C[A'] \,;\, \mathsf{N}) \equiv_i (C[B'] \,;\, \mathsf{N})}$$

Moreover, since $(A \,;\, \mathsf{N})$ is the θ-concretization of $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ we have that $A_s(\theta\sigma_A)^\star = C[A']$. Hence, we deduce that $A_s = C_s[A'_s]$ for some evaluation context C_s and some process A'_s such that $C_s(\theta\sigma_A)^\star = C$ and $A'_s(\theta\sigma_A)^\star = A'$. Since $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ is well-formed, we have also that $\mathcal{C}_A = C_s[\mathcal{C}'_A]$ for some constraint system $\mathcal{C}'_A$. Let

$$\mathsf{N}'_s(u) = \begin{cases} \mathsf{N}'(u) & \text{if } u \in \mathcal{N} \cup \mathcal{X} \\ \mathsf{N}_s(u) & \text{if } u \in \mathcal{Y}. \end{cases}$$

Let $\theta' = (\theta\psi(C_s))^\star$. By Lemma B.1 we have that $\theta' \in Sol_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}'_s)$. We can apply our induction hypothesis on $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s)$ and $(A' \,;\, \mathsf{N}') \equiv_i (B' \,;\, \mathsf{N}')$.

We deduce that there exists a well-formed symbolic process $(B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)$ such that $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s) \equiv_s (B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)$, $\theta' \in Sol_{\mathsf{E}}(\mathcal{C}'_B \,;\, \mathsf{N}'_s)$ and $B'_s(\theta'\sigma'_B)^\star = B'$. Let $B_s = C_s[B'_s]$ and $\mathcal{C}_B = C_s[\mathcal{C}'_B]$. We have that

1. $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s) \equiv (B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s)$,
2. $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_B \,;\, \mathsf{N_s})$ (by Lemma B.1),
3. $B_s(\theta\sigma_B)^\star = C_s(\theta\sigma_B)^\star[B'_s(\theta\sigma_B)^\star] = C[B'_s(\theta'\sigma'_B)^\star] = C[B'] = B$, i.e., $(B \,;\, \mathsf{N})$ is the θ-concretization of $(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s)$.

This concludes our proof. □

Proposition B.5 (completeness of $\rightarrow_s$) *Let $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ be a well-formed symbolic process and $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_A \,;\, \mathsf{N}_s)$. Let $(A \,;\, \mathsf{N})$ be the θ-concretization of $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ and (A', N) be an intermediate process such that $(A \,;\, \mathsf{N}) \rightarrow_i (A' \,;\, \mathsf{N})$. Then there exists a well-formed symbolic process $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}_s)$ such that:*

1. $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s) \rightarrow_s (A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}_s)$,
2. $\theta \in Sol_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}_s)$,
3. $(A' \,;\, \mathsf{N})$ *is the θ-concretization of* $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}_s)$.

Proof. We show this result by induction on the proof tree witnessing the fact that $(A \,;\, \mathsf{N}) \rightarrow_i (A' \,;\, \mathsf{N})$. First, we need to consider the three base cases, i.e. the rules Then, Else and Comm. We detail the case of the rule Else, the two other ones are very similar.
Case Else: (if $M = N$ then P else $Q \,;\, \mathsf{N}) \rightarrow_i (Q \,;\, \mathsf{N})$ with M, N ground terms such that $M \neq_{\mathsf{E}} N$. In such a case, we have that

- $A =$ if $M = N$ then P else Q for some ground terms M, N such that $M \neq_{\mathsf{E}} N$ and some processes P and Q, and
- $A' = Q$.

Moreover, since $(A \,;\, \mathsf{N})$ is the θ-concretization of $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ we have that $A = A_s(\theta\sigma_A)^\star$. Hence, we deduce that

- $A_s =$ if $M_s = N_s$ then P_s else Q_s for some terms M_s, N_s, some processes P_s and Q_s such that
- $M_s(\theta\sigma_A)^\star = M$, $N_s(\theta\sigma_A)^\star = N$, $P_s(\theta\sigma_A)^\star = P$ and $Q_s(\theta\sigma_A)^\star = Q$.

Let $A'_s = Q_s$ and $\mathcal{C}'_A = \mathcal{C}_A \cup \{M_s \neq N_s \,,\, \mathsf{gd}(M_s) \,,\, \mathsf{gd}(N_s)\}$. The symbolic process $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}_s)$ is well-formed. Moreover, we have

1. $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s) \rightarrow_s (Q_s \,;\, \mathcal{C}_A \cup \{M_s \neq N_s \,,\, \mathsf{gd}(M_s) \,,\, \mathsf{gd}(N_s)\} \,;\, \mathsf{N}_s)$
2. $\theta \in Sol_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}_s)$. Indeed, by hypothesis, we know that $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_A \,;\, \mathsf{N}_s)$. We know also that $M_s(\theta\sigma_A)^\star$ and $N_s(\theta\sigma_A)^\star$ are ground terms which are not equal modulo E.

3. $A'_s(\theta\sigma_A)^\star = Q_s(\theta\sigma_A)^\star = Q = A'$, i.e., $(A' \,;\, \mathsf{N})$ is the θ-concretization of $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}_s)$.

Now, we show the inductive cases. In case of the structural equivalence inductive rule, we easily conclude by induction and thanks to Proposition B.4.

Case Application of an evaluation context. In such a case, we have that the tree witnessing the fact that $(A \,;\, \mathsf{N}) \rightarrow_i (B \,;\, \mathsf{N})$ ends with an application of the following inference rule.

$$\frac{(A' \,;\, \mathsf{N}') \rightarrow_i (B' \,;\, \mathsf{N}')}{(C[A'] \,;\, \mathsf{N}) \rightarrow_i (C[B'] \,;\, \mathsf{N})}$$

Moreover, since $(A \,;\, \mathsf{N})$ is the θ-concretization of $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ we have that $A_s(\theta\sigma_A)^\star = C[A']$. Hence, we deduce that $A_s = C_s[A'_s]$ for some evaluation context C_s and some process A'_s such that $C_s(\theta\sigma_A)^\star = C$ and $A'_s(\theta\sigma_A)^\star = A'$. Since $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ is well-formed, we have also that $\mathcal{C}_A = C_s[\mathcal{C}'_A]$ for some constraint system $\mathcal{C}'_A$. Let

$$\mathsf{N}'_s(u) = \begin{cases} \mathsf{N}'(u) & \text{if } u \in \mathcal{N} \cup \mathcal{X} \\ \mathsf{N}_s(u) & \text{if } u \in \mathcal{Y} \end{cases}$$

Let $\theta' = (\theta\phi(C_s))^\star$. By Lemma B.1 we have that $\theta' \in Sol_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}'_s)$.

We can apply our induction hypothesis on $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s)$ and $(A' \,;\, \mathsf{N}') \rightarrow_i (B' \,;\, \mathsf{N}')$. We deduce that there exists a well-formed symbolic process $(B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)$ such that $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s) \rightarrow_s (B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)$, $\theta' \in Sol_{\mathsf{E}}(\mathcal{C}'_B \,;\, \mathsf{N}'_s)$ and $(B' \,;\, \mathsf{N}')$ is the θ-concretization of $(B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)$. Let $B_s = C_s[B'_s]$ and $\mathcal{C}_B = C_s[\mathcal{C}'_B]$. We have that

1. $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s) \rightarrow_s (B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s)$,
2. $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_B \,;\, \mathsf{N}_s)$ (by Lemma B.1),
3. $B_s(\theta\sigma_B)^\star = C_s(\theta\sigma_B)^\star[B'_s(\theta\sigma_B)^\star] = C[B'_s(\theta'\sigma'_B)^\star] = C[B'] = B$ i.e., $(B \,;\, \mathsf{N})$ is the θ-concretization of $(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s)$.

This concludes our proof. □

Proposition 8.4 (completeness of $\xrightarrow{\alpha}_s$) *Let $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ be a well-formed symbolic process and $\theta_A \in Sol_{\mathsf{E}}(\mathcal{C}_A \,;\, \mathsf{N}_s)$. Let $(A \,;\, \mathsf{N})$ be the θ_A-concretization of $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ and $(A' \,;\, \mathsf{N}')$ be an intermediate process such that $(A \,;\, \mathsf{N}) \xrightarrow{\alpha}_i (A' \,;\, \mathsf{N}')$. Then there exists a well-formed symbolic process $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s)$ and a substitution θ'_A such that:*

1. $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s) \xrightarrow{\alpha_s}_s (A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s)$,
2. $\theta'_A \in Sol_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}'_s)$ *and* $\theta'_A|_{cv(\mathcal{C}_A)} = \theta_A$,
3. $(A' \,;\, \mathsf{N}')$ *is the* θ'_A*-concretization of* $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s)$, *and*
4. $\alpha_s\theta'_A = \alpha$.

Proof. We show this result by induction on the tree witnessing the fact that $(A\ ;\ \mathsf{N}) \xrightarrow{\alpha}_i (A'\ ;\ \mathsf{N}')$. First, we need to consider the following base cases:

Case IN$_i$: $(\mathsf{in}(a,x).P\ ;\ \mathsf{N}) \xrightarrow{in(a,M)}_i (P\{^M/_x\}\ ;\ \mathsf{N})$. In such a case, we have that

- $A = \mathsf{in}(a,x).P$ for some channel name a, some variable x and some P,
- $A' = P\{^M/_x\}$,
- $\alpha = \mathsf{in}(a,M)$ for some term M, and
- $\mathsf{N}(fn(M) \cup fv(M)) = \mathsf{f}$.

