PRECISE ANALYSIS OF π-CALCULUS IN CUBIC TIME

L.Colussi, G.Filè and A.Griggio
Department of Pure and Applied Mathematics
University of Padova, Italy

Abstract It is known that a static analysis of π-calculus can be done rather simply and also efficiently, i.e. in $O(n^3)$ time. Clearly, a static analysis should be as precise as possible. We show that it is not only desirable, but also possible to improve the precision of the analysis without worsening its asymptotic complexity. We illustrate the main principles of this efficient algorithm, we prove that it is indeed cubic and we also show that it is correct. The technique introduced here appears to be useful also for other applications, in particular, for the static analysis of languages that extend the π-calculus.

Keywords: static analysis, π-calculus, algorithm complexity

Introduction

The π-calculus [11, 10] is an algebra of processes that models communications among agents that share a common channel. When an input and an output operation synchronize on a common channel, then the bound name of the input gets instantiated to the name sent by the output operation. The algebra also models mobility by allowing the exchange of channel names among agents.

In a real computation of a π-calculus process, the bound name of an input operation can be instantiated at most once. However, when one wants to compute statically all possible behaviours of the process he/she must take into account the fact that an input action can in general synchronize with many output actions (in different real computations) and therefore, a static analysis generally associates a set of names to each input bound name. Thus, in general, a static analysis applied to a process P is *correct* if it computes a function ρ that we call *name-association* such that for each input operation $B = b(v)$ of P, $\rho(v)$ contains all the names that may instantiate v as a result of the synchronization of B with some output operations of P. It is easy to see that a correct static analysis could be designed according to the following scheme:

- First compute all the input/output pairs (A, B) of P that may synchronize;
- For each such pair (A, B), where $A = \overline{a}\langle u\rangle$ and $B = b(v)$, the pair can communicate only when $\rho(a) \cap \rho(b) \neq \emptyset$ and when this condition is satisfied, one accounts for the communication from A to B by adding $\rho(u)$ to $\rho(v)$.

Such an analysis is surely very simple, but it is also bound to be very poor in terms of precision, because in general it considers many synchronizations that cannot take place in real computations. Let us consider this example.

EXAMPLE 1 *In this example we want to model the situation of a client that downloads an applet from a server and when this applet requests a connection to some host with a given IP number, accepts the request only if this IP number meets two conditions: (i) it is the same as that of the server from which the applet was downloaded and, (ii) it is in a* white list *that contains the client's trustworthy servers.*

For simplicity we assume that the client C already shares a channel C_S with the server S and a channel C_A with the applet A. Channel C_W connects C with its white list W. OK is a message that client C sends to A to signal that it accepts its request. The applet A sends to C three IP numbers of hosts to which it wishes to connect. The system is the parallel composition of four processes $C|W|S|A$ where:

$$\begin{array}{rcl} C & = & C_S(x).C_A(y).C_W(z).[x=y].[x=z].\overline{C_A}\langle OK\rangle \\ W & = & \overline{C_W}\langle IP_1\rangle + \overline{C_W}\langle IP_2\rangle \qquad\qquad S \ = !\overline{C_S}\langle IP_S\rangle \\ A & = & (\nu\, M)(\overline{C_A}\langle IP_S\rangle.C_A(x).[x=OK].\overline{IP_S}\langle M\rangle + \overline{C_A}\langle IP_1\rangle.C_A(x). \\ & & [x=OK].\overline{IP_1}\langle M\rangle + \overline{C_A}\langle IP_2\rangle.C_A(x).[x=OK].\overline{IP_2}\langle M\rangle) \end{array}$$

It should be easy to see that, each test of the client C can be satisfied, but that they cannot be satisfied together. Therefore there is no execution of the system in which C sends OK to the applet A. For a static analysis to discover this fact, it is important that the 2 tests are considered together. The analysis presented below does this and therefore it will statically discover this fact.