Since $(A\ ;\ \mathsf{N})$ is the θ-concretization of $(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s)$ we have that $A_s(\theta\sigma_A)^\star = A$. Hence, we know that

- $A_s = \mathsf{in}(u,x).P_s$ for some metavariable u and some process P_s such that
- $u(\theta\sigma_A)^\star = a$ and $P_s(\theta\sigma_A)^\star = P$.

We have that u is either a channel name or a constraint variable of channel type since $u(\theta\sigma_A)^\star = a$ and a is a channel name.

Let $y \in \mathcal{Y}$ having the same type than M and such that $\mathsf{N}_s(y) = \mathsf{n}$. Let $A'_s = P_s\{^y/_x\}$, $\mathcal{C}'_A = \mathcal{C}_A \cup \{0 \Vdash y\ ,\ \mathsf{gd}(u)\}$, $\alpha_s = \mathsf{in}(u,y)$, $\theta' = \theta \cup \{y \mapsto M\}$ and $\mathsf{N}'_s = \mathsf{N}_s[y \mapsto \mathsf{c}]$. The symbolic process $(A'_s\ ;\ \mathcal{C}'_A\ ;\ \mathsf{N}'_s)$ is well-formed, and we have:

1. $$\begin{aligned}(A_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s) &= (\mathsf{in}(u,x).P_s\ ;\ \mathcal{C}_A\ ;\ \mathsf{N}_s)\\ &\xrightarrow{\alpha_s}_s (P_s\{^y/_x\}\ ;\ \mathcal{C}_A \cup \{0 \Vdash y\ ,\ \mathsf{gd}(u)\}\ ;\ \mathsf{N}'_s)\\ &= (A'_s\ ;\ \mathcal{C}'_A\ ;\ \mathsf{N}'s)\end{aligned}$$
2. We know that $\theta \in Sol_{\mathsf{E}}(\mathcal{C}_A\ ;\ \mathsf{N}_s)$. It remains to check that $\theta' \in Sol_{\mathsf{E}}(\mathcal{C}'_A\ ;\ \mathsf{N}'_s)$, i.e. θ' satisfies the constraints $\Vdash y$ and $\mathsf{gd}(u)$. This is clearly true due to the fact that $\mathsf{N}_s(fn(M) \cup fv(M)) = \mathsf{f}$ and $u(\theta\sigma_A)^\star = a$. Lastly, by definition of θ', we have that $\theta'|_{cv(\mathcal{C}_A)} = \theta$.
3. We have $A'_s(\theta'\sigma'_A)^\star = P_s\{^y/_x\}(\theta\sigma_A)^\star[y \mapsto M] = P\{^M/_x\} = A'$ since $\mathrm{dom}(\sigma_A) = \mathrm{dom}(\sigma'_A) = \emptyset$, i.e., $(A'\ ;\ \mathsf{N}')$ is the θ'-concretization of $(A'_s\ ;\ \mathcal{C}'_A\ ;\ \mathsf{N}'_s)$, and
4. $\alpha_s\theta' = \mathsf{in}(u,y)\theta' = \mathsf{in}(u\theta', y\theta') = \mathsf{in}(a,M) = \alpha$.

We can deal with the rules OUT-CH$_i$ and OUT-T$_i$ in a rather similar way.

We now consider the inductive cases.

Case OPEN-CH$_i$: In such a case, we have that the tree witnessing the fact that $(A\ ;\ \mathsf{N}) \xrightarrow{\alpha}_i (A'\ ;\ \mathsf{N}')$ ends with an application of the following inference rule

$$\frac{(B\ ;\ \mathsf{N}'') \xrightarrow{out(a,c)}_i (B'\ ;\ \mathsf{N}''') \quad c \neq a,\ \mathsf{N}''(d) = \mathsf{n} \text{ and } d \in \mathcal{N}_{ch}}{(\nu c.B\ ;\ \mathsf{N}) \xrightarrow{\nu d.out(a,d)}_i (B'\{^d/_c\}\ ;\ \mathsf{N}')}$$

Since $(\nu c.B \ ; \ \mathsf{N})$ is the θ_A-concretization of $(A_s \ ; \ \mathcal{C}_A \ ; \ \mathsf{N}_s)$ we have that $A_s(\theta_A\sigma_A)^\star = \nu c.B$. Hence, we know that $A_s = \nu c.B_s$ for some process B_s such that $B_s(\theta_A\sigma_A)^\star = B$. Since $(\nu c.B_s \ ; \mathcal{C}_A \ ; \mathsf{N}_s)$ is well-formed, we know that $\mathcal{C}_A = \nu c.\mathcal{C}_B$ for some well-formed constraint system $\mathcal{C}_B$. Let N''_s be the symbolic naming environment such that

$$\mathsf{N}''_s(u) = \begin{cases} \mathsf{N}''(u) & \text{if } u \in \mathcal{N} \cup \mathcal{X} \\ \mathsf{N}_s(u) & \text{if } u \in \mathcal{Y}. \end{cases}$$

Firstly, we have that $(B_s \ ; \mathcal{C}_B \ ; \mathsf{N}''_s)$ is well-formed. We have also that $\theta_A \in Sol_{\mathsf{E}}(\mathcal{C}_B \ ; \mathsf{N}''_s)$. We can apply our induction hypothesis on $(B_s \ ; \ \mathcal{C}_B \ ; \ \mathsf{N}''_s)$, θ_A, $(B \ ; \mathsf{N}'') \xrightarrow{out(a,c)}_i (B' \ ; \mathsf{N}''')$. We deduce that there exist a well-formed symbolic process $(B'_s \ ; \mathcal{C}'_B \ ; \mathsf{N}'''_s)$, a substitution θ'_B and a label α^B_s such that

1. $(B_s \ ; \mathcal{C}_B \ ; \mathsf{N}''_s) \xrightarrow{\alpha^B_s}_s (B'_s \ ; \mathcal{C}'_B \ ; \mathsf{N}'''_s)$ and $\mathsf{N}''' = \mathsf{N}'''_s[\mathsf{N}'''^{-1}_s(\mathsf{c}) \mapsto \mathsf{b}]$,
2. $\theta'_B \in Sol_{\mathsf{E}}(\mathcal{C}'_B \ ; \mathsf{N}'''_s)$ and $\theta'_B|_{cv(\mathcal{C}_B)} = \theta_A$ and $\theta'_B|_{cv(\mathcal{C}_B)} = \theta'_B$ since constraint variables increase only after an input action.
3. $B'_s(\theta_A\sigma_A)^\star = B'$, i.e., $(B' \ ; \mathsf{N}''')$ is the θ'_B-concretization of $(B'_s \ ; \mathcal{C}'_B \ ; \mathsf{N}'''_s)$ and
4. $\alpha^B_s\theta_A = out(a,c)$. Note also that since $c \notin names(\mathrm{img}(\theta_A))$, we have that $\alpha^B_s = out(u,c)$ for some metavariable u such that $u\theta_A = a$.

Let $A'_s = B'_s\{^d/_c\}$, $\mathcal{C}'_A = \nu d.(\mathcal{C}'_B\{^d/_c\})$, $\mathsf{N}'_s = \mathsf{N}'''_s[c \mapsto \mathsf{b}, d \mapsto \mathsf{f}]$. Let $\theta'_A = \theta'_B = \theta_A$ and $\alpha_s = \nu d.out(u,d)$. We have that

1. $(A_s \ ; \mathcal{C}_A \ ; \mathsf{N}_s) \xrightarrow{\alpha_s}_s (A'_s \ ; \mathcal{C}'_A \ ; \mathsf{N}'_s)$. Indeed, we have that

$$\frac{(B_s \ ; \mathcal{C}_B \ ; \mathsf{N}''_s) \xrightarrow{out(u,c)}_s (B'_s \ ; \mathcal{C}'_B \ ; \mathsf{N}'''_s)}{(\nu c.B_s \ ; \nu c.\mathcal{C}_B \ ; \mathsf{N}_s) \xrightarrow{\nu d.out(u,d)}_s (B'_s\{^d/_c\} \ ; \nu d.(\mathcal{C}'_B\{^d/_c\}) \ ; \mathsf{N}'_s)}$$

2. $\theta'_A \in Sol_{\mathsf{E}}(\mathcal{C}'_A \ ; \mathsf{N}'_s)$ since $\theta'_B \in Sol_{\mathsf{E}}(\mathcal{C}'_B \ ; \mathsf{N}'''_s)$ and $c, d \notin names(\mathrm{img}(\theta'_B))$. We have also that $\theta'_A|_{cv(\mathcal{C}_A)} = \theta'_B|_{cv(\mathcal{C}_B)} = \theta_A$,
3. We have that $A'_s(\theta'_A\sigma'_A)^\star = (B'_s\{^d/_c\})(\theta_A\sigma'_A)^\star = (B'_s\{^d/_c\})(\theta_A(\sigma_A\{^d/_c\}))^\star = B'_s(\theta_A\sigma_A)^\star\{^d/_c\} = B'\{^d/_c\} = A'$, i.e., $(A' \ ; \mathsf{N}')$ is the θ'_A-concretization of $(A' \ ; \mathcal{C}'_A \ ; \mathsf{N}'_s)$,
4. $\alpha_s\theta'_A = (\nu d.out(u,d))\theta_A = \nu d.out(a,d) = \alpha$.

Case Scope$_i$: In such a case, we have that the proof tree witnessing the fact that $(A \ ; \mathsf{N}) \xrightarrow{\alpha} (A' \ ; \mathsf{N}')$ ends with an application of the following inference rule

$$\frac{(B \ ; \mathsf{N}'') \ \xrightarrow{\alpha}_i \ (B' \ ; \mathsf{N}''')}{(\nu n.B \ ; \mathsf{N}) \ \xrightarrow{\alpha}_i \ (\nu n.B' \ ; \mathsf{N}')} \qquad \text{with } n \text{ does not occur in } \alpha$$

Hence, we know that there exist a name n, a label α such that n does not occur in α and two intermediate extended processes $(B \,;\, \mathsf{N}'')$ and $(B' \,;\, \mathsf{N}''')$ such that $A = \nu n.B$, $A' = \nu n.B'$ and $(B \,;\, \mathsf{N}'') \xrightarrow{\alpha} (B' \,;\, \mathsf{N}''')$. Since $(\nu n.B \,;\, \mathsf{N})$ is the θ_A-concretization of $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ we have that $A_s(\theta_A\sigma_A)^\star = \nu n.B$. Hence, we know that $A_s = \nu n.B_s$ for some process B_s such that $B_s(\theta_A\sigma_A)^\star = B$. Since $(\nu n.B_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ is well-formed, we know that $\mathcal{C}_A = \nu n.\mathcal{C}_B$ for some well-formed constraint system $\mathcal{C}_B$. Let N''_s be the symbolic naming environment such that

$$\mathsf{N}''_s(u) = \begin{cases} \mathsf{N}''(u) & \text{if } u \in \mathcal{N} \cup \mathcal{X} \\ \mathsf{N}_s(u) & \text{if } u \in \mathcal{Y}. \end{cases}$$

Firstly, we have $(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}''_s)$ is well-formed. By Lemma B.1 we have $\theta_A \in Sol_{\mathsf{E}}(\mathcal{C}_B \,;\, \mathsf{N}''_s)$. We apply our induction hypothesis on $(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}''_s)$, θ_A, $(B \,;\, \mathsf{N}'') \xrightarrow{\alpha} (B' \,;\, \mathsf{N}''')$. We deduce that there exist a well-formed symbolic process $(B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'''_s)$, a substitution θ'_B and a label α^B_s such that:

1. $(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}''_s) \xrightarrow{\alpha^B_s}_s (B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'''_s)$ and $\mathsf{N}''' = \mathsf{N}'''_s|_{\mathcal{N}\cup\mathcal{X}}$.
2. $\theta'_B \in Sol_{\mathsf{E}}(\mathcal{C}'_B \,;\, \mathsf{N}'''_s)$ and $\theta'_B|_{cv(\mathcal{C}_B)} = \theta_A$,
3. $B'_s(\theta'_B\sigma'_B)^\star = B'$, i.e., $(B' \,;\, \mathsf{N}''')$ is the θ'_B-concretization of $(B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'''_s)$,
4. $\alpha^B_s\theta'_B = \alpha$.

Let $A'_s = \nu n.B'_s$, $\mathcal{C}'_A = \nu n.\mathcal{C}'_B$, $\mathsf{N}'_s = \mathsf{N}'''_s[n \mapsto \mathsf{b}]$. Let $\theta'_A = \theta'_B$ and $\alpha_s = \alpha^B_s$. Note that the symbolic process $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s)$ is well-formed. Moreover, we have

1. $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s) \xrightarrow{\alpha_s} (A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s)$. Indeed, we have that

$$\frac{(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}''_s) \xrightarrow{\alpha_s}_s (B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'''_s)}{(\nu n.B_s \,;\, \nu n.\mathcal{C}_B \,;\, \mathsf{N}_s) \xrightarrow{\alpha_s}_s (\nu n.B'_s \,;\, \nu n.\mathcal{C}'_B \,;\, \mathsf{N}'_s)}$$

2. $\theta'_A \in Sol_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}'_s)$ by Lemma B.1 since $\theta'_B \in Sol_{\mathsf{E}}(\mathcal{C}'_B \,;\, \mathsf{N}'''_s)$ and $n \notin \mathit{names}(\mathrm{img}(\theta'_B))$. We have also that $\theta'_A|_{cv(\mathcal{C}_A)} = \theta'_B|_{cv(\mathcal{C}_B)} = \theta_A$,
3. We have that $A'_s(\theta'_A\sigma'_A)^\star = (\nu n.B'_s)(\theta'_B\sigma'_B)^\star = \nu n.B' = A'$, i.e., $(A' \,;\, \mathsf{N}')$ is the θ'_A-concretization of $(A'_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}'_s)$,
4. $\alpha_s\theta'_A = (\alpha^B_s)\theta'_B = \alpha$.

Case $\textsc{Par}_i$: In such a case, we have that the proof tree witnessing the fact that $(A \,;\, \mathsf{N}) \xrightarrow{\alpha}_i (A' \,;\, \mathsf{N}')$ ends with an application of the following inference rule.

$$\frac{(B \,;\, \mathsf{N}) \xrightarrow{\alpha\psi(D)}_i (B' \,;\, \mathsf{N}')}{(B \mid D \,;\, \mathsf{N}) \xrightarrow{\alpha}_i (B' \mid D \,;\, \mathsf{N}')}$$

Since $(A_s \,;\, \mathsf{N})$ is the θ_A-concretization of $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ we have that $A = B \mid D = A_s(\theta_A\sigma_A)^\star$. Hence, we know that

- $A_s = B_s \mid D_s$ for some processes B_s and D_s such that
- $B_s(\theta_A\sigma_A)^\star = B$ and $D_s(\theta_A\sigma_A)^\star = D$.

Since $(B_s \mid D_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s)$ is well-formed, we deduce that $\mathcal{C}_A = \mathcal{C}_B \mid \psi(D_s)$ for some well-formed constraint system $\mathcal{C}_B$. We have that $(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s)$ is well-formed. Let $\theta_B = (\theta_A\psi(D_s))^\star$. By Lemma B.1 we have that $\theta_B \in Sol_{\mathsf{E}}(\mathcal{C}_B \,;\, \mathsf{N}_s)$. We can apply our induction hypothesis. We deduce that there exists a well-formed symbolic process $(B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)$, a substitution θ'_B and a label α_s^B such that:

1. $(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s) \xrightarrow{\alpha_s^B}_s (B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)$ and
$$\mathsf{N}'_s(u) = \begin{cases} \mathsf{N}'(u) & \text{if } u \in \mathcal{N} \cup \mathcal{X} \\ \mathsf{N}_s(u) & \text{if } u \in \mathcal{Y}. \end{cases}$$
2. $\theta'_B \in Sol_{\mathsf{E}}(\mathcal{C}'_B \,;\, \mathsf{N}'_s)$ and $\theta'_B|_{cv(\mathcal{C}_B)} = \theta_B$,
3. $B'_s(\theta'_B\sigma'_B)^\star = B'$, i.e., $(B' \,;\, \mathsf{N}')$ is the θ'_B-concretization of $(B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)$,
4. $\alpha_s^B\theta'_B = \alpha\psi(D)$.

Let $A'_s = B'_s \mid D_s$, $\mathcal{C}'_A = \mathcal{C}'_B \mid \psi(D_s)$. To define θ'_A, we distinguish two cases.

1. Either α is of the form $in(c, M)$ and $\alpha_s^B = in(u, y)$ for some metavariable u and some variable y with $\mathsf{N}_s(y) = \mathsf{n}$ such that $u\theta_A = u\theta_B = c$. In such a case, let $\theta'_A = \theta_A \cup \{y \mapsto M\}$. Moreover, as $\theta_B = (\theta_A\psi(D_s))^\star$, $\theta'_B|_{cv(\mathcal{C}_B)} = \theta_B$, $\alpha_s^B\theta'_B = \alpha\psi(D)$ and $D_s(\theta_A\sigma_A)^\star = D$ we have that $\theta'_B = (\theta'_A\psi(D_s))^\star$.
2. Otherwise, $\theta'_A = \theta_A$. Moreover in this case we have that $\theta'_B = \theta_B = (\theta_A\psi(D_s))^\star = (\theta'_A\psi(D_s))^\star$.

Let $\alpha_s = \alpha_s^B$. Note that the symbolic process $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s)$ is well-formed. Moreover, we have

1. $(A_s \,;\, \mathcal{C}_A \,;\, \mathsf{N}_s) \xrightarrow{\alpha_s} (A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s)$. Indeed, we have that
$$\frac{(B_s \,;\, \mathcal{C}_B \,;\, \mathsf{N}_s) \xrightarrow{\alpha_s}_s (B'_s \,;\, \mathcal{C}'_B \,;\, \mathsf{N}'_s)}{(B_s \mid D_s \,;\, \mathcal{C}_B \mid \psi(D_s) \,;\, \mathsf{N}_s) \xrightarrow{\alpha_s}_s (B'_s \mid D_s \,;\, \mathcal{C}'_B \mid \psi(D_s) \,;\, \mathsf{N}'_s)}$$
2. We have to show that $\theta'_A \in Sol_{\mathsf{E}}(\mathcal{C}'_A \,;\, \mathsf{N}'_s)$. As $\theta'_B = (\theta'_A\psi(D_s))^\star \in Sol(\mathcal{C}'_B \,;\, \mathsf{N}'_s)$ we have by Lemma B.1 that $\theta'_A \in Sol_{\mathsf{E}}(\mathcal{C}'_B \mid \psi(D_s) \,;\, \mathsf{N}'_s)$. It is clear that we have also $\theta'_A|_{cv(\mathcal{C}_A)} = \theta_A$.
3. We have that $A'_s(\theta'_A\sigma'_A)^\star = (B'_s \mid D_s)(\theta'_A\sigma'_A)^\star = B'_s(\theta'_A\sigma'_A)^\star \mid D_s(\theta'_A\sigma'_A)^\star = B'_s(\theta'_B\sigma'_B)^\star \mid D_s(\theta_A\sigma_A)^\star = B' \mid D = A'$, i.e., $(B' \mid D \,;\, \mathsf{N}')$ is a θ'_A-concretization of $(A'_s \,;\, \mathcal{C}'_A \,;\, \mathsf{N}'_s)$,

4. In the case where $\alpha = in(c, M)$, we have that $\alpha_s\theta'_A = in(u, y)\theta'_A = in(c, M) = \alpha$. Otherwise, the equality holds since $\psi(D)$ and $\psi(D_s)$ do not affect variables which occurs in a label since those variables are of type channel.