In what follows we consider an input/output pair (A, B) and we assume that A is the output action $\overline{a}\langle u\rangle$ and B the input action $b(v)$. The above example indicates that it is desirable to have static analyses that consider that pair (A, B) can synchronize only when they really **can** synchronize! More precisely, only when:

(i) there is a real computation in which A and B synchronize and moreover,

(ii) if this is the case, then we would like to model this synchronization by adding to $\rho(v)$ only the name that may instantiate u in the corresponding computation.

These two points cannot be accomplished in general as deciding point (i) is an unsolvable problem. However, during a static analysis, it is possible to use the name-association ρ that is being computed in order to approximate safely these two wishes.

Concerning point (i), we can discover that A and B can never synchronize if they are preceded by a test $[x = y]$ such that $\rho(x) \cap \rho(y) = \emptyset$. By the correctness of ρ this fact clearly implies that the test is never satisfied. As a matter of fact, it is easier to reason in the opposite direction, i.e., to conclude that the test may be satisfied in some real computation only when $\rho(x) \cap \rho(y) \neq \emptyset$. Clearly, if together with $[x = y]$ also the test $[y = z]$ precedes A and B, then both tests must be satisfied together and this is possible only when $\rho(x) \cap \rho(y) \cap \rho(z) \neq \emptyset$ and so on. The tests that precede A and B, permit to refine ρ into a more precise ρ'. For instance, $\rho'(x) = \rho'(y) = \rho'(z) = \rho(x) \cap \rho(y) \cap \rho(z)$. This refined ρ' may allow to detect that indeed the communication between A and B is impossible in real computations. This happens when $\rho'(a) \cap \rho'(b) = \emptyset$ even though $\rho(a) \cap \rho(b) \neq \emptyset$. Similarly using ρ', we may deduce that certain input actions $A = c(w)$ never synchronize with an output action: in this case $\rho'(w) = \emptyset$. The presence of an input action like A is important for the precision of the analysis because it implies that all actions that follow A will never execute. Unfortunately, ρ' does not carry an analogous information also for the output actions. In fact, the values that ρ' associates to the names used in an output action do not reveal whether the action can synchronize or not.

The aim of point(ii) is approximated by adding $\rho'(u)$ to $\rho(v)$, in place of $\rho(u)$, where ρ' is the refined name-association introduced in the previous point.

The notion of satisfaction of tests and input actions and how they can be used to refine a name-association is illustrated in the following Example 2.

EXAMPLE 2 *Consider process $P = [a = r].a(z).[r = z].\overline{a}\langle r\rangle$ and the name-association ρ such that $\rho(a) = \{c, d\}, \rho(r) = \{d, e\}, \rho(z) = \{e, f\}$. Clearly, ρ satisfies both $[a = r]$ and $[r = z]$ since $\rho(a) \cap \rho(r) = \{d\}$ and $\rho(r) \cap \rho(z) = \{e\}$ and shows that $a(z)$ can synchronize with some output action because $\rho(z) \neq \emptyset$. However, ρ does not satisfy the two tests together: $\rho(a) \cap \rho(r) \cap \rho(z) = \emptyset$! From this we can deduce that $\overline{a}\langle r\rangle$ can never be executed and this consideration may be useful to improve the quality of the static analysis of a process that contains P.*

Consider now a process with two concurrent processes, $P = Q \,|\, R$, where $Q = [a = r].r(z).[z = k].\overline{a}\langle r\rangle$ and $R = [l = a].b(v)$ with the following name-association: $\rho(a) = \{c, d\}$, $\rho(r) = \{d, e\}$, $\rho(z) = \{e, f\}$, $\rho(k) = \{f, g\}$, $\rho(l) = \{d, e\}$, $\rho(b) = \{c, e\}$, $\rho(v) = \emptyset$. If we consider Q and R independently, then we would deduce that the action $A = \overline{a}\langle r\rangle$ of Q and the action $B = b(v)$ of R are both possible. However, it is easy to see that these 2 actions cannot synchronize. In fact, in order for these two actions to synchronize, ρ must satisfy $[a = b]$ together with all the tests and input actions that precede the 2 actions. Clearly this in not the case here: $\rho(a) \cap \rho(r) \cap \rho(l) \cap \rho(b) = \emptyset$. If, on the other hand, $\rho(a) = \{c, d, e\}$ (with all other values of ρ unchanged) then ρ would satisfy the condition and thus we would deduce that $\overline{a}\langle r\rangle$ and $b(v)$ may synchronize and then consider this action in our analysis of P.