Lastly, we can deal with the rule $\textsc{Struct}_i$ by relying on our Proposition B.4. This allows us to conclude. □

Automating security analysis: symbolic equivalence of constraint systems *

Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune

LSV, ENS Cachan & CNRS & INRIA Saclay Île-de-France

Abstract. We consider security properties of cryptographic protocols, that are either trace properties (such as confidentiality or authenticity) or equivalence properties (such as anonymity or strong secrecy).
Infinite sets of possible traces are symbolically represented using *deducibility constraints*. We give a new algorithm that decides the trace equivalence for the traces that are represented using such constraints, in the case of signatures, symmetric and asymmetric encryptions. Our algorithm is implemented and performs well on typical benchmarks. This is the first implemented algorithm, deciding symbolic trace equivalence.

1 Introduction

Security protocols are small distributed programs aiming at some security goal, though relying on untrusted communication media. Formally proving that such a protocol satisfies a security property (or finding an attack) is an important issue, in view of the economical and social impact of a failure.

Starting in the 90s, several models and automated verification tools have been designed. For instance both protocols, intruder capabilities and security properties can be formalized within first-order logic and dedicated resolution strategies yield relevant verification methods [18, 21, 6]. Another approach, initiated in [19], consists in symbolically representing the traces using deducibility constraints. Both approaches were quite successful in finding attacks/proving security protocols. There are however open issues, that concern the extensions of the methods to larger classes of protocols/properties [11]. For instance, most efforts and successes only concerned, until recently, *trace properties*, i.e., security properties that can be checked on each individual sequence of messages corresponding to an execution of the protocol. A typical example of a trace property is the *confidentiality*, also called *weak secrecy*: a given message m should not be deducible from any sequence of messages, that corresponds to an execution of the protocol. Agreement properties, also called *authenticity properties*, are other examples of trace properties.

There are however security properties that cannot be stated as properties of a single trace. Consider for instance a voter casting her vote, encrypted with a public key of a server. Since there are only a fixed, known, number of possible

* This work has been partially supported by the ANR project SeSur AVOTÉ.

plaintexts, the confidentiality is not an issue. A more relevant property is the ability to relate the voter's identity with the plaintext of the message. This is a property in the family of *privacy* (or *anonymity*) properties [15]. Another example is the *strong secrecy*: m is strongly secret if replacing m with any m' in the protocol, would yield another protocol that is indistinguishable from the first one: not only m itself cannot be deduced, but the protocol also does not leak any piece of m. These two examples are not trace properties, but *equivalence properties*: they can be stated as the indistinguishability of two processes. In the present paper, we are interested in automating the proofs of equivalence properties. As far as we know, there are only three series of works that consider the automation of equivalence properties for security protocols[1].

The first one [7] is an extension of the first-order logic encoding of the protocols and security properties. The idea is to overlap the two processes that are supposedly equivalent, forming a *bi-process*, then formalize in first-order logic the simultaneous moves (the single move of the bi-process) upon reception of a message. This method checks a stronger equivalence than observational equivalence, hence it fails on some simple (cook up) examples of processes that are equivalent, but their overlapping cannot be simulated by the moves of a single bi-process. The procedure might also not terminate or produce false attacks, but considers an unbounded number of protocol instances.

The second one [3] (and [14]) assumes a fixed (bounded) number of sessions. Because of the infinite number of possible messages forged by an attacker, the number of possible traces is still infinite. The possible traces of the two processes are symbolically represented by two deducibility constraints. Then [3] provides with a decision procedure, roughly checking that the solutions, *and the recipes that yield the solutions* are identical for both constraints. This forces to compute the solutions and the associated recipes and yields an unpractical algorithm.

The third one [17, 9] is based on an extension of the small attack property of [20]. They show that, if two processes are not equivalent, then there must exist a small witness of non-equivalence. A decision of equivalence can be derived by checking every possible small witness. As in the previous method, the main problem is the practicality. The number of small witnesses is very large as all terms of size smaller than a given bound have to be considered. Consequently, neither this method nor the previous one have been implemented.

We propose in this paper another algorithm for deciding equivalence properties. As in [3, 9], we consider *trace equivalence*, which coincides with observational equivalence for determinate processes [14]. In that case, the equivalence problem can be reduced to the symbolic equivalence of finitely many pairs of deducibility constraints, each of which represents a set of traces (see [14]). We consider signatures, pairing, symmetric and asymmetric encryptions, which is slightly less general than [3, 9], who consider arbitrary subterm-convergent theories. The main idea of our method is to simultaneously solve pairs of constraints, instead of solving each constraint separately and comparing the solutions, as

[1] [16] gives a logical characterization of the equivalence properties. It is not clear if this can be of any help in deriving automated decision procedures.

in [3]. These pairs are successively split into several pairs of systems, while preserving the symbolic equivalence: roughly, the father pair is in the relation if, and only if, all the sons pairs are in the relation. This is not fully correct, since, for termination purposes, we need to keep track of some earlier splitting, using additional predicates. Such predicates, together with the constraint systems, yield another notion of equivalence, which is preserved upwards, while the former is preserved downwards. When a pair of constraints cannot be split any more, then the equivalence can be trivially checked.

A preliminary version of the algorithm has been implemented and works well (within a few seconds) on all benchmarks. The same implementation can also be used for checking the static equivalence and for checking the constraints satisfiability. We also believe that it is easier (w.r.t. [3, 9]) to extend the algorithm to a more general class of processes (including disequality tests for instance) and to avoid the detour through trace equivalence. This is needed to go beyond the class of determinate processes.

We first state precisely the problem in Section 2, then we give the algorithm, actually the transformation rules, in Section 3. We sketch the correctness and termination proofs in Section 4 and provide with a short summary of the experiments in Section 5. Detailed proofs of the results can be found in [8].

2 Equivalence properties and deducibility constraints

We use the following straightfoward example for illustrating some definitions:

Example 1. Consider the following very simple handshake protocol:

$$A \to B : \mathsf{enc}(N_A, K_{AB})$$
$$B \to A : \mathsf{enc}(f(N_A), K_{AB})$$

The agent A sends a random message N_A to B, encrypted with a key K_{AB}, that is shared by A and B only. The agent B replies by sending $f(N_A)$ encrypted with the same key. The function f is any function, for instance a hash function.

Consider only one session of this protocol: a sends $\mathsf{enc}(n_a, k_{ab})$ and waits for $\mathsf{enc}(f(n_a), k_{ab})$. The agent b is expecting a message of the form $\mathsf{enc}(x, k_{ab})$. The variable x represents the fact that b does not know in advance what is this randomly generated message. Then he replies by sending out $\mathsf{enc}(f(x\sigma), k_{ab})$. All possible executions are obtained by replacing x with any message $x\sigma$ such that the attacker can supply with $\mathsf{enc}(x\sigma, k_{ab})$ and then with $\mathsf{enc}(f(n_a), k_{ab})$. This is represented by the following constraint:

$$C := \begin{cases} a,\ b,\ \mathsf{enc}(n_a, k_{ab}) \stackrel{?}{\vdash} \mathsf{enc}(x, k_{ab}) \\ a,\ b,\ \mathsf{enc}(n_a, k_{ab}),\ \mathsf{enc}(f(x), k_{ab}) \stackrel{?}{\vdash} \mathsf{enc}(f(n_a), k_{ab}) \end{cases}$$

Actually, C has only one solution: x has to be replaced by n_a. There is no other way for the attacker to forge a message of the form $\mathsf{enc}(x, k_{ab})$.

2.1 Function symbols and terms

We will use the set of function symbols $\mathcal{F} = \mathcal{N} \cup \mathcal{C} \cup \mathcal{D}$ where:

- $\mathcal{C} = \{\mathsf{enc},\ \mathsf{aenc},\ \mathsf{pub},\ \mathsf{sign},\ \mathsf{vk},\ \langle\ \rangle\}$ is the set of *constructors*;
- $\mathcal{D} = \{\mathsf{dec},\ \mathsf{adec},\ \mathsf{check},\ \mathsf{proj}_1,\ \mathsf{proj}_2\}$ is the set of *destructors*;
- $\mathcal{N}$ is a set of constants, called *names*.

In addition, $\mathcal{X}$ is a set of variables x, y, z,... The *constructor terms* (resp. *ground constructor terms*) are built on $\mathcal{C}$, $\mathcal{N}$ and $\mathcal{X}$ (resp. $\mathcal{C}, \mathcal{N}$). The term rewriting system below is convergent: we let $t{\downarrow}$ be the normal form of t.

$$\mathsf{adec}(\mathsf{aenc}(x, \mathsf{pub}(y)), y) \to x \qquad \mathsf{proj}_1(\langle x, y\rangle) \to x \qquad \mathsf{dec}(\mathsf{enc}(x, y), y) \to x$$
$$\mathsf{check}(\mathsf{sign}(x, y), \mathsf{vk}(y)) \to x \qquad \mathsf{proj}_2(\langle x, y\rangle) \to y$$

A (ground) *recipe* records the attacker's computation. It is used as a witness of how some deduction has been performed. Formally, it is a term built on $\mathcal{C}, \mathcal{D}$ and a set of special variables $\mathcal{AX} = \{ax_1, \ldots, ax_n, \ldots\}$, that can be seen as pointers to the hypotheses, or known messages. Names are excluded from recipes: names that are known to the attacker must be given explicitly as hypotheses.

Example 2. Given $\mathsf{enc}(a, b)$ and b, the recipe $\zeta = \mathsf{dec}(ax_1, ax_2)$ is a witness of how to deduce a: $\zeta\{ax_1 \mapsto \mathsf{enc}(a, b); ax_2 \mapsto b\}{\downarrow} = a$.

The recipes are generalized, including possibly variables that range over recipes: (general) recipes are terms built on $\mathcal{C}, \mathcal{D}, \mathcal{AX}$ and $\mathcal{X}_r$, a set of recipe variables, that are written using capital letters $X, X_1, X_2, \ldots$.

We denote by $var(u)$ is the set of variables of any kind that occur in u.

2.2 Frames

The *frame* records the messages that have been sent by the participants of the protocol; it is a symbolic representation of a set of sequences of messages. The frame is also extended to record some additional informations on attacker's deductions. Typically $\mathsf{dec}(X, \zeta), i \triangleright u$ records that, using a decryption with the recipe ζ, on top of a recipe X, allows to get u (at stage i). After recording this information in the frame, we may forbid the attacker to use a decryption on top of X, forcing him to use this "direct access" from the frame.

Definition 1. *A* frame ϕ *is a sequence* $\zeta_1, i_1 \triangleright u_1, \ldots, \zeta_n, i_n \triangleright u_n$ *where* $u_1, \ldots, u_n$ *are constructor terms,* $i_1, \ldots, i_n \in \mathbb{N}$*, and* $\zeta_1, \ldots, \zeta_n$ *are general recipes. The* domain *of the frame* ϕ*, denoted* $dom(\phi)$*, is the set* $\{\zeta_1, \ldots, \zeta_n\} \cap \mathcal{AX}$*. It must be equal to* $\{ax_1, \ldots, ax_m\}$ *for some* m *that is called the* size *of* ϕ*. A frame is* closed *when* $u_1, \ldots, u_n$ *are ground terms and* $\zeta_1, \ldots, \zeta_n$ *are ground recipes.*

Example 3. The messages of Example 1 are recorded in a frame of size 4.

$$\{ax_1, 1 \triangleright a,\ ax_2, 2 \triangleright b,\ ax_3, 3 \triangleright \mathsf{enc}(n_a, k_{ab}),\ ax_4, 4 \triangleright \mathsf{enc}(f(x), k_{ab})\}.$$

A frame ϕ defines a substitution $\{ax \mapsto u \mid ax \in dom(\phi), ax \triangleright u \in \phi\}$. A closed frame is *consistent* if, for every $\zeta \triangleright u \in \phi$, we have that $\zeta\phi{\downarrow} = u$.

2.3 Deducibility constraints

The following definitions are consistent with [12]. We generalize however the usual definition, including equations between recipes, for example, in order to keep track of some choices in our algorithm.

Definition 2. *A* deducibility constraint *(sometimes called simply* constraint *in what follows) is either* $\bot$ *or consists of:*

1. *a subset* S *of* $\mathcal{X}$ *(the free variables of the constraint);*
2. *a frame* ϕ*, whose size is some* m*;*
3. *a sequence* $X_1, i_1 \overset{?}{\vdash} u_1;\ \ldots;\ X_n, i_n \overset{?}{\vdash} u_n$ *where*
   - $X_1, \ldots, X_n$ *are distinct variables in* $\mathcal{X}_r$*,* $u_1, \ldots, u_n$ *are constructor terms, and* $0 \leq i_1 \leq \ldots \leq i_n \leq m$*.*
   - *for every* $0 \leq k \leq m$*,* $var(ax_k\phi) \subseteq \bigcup_{i_j<k} var(u_j)$*;*
4. *a conjunction* E *of equations and disequations between terms;*
5. *a conjunction* E' *of equations and disequations between recipes.*

The variables X_i represent the recipes that might be used to deduce the right hand side of the deducibility constraint. The indices indicate which initial segment of the frame can be used. We use this indirect representation, instead of the seemingly simpler notation of Example 1, because the transformation rules that will change the frame don't need then to be reproduced on all relevant left sides of deducibility constraints.

Example 4. Back to Example 1, the deducibility constraint is formally given by $S = \{x, y\}$, $E = E' = \emptyset$, the frame ϕ as in Example 3 and the sequence:

$$D \ = \ X_1, 3 \overset{?}{\vdash} \mathsf{enc}(x, k_{ab});\ \ X_2, 4 \overset{?}{\vdash} \mathsf{enc}(f(n_a), k_{ab}).$$

For sake of simplicity, in what follows, we will forget about the first component (the free variables). This is justified by an invariant of our transformation rules: initially all variables are free and each time new variables are introduced, their assignment is determined by an assignment of the free variables.

Definition 3. *A* solution *of a deducibility constraint* $C = (\phi, D, E, E')$ *consists of a mapping* σ *from variables to ground constructor terms and a substitution* θ *mapping* $\mathcal{X}_r$ *to ground recipes, such that:*

- *for every* $\zeta, i \rhd u \in \phi$*,* $var(\zeta\theta) \subseteq \{ax_1, \ldots, ax_i\}$ *and* $\zeta\theta(\phi\sigma)\downarrow = u\sigma\downarrow$ *(i.e. the frame is consistent after instanciating the variables);*
- *for every* $X_i, j \overset{?}{\vdash} u_i$ *in* D*,* $var(X_i\theta) \subseteq \{ax_1, \ldots, ax_j\}$ *and* $X_i\theta(\phi\sigma)\downarrow = u_i\sigma\downarrow$*;*
- *for every equation* $u \overset{?}{=} v$ *(resp.* $u \overset{?}{\neq} v$*) in* E*,* $u\sigma\downarrow = v\sigma\downarrow$ *(resp.* $u\sigma\downarrow \neq v\sigma\downarrow$*);*
- *for every equation* $\zeta \overset{?}{=} \zeta'$ *(resp.* $\zeta \overset{?}{\neq} \zeta'$*) in* E'*,* $\zeta\theta = \zeta'\theta$ *(resp.* $\zeta\theta \neq \zeta'\theta$*).*

$\mathsf{Sol}(C)$ *is the set of solutions of* C*. By convention,* $\mathsf{Sol}(\bot) = \emptyset$*.*

Example 5. Coming back to Example 4, a solution is (σ, θ) with:

- $\sigma = \{x \mapsto n_a,\ y \mapsto \langle a, \mathsf{enc}(n_a, k_{ab})\rangle\}$, and
- $\theta = \{X_1 \mapsto ax_3,\ X_2 \mapsto ax_4,\ X_3 \mapsto \langle ax_1, ax_3\rangle\}$.

Each solution of a constraint corresponds to a possible execution of the protocol, together with the attacker's actions that yield this execution. For instance an attack on the confidentiality of a term s can be modeled by adding $X, m \overset{?}{\vdash} s$ to the constraint system (X is a fresh variable and m is the size of the frame). This represents the derivability of s from the messages sent so far. Note that there might be several attacker's recipes yielding the same trace.

Example 6. Consider another very simple example: the Encrypted Password Transmission protocol [13], which is informally described by the rules:

$$A \to B : \langle N_A, \mathsf{pub}(K_A)\rangle$$
$$B \to A : \mathsf{aenc}(\langle N_A, P\rangle, \mathsf{pub}(K_A))$$

Assume that a first sends a message whereas b is waiting for a message of the form $\langle x, \mathsf{pub}(k_a)\rangle$. Then b responds by sending $\mathsf{aenc}(\langle x, p\rangle, \mathsf{pub}(k_a))$. The corresponding deducibility constraint is (S, ϕ, D, E, E') where $S = \{x, y\}$, $E = E' = \emptyset$, and the sequences ϕ and D are as follows:

$$\phi = \begin{cases} ax_1, 1 \rhd \mathsf{pub}(k_a);\ ax_2, 2 \rhd \mathsf{pub}(k_b); \\ ax_3, 3 \rhd \langle n_a, \mathsf{pub}(k_a)\rangle; \\ ax_4, 4 \rhd \mathsf{aenc}(\langle x, p\rangle, \mathsf{pub}(k_a)) \end{cases} \quad D = \begin{cases} X_1, 3 \overset{?}{\vdash} \langle x, \mathsf{pub}(k_a)\rangle \\ X_2, 4 \overset{?}{\vdash} \mathsf{aenc}(\langle n_a, y\rangle, \mathsf{pub}(k_a)) \end{cases}$$

There are several solutions. For instance, the "honest solution" (σ_h, θ_h) is given by $\sigma_h = \{x \mapsto n_a, y \mapsto p\}$ and $\theta_h = \{X_1 \mapsto ax_3, X_2 \mapsto ax_4\}$. Another solution is (σ, θ) where $\sigma = \{x \mapsto \mathsf{pub}(k_a), y \mapsto n_a\}$ and $\theta = \{X_1 \mapsto \langle ax_1, ax_1\rangle, X_2 \mapsto \mathsf{aenc}(\langle \mathsf{proj}_1(ax_3), \mathsf{proj}_1(ax_3)\rangle, ax_1)\}$.