The above ideas are rather intuitive and can be used to design a precise static analysis for the π-calculus. We show that this analysis can also be implemented rather efficiently, namely, we show that its time complexity is cubic in the size of the process that is analyzed. Also [4] presents a static analysis of the π-calculus that has been shown in [14] to be cubic. However, the analysis of [4] checks the tests one by one instead of simultaneously as our analysis and therefore, it is in general less precise. For instance, it would not be able to infer that the system of Example 1 behaves safely.

In [5] a static analysis is presented for a language slightly different from that of [4] and of the present article. The main differences of the language considered inS [5] are that the repetition operator is absent and that the role of tests is played by special input actions called selective inputs. A selective input, before to accept an input, tests if the input is in a given set of names. This is in some sense equivalent to group together many tests. Thus, one could say that in [5] a complementary approach is taken with respect to the one we follow: in place of making the analysis more sophisticated by grouping together tests, the language of [5] allows to directly write protocols with more complex tests. On these protocols a simple analysis obtains results similar to those that our analysis obtains on the same protocols described with a simpler language. However, it is not difficult to show that all results obtainable with the approach of [5] can be obtained with ours, but not vice versa.

The rest of the article is organized as follows. Section 1 contains some standard definitions about π-calculus, some new notation, and a static analysis that uses the ideas explained in Example 2 for improving the precision of the analysis. This analysis is very abstract in the sense that it does not specify how its sophisticated tests are actually performed. Section 2 is devoted precisely to the illustration of how this analysis can be implemented in cubic time. This is done in 2 steps: a pre-processing step followed by the static analysis part. Subsection 2 illustrates the theoretical foundations on which the actual implementation is built. The implementation of the static analysis part is described in Subsection 2. The correctness of the efficient implementation is discussed in Section 3 and the complexity of the algorithm is discussed in Section 4. The work ends with Section 5 where we try to link our algorithm with similar proposals and we point out some directions for future investigation.

For the sake of brevity, only the proof of the main theorem 2 is reported, the proofs of all technical lemmata are given in [6]. Also the pre-processing phase of the algorithm (and the proof that also this phase is $O(n^3)$) is described in [6].

1 Preliminaries

In this section we first recall the syntax of the π-calculus, [10]. The semantics of the language is explained only intuitively by means of an Example. A complete description can be found in [10, 15]. After this we introduce some new notation and we describe a simple static analysis of the π-calculus.

DEFINITION 1 *Let $\mathcal{N}$ denote an infinite set of* names, *ranged over by a, b, x, y, ... Let also τ be a symbol not in $\mathcal{N}$. Processes of π-calculus are constructed according to the following syntax:*

$$P ::= \mathbf{0} \mid \mu.P \mid P + P \mid P \,|\, P \mid (\nu c)P \mid [x = y].P \mid !P$$

where μ is either a silent action indicated with τ, or an output action $\overline{a}\langle b\rangle$, or an input action $a(b)$. In these actions a is called the subject *and b the* object *of the action. The "$.$" operator indicates sequential execution, the "$+$" operator indicates nondeterministic choice and "$\mid$" denotes parallel execution. The operator "$!$" means replication and is very important because it replaces recursion. The (νc) operator introduces private name c. Process $\mathbf{0}$ does nothing and thus we shorten $P.\mathbf{0}$ into P.*