2.4 Static equivalence

Two sequences of terms are *statically equivalent* if, whatever an attacker observes on the first sequence, the same observation holds on the second sequence [2]:

Definition 4. *Two closed frames ϕ and ϕ' having the same size m are* statically equivalent*, which we write $\phi \sim_s \phi'$, if*

1. *for any ground recipe ζ such that $var(\zeta) \subseteq \{ax_1, \ldots, ax_m\}$, we have that*
   $$\zeta\phi{\downarrow} \text{ is a constructor term if, and only if, } \zeta\phi'{\downarrow} \text{ is a constructor term}$$
2. *for any ground recipes ζ, ζ' such that $var(\{\zeta, \zeta'\}) \subseteq \{ax_1, \ldots, ax_m\}$, and the terms $\zeta\phi{\downarrow}$, $\zeta'\phi{\downarrow}$ are constructor terms, we have that*
   $$\zeta\phi{\downarrow} = \zeta'\phi{\downarrow} \text{ if, and only, if } \zeta\phi'{\downarrow} = \zeta'\phi'{\downarrow}.$$

Example 7. Consider the frames $\phi_1 = \{ax_1 \rhd a, ax_2 \rhd \mathsf{enc}(a, b), ax_3 \rhd b\}$ and $\phi_2 = \{ax_1 \rhd a, ax_2 \rhd \mathsf{enc}(c, b), ax_3 \rhd b\}$. $\phi_1 \not\sim_s \phi_2$ since choosing $\zeta = \mathsf{dec}(ax_2, ax_3)$ and $\zeta' = ax_1$ yields $\zeta\phi_1{\downarrow} = \zeta'\phi_1{\downarrow} = a$ while $\zeta\phi_2{\downarrow} \neq \zeta'\phi_2{\downarrow}$.

On the other hand, $\{ax_1 \rhd a, ax_2 \rhd \mathsf{enc}(a, b)\} \sim_s \{ax_1 \rhd a, ax_2 \rhd \mathsf{enc}(c, b)\}$ since, intuitively, there is no way to open the ciphertexts or to construct them, hence no information on the content may leak.

2.5 Symbolic equivalence

Now we wish to check static equivalence on any possible trace. This is captured by the following definition:

Definition 5. *Let C and C' be two constraints whose corresponding frames are ϕ and ϕ'. C is* symbolically equivalent *to C', $C \approx_s C'$, if:*
- for all $(\theta, \sigma) \in \mathsf{Sol}(C)$, there exists σ' such that $(\theta, \sigma') \in \mathsf{Sol}(C')$, and $\phi\sigma \sim_s \phi'\sigma'$,
- for all $(\theta, \sigma') \in \mathsf{Sol}(C')$, there exists σ such that $(\theta, \sigma) \in \mathsf{Sol}(C)$, and $\phi\sigma \sim_s \phi'\sigma'$.

Example 8. As explained for instance in [3], the security of the handshake protocol against offline guessing attacks can be modeled as an equivalence property between two samples of the protocol instance, one in which, at the end of the protocol, the key is revealed and the other in which a random number is revealed instead. This amounts to check the symbolic equivalence of the two constraints:

- $C_1 = (\phi \cup \{ax_5, 5 \triangleright k_{ab}\}, D \cup \{X_3, 5 \overset{?}{\vdash} y\}, \emptyset, \emptyset)$, and
- $C_2 = (\phi \cup \{ax_5, 5 \triangleright k\}, D \cup \{X_3, 5 \overset{?}{\vdash} y\}, \emptyset, \emptyset)$

where D is as in Example 4 and ϕ is as in Example 3.

The constraints C_1 and C_2 are *not* symbolically equivalent: considering the assignment $\sigma = \{x \mapsto n_a, y \mapsto n_a\}$, there is a recipe $X_3\theta = \mathsf{dec}(ax_3, ax_5)$ yielding this solution, while any solution σ' of C_2 maps x to n_a and, if $X_3\theta = \mathsf{dec}(ax_3, ax_5)$, we must have $y\sigma'{\downarrow} = \mathsf{dec}(\mathsf{enc}(n_a, k_{ab}), k)$, which is not possible since this is not a constructor term.

Any trace equivalence problem can be expressed as an instance of the equivalence of an *initial pair of constraints*, that is a pair of the form (ϕ_1, D_1, E_1, E'_1), (ϕ_2, D_2, E_2, E'_2) in which:

- $E'_1 = E'_2 = \emptyset$, and E_1, E_2 only contain equations;
- $\phi_1 = \{ax_1, 1 \triangleright u_1, \ldots, ax_m, m \triangleright u_m\}$, and $D_1 = X_1, i_1 \overset{?}{\vdash} s_1;\ \ldots;\ X_n, i_n \overset{?}{\vdash} s_n$;
- $\phi_2 = \{ax_1, 1 \triangleright v_1, \ldots, ax_m.m \triangleright v_m\}$, and $D_2 = X_1, i_1 \overset{?}{\vdash} t_1;\ \ldots;\ X_n, i_n \overset{?}{\vdash} t_n$.

Or else it is a pair as above, in which one of the components is replaced with $\bot$.

In particular, the number of components in the frame and in the deducibility part are respectively identical in the two constraints, when none of them is $\bot$. *This will be an invariant in all our transformation rules.* Hence we will always assume this without further mention. This is unchanged by the transformations, unless the constraint becomes $\bot$. We keep the notation m for the size of the frames. Finally, the consistency of the frame after instanciation (the first condition of Definition 3) is satisfied for all solutions of initial constraints and is again an invariant, hence we will not care of this condition.

As explained in [14], such initial constraints are sufficient for our applications. The case where one of the component is $\bot$ solves the satisfiability problem for the constraint: the constraint solving procedure of [12] solves this specific instance.

3 Transformation rules

The main result of this paper is a decision procedure for symbolic equivalence of an initial pair of constraints:

Theorem 1. *Given an initial pair (C, C'), it is decidable whether $C \approx_s C'$.*

This result in itself is already known (e.g. [3, 9]), but, as claimed in the introduction, the known algorithms cannot yield any reasonable implementation. We propose here a new algorithm/proof, which is implemented. As pointed in [14], this yields a decision algorithm for the observational equivalence of simple processes without replication nor else branch. The class of simple processes captures most existing protocols.

The decision algorithm works by rewriting pairs of constraints, until a trivial failure or a trivial success is found. These rules are branching: they rewrite a pair of constraints into two pairs of constraints. Transforming the pairs of constraints therefore builds a binary tree. Termination requires to keep track of some information, that is recorded using flags, which we describe first. In Section 4, we show that the tree is then finite: the rules are terminating. The transformation rules are also correct: if all leaves are success leaves, then the original pair of constraints is equivalent. They are finally complete: if the two original constraints are equivalent then any of two pairs of constraints resulting from a rewriting steps are also equivalent.

3.1 Flags

The flags are additional constraints that restrict the recipes. We list them here, together with (a sketch of) their semantics.

Constraints $X, i \overset{?}{\vdash}_F u$ may be indexed with a set F consisting of propositions $\texttt{NoCons}_f$ where f is a constructor. Any solution (θ, σ) such that $X\theta$ is headed with f is then excluded. Expressions $\zeta, j \triangleright_F u$ in a frame are indexed with a set F consisting of:

- $\texttt{NoCons}_f$ (as above) discards the solutions (θ, σ) such that a subterm of a recipe allows to deduce $u\sigma$ using f as a last step.
- $\texttt{NoDest}_f(i)$ where f is a destructor and $i \leq m$ discards the solutions (θ, σ) such that there exists $X, j \overset{?}{\vdash} v$ with $j \leq i$ and $\zeta'_2, \ldots, \zeta'_n$ where $f(\zeta\theta, \zeta'_2, \ldots, \zeta'_n)$ occurs as a subterm in $X\theta$, unless we use a shortcut explicitly given in the frame.
- $\texttt{NoUse}$. The corresponding elements of the frame cannot be used in any recipe, and avoids shifting the indices.

3.2 The rules

The rules are displayed in Figure 1 for single constraints. We explain in Section 3.3 how they are applied to pairs of constraints (an essential feature of our

algorithm). A simple idea would be to guess the top function symbol of a recipe and replace the recipe variable with the corresponding instance. When the head symbol of a recipe is a constructor and the corresponding term is not a variable, this is nice, since the constraint becomes simpler. This is the purpose of the rule Cons. When the top symbol of a recipe is a destructor, the constraint becomes more complex, introducing new terms, which yields non-termination.

Our strategy is different for destructors: we switch roughly from the top position of the recipe to the *redex position*. Typically, in case of symmetric encryption, if a ciphertext is in the frame, we will guess whether the decryption key is deducible, and at which stage.

The Cons rule simply guesses whether the top symbol of the recipe is a constructor f. Either it is, and then we can split the constraint, or it is not and we add a flag forbidding this. The rule Axiom also guesses whether a trivial recipe can be applied. If so, the constraint can simply be removed. Otherwise, it means that the right-hand-side of the deducibility constraint is different from the members of the frame. The Dest rule is more tricky. If v is a non-variable member of the frame, that can be unified with a non variable subterm of a left side of a rewrite rule (for instance v is a ciphertext), we guess whether the rule can be applied to v. This corresponds to the equation $u_1 \stackrel{?}{=} v$, that yields an instance of w, the right member of the rewrite rule, provided that the rest of the left member is also deducible: we get constraints $X_2, i \stackrel{?}{\vdash} u_2; \ldots; X_n, i \stackrel{?}{\vdash} u_n$. The flag `NoDest` is added in any case to the frame, since we either already applied the destructor, and this application result is now recorded in the frame by $f(\zeta, X_2, \ldots, X_n), i \triangleright w$, or else it is assumed that f applied to v will not yield a redex.

The remaining rules cover the comparisons that an attacker could perform at various stages. The equality rules guess equalities between right sides of deducibility constraints and/or members of the frame. If a member of the frame is deducible at an early stage, then this message does not bring any new information to the attacker: it becomes useless, hence the `NoUse` flag.

Finally, the last rule is the only rule that is needed to get in addition a static equivalence decision algorithm, as in [1]. Thanks to this rule, if a subterm of the frame is deducible, then there will be a branch in which it is deduced.

3.3 How to use the transformation rules

In the previous section we gave rules that apply on a single constraint. We explain here how they are extended to pairs of constraints. If one of the constraint is $\bot$, then we proceed as if there was a single constraint. Otherwise, the indices i (resp. i_1, i_2) and the recipes X, ζ (resp. $X_1, X_2, \zeta_1, \zeta_2$) matching the left side of the rules *must be identical in both constraints*: we apply the rules at the same positions in both constraints.

We have to explain now what happens when, on a given pair (C, C') a rule can be applied on C and not on C' (or the converse).

$$\underline{\text{Cons}} : X, i \overset{?}{\vdash}_F f(t_1, \ldots, t_n) \rightarrow \begin{array}{l} X_1, i \overset{?}{\vdash}_F t_1; \cdots; X_n, i \overset{?}{\vdash}_F t_n; X \overset{?}{=} f(X_1, \ldots, X_n) \\ X, i \overset{?}{\vdash}_{F+\texttt{NoCons}_f} f(t_1, \ldots, t_n) \end{array}$$

If $\texttt{NoCons}_f \notin F$ and $X_1, \ldots X_n$ are fresh variables.

$$\underline{\text{Axiom}} : X, i \overset{?}{\vdash}_F v \rightarrow \begin{array}{l} u \overset{?}{=} v;\ X \overset{?}{=} \zeta \\ X, i \overset{?}{\vdash}_F v;\ X \overset{?}{\neq} \zeta \end{array}$$

If $v \notin \mathcal{X}$, ϕ contains $\zeta, j \rhd_G u$ with $\texttt{NoUse} \notin G$, and $i \geq j$.

$$\underline{\text{Dest}} : \zeta, y \rhd_G v \rightarrow \begin{array}{l} X_2, i \overset{?}{\vdash} u_2;\ \cdots;\ X_n, i \overset{?}{\vdash} u_n;\ u_1 \overset{?}{=} v;\ \zeta, j \rhd_{G+\texttt{NoDest}_f(m)} v; \\ \qquad\qquad f(\zeta, X_2, \ldots, X_n), i \rhd w \\ \zeta, j \rhd_{G+\texttt{NoDest}_f(i)} v \end{array}$$

If $v \notin \mathcal{X}$, $\texttt{NoUse} \notin G$, there is a rewrite rule $f(u_1, \ldots, u_n) \to w$, $k < i$ whenever $\texttt{NoDest}_f(k) \in G$ and i is minimal such that $j \leq i$ and there is some constraint $X, i \overset{?}{\vdash} w$ ($i = m$ if there is no such constraint).

$$\underline{\text{Eq-left-left}} : \zeta_1, i_1 \rhd_{F_1} u_1;\ \zeta_2, i_2 \rhd_{F_2} u_2 \rightarrow \begin{array}{l} \zeta_1, i_1 \rhd_{F_1} u_1;\ \zeta_2, i_2 \rhd_{F_2} u_1;\ u_1 \overset{?}{=} u_2 \\ \zeta_1, i_1 \rhd_{F_1} u_1;\ \zeta_2, i_2 \rhd_{F_2} u_2;\ u_1 \overset{?}{\neq} u_2 \end{array}$$

If $\texttt{NoUse} \notin F_1 \cup F_2$ and $i_1 \leq i_2$.

$$\underline{\text{Eq-right-right}} : X_2, i_2 \overset{?}{\vdash} u_2 \rightarrow \begin{array}{l} X_1 = X_2;\ u_1 \overset{?}{=} u_2 \\ X_2, i_2 \overset{?}{\vdash} u_2;\ u_1 \overset{?}{\neq} u_2 \end{array}$$

If $X_1, i_1 \overset{?}{\vdash} u_1;$ and $i_1 \leq i_2$.

$$\underline{\text{Eq-left-right}} : \zeta, j \rhd_G v \rightarrow \begin{array}{l} \zeta, j \rhd_{G+\texttt{NoUse}} u;\ u \overset{?}{=} v \\ \zeta, j \rhd_G v;\ u \overset{?}{\neq} v \end{array}$$

If $X, i \overset{?}{\vdash}_F u;$, $\texttt{NoUse} \notin G$ and $j > i$.

$$\underline{\text{Ded-subterms}} : \zeta, i \rhd_F f(u_1, \ldots, u_n) \rightarrow \begin{array}{l} X_1, m \overset{?}{\vdash} u_1;\ \cdots;\ X_n, m \overset{?}{\vdash} u_n; \\ \qquad\qquad \zeta, i \rhd_{F+\texttt{NoCons}_f} u \\ \zeta, i \rhd_{F+\texttt{NoCons}_f} f(u_1, \ldots, u_n) \end{array}$$

If $\texttt{NoCons}_f, \texttt{NoUse} \notin F$ and $X_1, \ldots, X_n$ are fresh variables.

All rules assume that the equations have a mgu and that this mgu is eagerly applied to the resulting constraint without yielding any trivial disequation.

Fig. 1. Transformation rules

Example 9. Let $C = (\phi, D, E, E')$ and $C' = (\phi, D', E, E')$ where $E = E' = \emptyset$, $\phi = ax_1, 1 \triangleright a$, $D = X, 1 \overset{?}{\vdash} \mathsf{enc}(x_1, x_2)$, and $D' = X, 1 \overset{?}{\vdash} x$. The rule Cons can be applied on C and not on C'. However, we have to consider solutions where $\mathsf{enc}(x_1, x_2)\sigma$ and $x\sigma'$ are both obtained by a construction. Hence, it is important to enable this rule on both sides. For this, we first apply the substitution $x \mapsto \mathsf{enc}(y_1, y_2)$ where y_1, y_2 are fresh variables. This yields the two pairs of constraints (C_1, C_1') and (C_2, C_2') (forgetting about equations):

- $C_1 = (\phi, X_1, 1 \overset{?}{\vdash} x_1; X_2, 1 \overset{?}{\vdash} x_2)$ and $C_1' = (\phi, X_1, 1 \overset{?}{\vdash} y_1;\ X_2, 1 \overset{?}{\vdash} y_2)$;
- $C_2 = (\phi, X, 1 \overset{?}{\vdash}_{\mathtt{NoCons}_{\mathsf{enc}}} \mathsf{enc}(x_1, x_2))$ and $C_2' = (\phi, X, 1 \overset{?}{\vdash}_{\mathtt{NoCons}_{\mathsf{enc}}} x)$.

Therefore, the rule Cons, (this is similar for Ded-subterms), when applied to pairs of constraints comes in three versions: either the rule is applied on both sides or, if $X, i \overset{?}{\vdash} f(t_1, \ldots, t_n)$ (resp. $\zeta \triangleright f(t_1, \ldots, t_n)$) is in C, and $X, i \overset{?}{\vdash} x$ (resp. $\zeta \triangleright x$) is in C', we may apply the rule on the pair of constraints, adding to C' the equation $x \overset{?}{=} f(x_1, \ldots, x_n)$ where $x_1, \ldots, x_n$ are fresh variables. The third version is obtained by switching C and C'. This may introduce new variables, that yield a termination issue, which we discuss in Section 4.1. Similarly, the rules Axiom and Dest assume that $v \notin \mathcal{X}$. This has to be satisfied by C or C'. In case of the rule Dest, this means that the variables of the rewrite rule might not be immediately eliminated: this may also introduce new variables. For the rules Eq-left-left, Eq-right-right and Eq-left-right, we require that at least one new non-trivial equality (or disequality) is added to one of the two constraints (otherwise there is a trivial loop).

For all rules, if a rule is applicable on one constraint and not the other, we do perform the transformation, however replacing a constraint with $\bot$ when a condition becomes false or meaningless. Furthermore, we also replace a constraint C with $\bot$ when:

- the rule Dest cannot be applied on C; and
- C contains a constraint $X, i \overset{?}{\vdash} v$ such that v is not a variable and the rules Cons and Axiom cannot be applied to it.

Altogether this yields a transformation relation $(C, C') \to (C_1, C_1'), (C_2, C_2')$ on pairs of constraints: a node labeled (C, C') has two sons, respectively labeled (C_1, C_1') and (C_2, C_2').

Our algorithm can be stated as follows:

- Construct, from an initial pair of constraints (C_0, C_0') a tree, by applying as long as possible a transformation rule to a leaf of the tree.
- If, at some point, there is a leaf to which no rule is applicable and that is labeled $(C, \bot)$ or $(\bot, C)$ where $C \neq \bot$, then we stop with $C_0 \not\approx_s C_0'$.
- Otherwise, if the construction of the tree stops without reaching such a failure, return $C_0 \approx_s C_0'$.

Our algorithm can also be used to decide static equivalence of frames, as well as the (un)satisfiability of a constraint. Furthermore, in case of failure, a witness of the failure can be returned, using the equations of the non-$\bot$ constraint.

4 Correctness, completeness and termination

4.1 Termination

In general, the rules might not terminate, as shown by the following example:

Example 10. Consider the initial pair of contraints (C, C') given below:

$$C = \begin{cases} a \overset{?}{\vdash} \mathsf{enc}(x_1, x_2) \\ a, b \overset{?}{\vdash} x_1 \end{cases} \qquad C' = \begin{cases} a \overset{?}{\vdash} y_1 \\ a, b \overset{?}{\vdash} \mathsf{enc}(y_1, y_2) \end{cases}$$

We may indeed apply Cons yielding (on one branch):

$$C_1 = \begin{cases} a \overset{?}{\vdash} x_1 \\ a \overset{?}{\vdash} x_2 \\ a, b \overset{?}{\vdash} x_1 \end{cases} \qquad C'_1 = \begin{cases} a \overset{?}{\vdash} z_1 \\ a \overset{?}{\vdash} z_2 \\ a, b \overset{?}{\vdash} \mathsf{enc}(\mathsf{enc}(z_1, z_2), y_2) \end{cases} \quad \text{and } y_1 \overset{?}{=} \mathsf{enc}(z_1, z_2)$$

Then, again using Cons, we get back as a subproblem the original constraints.