EXAMPLE 3 *Consider process $P = \overline{a}\langle x\rangle.x(v).[v = x].R \,|\, a(w).[w = x].\overline{w}\langle x\rangle$. P consists of two processes that execute concurrently and whose input and output actions can synchronize. First, $\overline{a}\langle x\rangle$ can synchronize with $a(w)$ producing $x(v).[v = x].R \,|\, [x = x].\overline{x}\langle x\rangle$. Note that as an effect of this synchronization step, the name x has been substituted to w and thus the test $[w = x]$ has become $[x = x]$ which is satisfied. Thus the output $\overline{x}\langle x\rangle$ can execute and synchronize with $x(v)$ producing $[x = x].R \,|\, \mathbf{0} = R$. Clearly, in process P, actions and tests are partially ordered by the execution order induced by the sequencing operator "$.$". For instance, in $\overline{a}\langle x\rangle.x(v).[v = x].R$, $\overline{a}\langle x\rangle$ is executed first, then $x(v)$, then $[v = x]$ and finally R.*

In what follows P is a π-calculus process. In our analysis private names are considered as free names. So we will just ignore them. This causes no loss of precision because the fact that a name is private is irrelevant to our analysis. We simply study how names can propagate inside P. Being able to distinguish the private names from the free ones may become an issue when considering the problem of approximating the communications of P with the "outside world". $fn(P)$ and $bn(P)$ are, respectively, the set of free names of P and that of the bound names of P. We always assume, w.l.g., that $fn(P) \cap bn(P) = \emptyset$ and we call $n(P) = fn(P) \cup bn(P)$.

DEFINITION 2 *Two actions or tests of P are said to be* in concurrent positions *when either they occur in a same replicated subprocess* $!Q$ *or they occur in opposite sides of a parallel composition* $Q \mid R$.

PAIR(P) is the set of all pairs (A, B) *of actions and tests of* P *that are in concurrent positions and such that* A *and* B *are not both input or both output actions. We will also assume that* $(A, B) \in PAIR(P)$ *only when* A *occurs to the left of* B *in the process* P*. Thus, if* $(A, B) \in PAIR(P)$ *then* $(B, A) \notin PAIR(P)$.

EXAMPLE 4 *In process* P *of Example 3, if we call* $A = \overline{a}\langle x\rangle$, $B = x(v)$, $C = a(w)$ *and* $D = \overline{w}\langle x\rangle$, *the pairs in concurrent positions are* (A, C) *and* (B, D)*. If we consider the process* $!P$, *then we should add also* (A, B) *and* (C, D).

DEFINITION 3 *A* name-association *is a function* $\rho : \mathcal{N} \rightarrow \mathcal{PS}(\mathcal{N})$ *(where* $\mathcal{PS}$ *denotes the power set). Let* K *be a set of tests and actions, such that their variables are in* $\mathcal{N}$*. If* $n(K)$ *is the set of names contained in* K*, then the tests in* K *define an equivalence relation on* $n(K)$ *whose corresponding partition is denoted with* $\Pi(K)$*. We say that a name-association* ρ satisfies K*, when* $\bigcap_{x\in W} \rho(x) \neq \emptyset$ *for all* $W \in \Pi(K)$*. Observe that this condition also implies that for all input and output actions in* K*, if* a *is the subject of the action and* u *the object, then* $\rho(a)$ *and* $\rho(u)$ *are both not empty. That* ρ *satisfies* K *is denoted with* $\rho \models K$.

The refinement *of* ρ *wrt* K *is a new name-association* ρ' *as follows:*

$$\rho'(z) = \begin{cases} \rho(z) & \text{if } z \notin n(K) \\ \bigcap_{x\in W} \rho(x) & \text{if } z \in W \in \Pi(K) \end{cases}$$

Observe that if $\rho \not\models K$*, then for some* $x \in n(K)$, $\rho'(x) = \emptyset$*. Given any name* x*, with* $[x]_K$ *we denote the equivalence class in* $\Pi(K)$ *that contains* x*. This operation is obvious when* $x \in n(K)$*. When* $x \notin n(K)$*, conventionally* $[x]_K = \{x\}$.