Fortunately, there is a simple complete strategy that avoids this behavior, by breaking the symmetry between the two constraints components. We assume in the following that, applying

- Cons to (C, C') where $X, i \overset{?}{\vdash} x \in C$ and $X, i \overset{?}{\vdash} f(t_1, \ldots, t_n) \in C'$,
- Ded-subterms to (C, C') where $\zeta, j \rhd x \in C$ and $\zeta, j \rhd f(t_1, \ldots, t_n) \in C'$,
- Dest to (C, C') where $X, i \overset{?}{\vdash} u; \zeta, j \rhd x \ \in C$ and $X, i \overset{?}{\vdash} u'; \zeta, j \rhd v' \ \in C'$

are only allowed when no other rule can be applied.

There is however no such restriction, when we switch the elements of the pair. If we come back to Example 10, we still apply the same transformation rule to the pair (C, C'), but we cannot apply Cons to (C_1, C'_1) since Eq-right-right can be applied to the constraint C_1, yielding a failure: $C \not\approx_s C'$.

Lemma 1. *With the above strategy, the transformation rules are terminating on any initial pair of constraint systems.*

Idea of the proof: as long as no new first-order variable is introduced, the set of first-order terms appearing in the constraint is roughly bounded by the subterms of the constraint. (This relies on the properties of the rewrite system). Loops are then prevented by the flags. Now, because of the eager application of substitutions, the only cases in which new first-order variables are introduced are the above cases of applications of Cons, Ded-subterms and Dest. Until new variables are introduced in the right constraints, the above argument applies:

the sequence of transformations is finite. Then, according to the strategy, when new variables are introduced on the right constraint, no other rule may apply. This implies that the left constraint (considered in isolation) is irreducible: it is of the form $X_1, i_1 \overset{?}{\vdash} x_1, \ldots, X_n, i_n \overset{?}{\vdash} x_n, \ldots$ where $x_1, \ldots x_n$ are distinct variables (which we call a *solved constraint*). From this point onwards, the rules DEST, DED-SUBTERMS will never be applicable and therefore, no element will be added to the frames. Then, either usable elements of the frames are strictly decreasing (using a EQ-LEFT-RIGHT) or else we preserve the property of being solved on the left. In the latter case, the first termination argument can be applied to the right constraint.

4.2 Correctness

The transformation rules yield a finite tree labeled with pairs of constraints.

Lemma 2. *If all leaves of a tree, whose root is labeled with (C_0, C_0') (a pair of initial constraints), are labeled either with $(\bot, \bot)$ or with some (C, C') with $C \neq \bot, C' \neq \bot$, then $C_0 \approx_s C_0'$.*

The idea of the proof is to first analyse the structure of the leaves. We introduce a restricted symbolic equivalence $\approx_s^r$ such that $C \approx_s^r C'$ for any leaf whose two label components are distinct from $\bot$. Roughly, this restricted equivalence will only consider the recipes that satisfied the additional constraints induced by the flags. Then we show that $\approx_s^r$ is preserved upwards in the tree: for any transformation rule, if the two pairs of constraints labeling the sons of a node are respectively in $\approx_s^r$, then the same property holds for the father. Finally, $\approx_s^r$ coincides with $\approx_s$ on the initial constraints (that contain no flag).

4.3 Completeness

We prove that the symbolic equivalence is preserved by the transformation rules, which yields:

Lemma 3. *If (C_0, C_0') is a pair of initial constraints such that $C_0 \approx_s C_0'$, then all leaves of a tree, whose root is labeled with (C_0, C_0'), are labeled either with $(\bot, \bot)$ or with some (C, C') with $C \neq \bot$ and $C' \neq \bot$.*

5 Implementation and experiments

An Ocaml implementation of an early version of the procedure described in this paper, as well as several examples, are available at `http://www.lsv.ens-cachan.fr/~cheval/programs/index.php` (around 5000 lines of Ocaml). Our implementation closely follows the transformation rules that we described. For efficiency reasons, a strategy on checking the rules applicability has been designed in addition.

We checked the implementation on examples of static equivalence problems, on examples of satisfiability problems, and on symbolic equivalence problems that come from actual protocols. On all examples the tool terminates in less than a second (on a standard laptop). Note that the input of the algorithm is a pair of constraints: checking the equivalence of protocols would require in addition an interleaving step, that could be expensive.

We have run our tool on the following family of examples presented in [5]:

$$\phi_n = \{ax_1 \triangleright t_n^0, ax_2 \triangleright c_0, ax_3 \triangleright c_1\} \text{ and } \phi'_n = \{ax_1 \triangleright t_n^1, ax_2 \triangleright c_0, ax_3 \triangleright c_1\}$$

where $t_0^i = c_i$ and $t_{n+1}^i = \langle \mathsf{enc}(t_n^i, k_n^i), k_n^i \rangle$, $i \in \{0, 1\}$. In these examples, the size of the distinguishing tests increase exponentially while the sizes of the frames grow linearly. As KiSs [10], our tool outperforms YAPA [4] on such examples.

For symbolic equivalences, we cannot compare with other tools (there is no such tools); we simply tested the program on some home made benchmarks as well as on the handshake protocol, several versions of the encrypted password transmission protocol, the encrypted key exchange protocol [13], each for the offline guessing attack property. We checked also the strong secrecy for the corrected Dennin-Sacco key distribution protocol. Unfortunately we cannot (yet) check anonymity properties for e-voting protocols, as we would need to consider more cryptographic primitives.

6 Conclusion

We presented a new algorithm for deciding symbolic equivalence, which performs well in practice. There is still some work to do for extending the results and the tool. First, we use trace equivalence, which requires to consider all interleavings of actions; for each such interleaving, a pair of constraints is generated, which is given to our algorithm. This requires an expensive overhead (which is not implemented), that might be unnecessary. Instead, we wish to extend our algorithm, considering pairs of sets of constraints and use a symbolic bisimulation. This looks feasible and would avoid the detour through trace equivalence. This would also allow drop the determinacy assumption on the protocols and to compare our method with ProVerif [7].

We considered only positive protocols; we wish to extend the algorithm to non-positive protocols, allowing disequality constraints from the start. Finally, we need to extend the method to other cryptographic primitives, typically blind signatures and zero-knowledge proofs.

Acknowledgments. We wish to thank Sergiu Bursuc for fruitful discussions.

References

1. M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 367(1–2):2–32, 2006.

2. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. of 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, 2001.
3. M. Baudet. Deciding security of protocols against off-line guessing attacks. In *Proc. of 12th ACM Conference on Computer and Communications Security*, 2005.
4. M. Baudet. YAPA (Yet Another Protocol Analyzer), 2008. `http://www.lsv.ens-cachan.fr/~baudet/yapa/index.html`.
5. M. Baudet, V. Cortier, and S. Delaune. YAPA: A generic tool for computing intruder knowledge. In *Proc. of 20th International Conference on Rewriting Techniques and Application (RTA'09)*, LNCS, 2009.
6. B. Blanchet. An automatic security protocol verifier based on resolution theorem proving (invited tutorial). In *Proc. of 20th International Conference on Automated Deduction (CADE'05)*, 2005.
7. B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.
8. V. Cheval, H. Comon-Lundh, and S. Delaune. Automating security analysis: symbolic equivalence of constraint systems. Technical report, `http://www.lsv.ens-cachan.fr/~cheval/programs/technical-report.pdf`, 2010.
9. Y. Chevalier and M. Rusinowitch. Decidability of symbolic equivalence of derivations. Unpublished draft, 2009.
10. Ş. Ciobâcă. KiSs, 2009. `http://www.lsv.ens-cachan.fr/~ciobaca/kiss`.
11. H. Comon-Lundh. Challenges in the automated verification of security protocols. In *Proc. of 4th International Joint Conference on Automated Reasoning (IJCAR'08)*, volume 5195 of *LNAI*, pages 396–409, Sydney, Australia, 2008. Springer-Verlag.
12. H. Comon-Lundh, V. Cortier, and E. Zalinescu. Deciding security properties of cryptographic protocols. application to key cycles. *Transaction on Computational Logic*, 11(2), 2010.
13. R. Corin, J. Doumen, and S. Etalle. Analysing password protocol security against off-line dictionary attacks. *Electr. Notes Theor. Comput. Sci.*, 121:47–63, 2005.
14. V. Cortier and S. Delaune. A method for proving observational equivalence. In *Proc. of 22nd Computer Security Foundations Symposium (CSF'09)*, pages 266–276. IEEE Comp. Soc. Press, 2009.
15. S. Delaune, S. Kremer, and M. D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009.
16. U. Fendrup, H. Hüttel, and J. N. Jensen. Modal logics for cryptographic processes. *Theoretical Computer Science*, 68, 2002.
17. H. Huttel. Deciding framed bisimulation. In *4th International Workshop on Verification of Infinite State Systems INFINITY'02*, pages 1–20, 2002.
18. C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.
19. J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. of 8th ACM Conference on Computer and Communications Security*, 2001.
20. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is np-complete. In *Proc. of 14th Computer Security Foundations Workshop*, 2001.
21. C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In *Proc. of 16th Conference on Automated Deduction*, volume 1632, pages 314–328. LNCS, 1999.