Example 2 of the Introduction illustrates the above notions. The following technical fact is a basis for next results.

FACT 1 *Let* K *be a set of tests and* W *a non singleton equivalence class in* $\Pi(K)$*, let also* $S \in K$ *such that* $n(S) \cap W \neq \emptyset$ *then the following 3 statements hold:*

1 $n(S) \subseteq W$;
2 for any name $a \in W$, $[a]_{K\setminus\{S\}} \cap n(S) \neq \emptyset$.
3 let S' *be another test in* K *and assume that* a *is a name of* S *and* a' *one of* S' *and finally, let* W' *be the equivalence class in* $\Pi(K)$ *that contains* $n(S')$*. Then it holds that* $[a]_{K\setminus\{S'\}} \cap [a']_{K\setminus\{S\}} \neq \emptyset$ *iff* $W = W' = [a]_{K\setminus\{S'\}} \cup [a']_{K\setminus\{S\}}$

DEFINITION 4 *For any action or test* X *of* P*, with* $PRED(X)$ *we denote the set of actions and tests that precede* X *in* P *according to the execution order explained in Example 3 (observe that* $PRED(X)$ *does not contain* X*).* $COND(X) \subseteq$

$PRED(X)$ is the set of all the tests that precede X in P. Recall from Example 3 that the tests and actions in $PRED(X)$ are totally ordered according to their execution order. For any pair $(A, B) \in PAIR(P)$, $PRED(A, B) = PRED(A) \cup PRED(B)$ and $COND(A, B) = COND(A) \cup COND(B)$.

The following Example explains the previous Definition.

EXAMPLE 5 *Let P be the following process,*
$a(x).a(y).a(z).a(w).[x = w].\Big([x = y].\overline{b}\langle x\rangle \mid [x = z].\overline{c}\langle x\rangle \mid [w = z].[w = y].d(k)\Big)$
then $PRED(\overline{b}\langle x\rangle) = \{a(x), a(y), a(z), a(w), [x = w], [x = y]\}$, $COND(\overline{b}\langle x\rangle) = \{[x = w], [x = y]\}$ and the set of actions and tests in $PRED(\overline{b}\langle x\rangle) \cap PRED(d(k))$ is $\{a(x), a(y), a(z), a(w), [x = w]\}$. These actions and test precede both $\overline{b}\langle x\rangle$ and $d(k)$. Observe that all tests and actions in the above sets are listed in execution order.

This Section is concluded with a very simple but powerful static analysis for the π-calculus, that we call in fact the **Simple Analysis**.

1 Let P be the process to be analyzed and ρ_0 be the following name-association: for each free name x in P, $\rho_0(x) = \{x\}$ and for each bound name y in P, $\rho_0(y) = \emptyset$. Set $i = 1$ and proceed to the following step,

2 consider any pair $(A, B) \in PAIR(P)$, such that A is an output action $\overline{a}\langle u\rangle$ and B an input action $b(v)$. If $\rho_{i-1} \models PRED(A, B) \cup [a = b]$, let ρ'_{i-1} be the refinement of ρ_{i-1} wrt $PRED(A, B) \cup [a = b]$ (cf. Definition 3), then $\rho_i(v) = \rho_{i-1}(v) \cup \rho'_{i-1}(u)$.

3 if $\rho_i = \rho_{i-1}$ then stop with output $\rho_{SA} = \rho_i$, otherwise go back to step 2.

Even though the above analysis is very simple to describe, it contains operations that seem to require a high polynomial number of steps (in particular, the test whether $\rho_i \models PRED(A, B) \cup [a = b]$ and the computation of ρ'_{i-1} in step (2)). It is in fact fairly easy to see how to perform these operations in $O(n^5)$ steps. Observe that the operations of step (2) of the Simple Analysis that seem to be particularly complex are exactly those that perform the improvements mentioned in points (1) and (2) of the Introduction. Improving this bound was for us not easy, but we succeeded and in the following Sections we report the algorithm we found. This algorithm implements the Simple Analysis and has worst case time complexity $O(n^3)$.

The reader may wonder why in the above step (2) we consider $PRED(A, B)$ and not $COND(A, B)$. Notice that $\Pi(COND(A, B)) \subseteq \Pi(PRED(A, B))$ and in some cases the containment is proper and the difference consists of some singletons. This may happen when $PRED(A, B)$ contains some input or output action with names that do not appear in any test in $COND(A, B)$. As already observed in the Introduction (cf. Point (1)), the names in these singletons that are objects of input actions, can be exploited for improving the analysis. This explains the choice in step (2).

2 The Efficient Algorithm

In this Section we explain how the Simple Analysis of the previous Section can be implemented efficiently obtaining an algorithm that has cubic worst case time complexity. This algorithm will be called in what follows **the Efficient Algorithm**.

The problem is to perform efficiently the tests of point (2) of the Simple Analysis and the computation of a refined name-association (called ρ'_{i-1} in the Simple Analysis). The key idea is that of computing and maintaining all the necessary refined values throughout the analysis (instead of recomputing them each time they are needed as in a naive implementation of the Simple Analysis). To this end we introduce a set of new names whose role is to hold the refined values. Roughly this works as follows. Consider a pair $A = \overline{a}\langle u \rangle$ and $B = b(v)$ that may synchronize. The tests and actions that precede A and B, together with $[a = b]$, determine equivalence classes of names, cf. Example 2. Call these classes $X_1, \ldots, X_m$. For each X_i, a new name c_i is introduced and during the analysis, if ρ is the name-association computed so far, then the value of $\rho(c_i)$ will always satisfy the following relation: $\rho(c_i) = \bigcap_{y \in X_i} \rho(y)$. Thus, $\rho(c_i)$ is the refined value of each name in X_i. Namely, it is the set of names that are assigned to all the names in X_i and that satisfy all the tests and actions in $PRED(A, B)$ that have formed the class X_i. The above description is necessarily simplified. In particular, the new names that are used in the algorithm are not simply c_i. For instance, the new name that corresponds to the class that contains the subjects a and b is $c_{A,B}$ and the new name that corresponds to the class that contains the object u of the output is $c_{A\downarrow B}$. With these new names that hold the refined values of the equivalence classes, it is possible to implement the actions of point (2) of the Simple Analysis as follows :

(a) the synchronization between A and B is considered by the analysis only when each $\rho(c_i) \neq \emptyset$, this guarantees that all actions and tests in $PRED(A, B) \cup [a = b]$ can be executed/satisfied; observe that $[a = b]$ is added to check that A and B can actually communicate;

(b) the synchronization of A and B is modelled by adding $\rho(c_{A\downarrow B})$ to $\rho(v)$. Observe that this is the refined value of u, as requested in point (2) of the Simple Analysis.

The number of new names introduced is quadratic. However, maintaining the value of each of these names (and also of those in $n(P)$ that in what follows will be called *old*) takes linear time. This follows from the fact that each new name c depends on only 2 other names (new or old), say c' and c''. This dependency is as follows: when $x \in \rho(c') \cap \rho(c'')$ then x must be also in $\rho(c)$. Moreover, c' and c'' are strictly smaller than c wrt a partial order and thus there is no circularity in these dependencies. Exploiting this fact, it is possible to maintain the value of each name in linear time.

The test described in point (a) above can also be done very efficiently: a counter $Ready(c_{A\downarrow B})$ is initially set to the number of classes in $\Pi(PRED(A, B) \cup [a = b])$ and is decreased by 1 each time the value of $\rho(c_i)$ (where c_i corresponds to one of the classes) becomes not empty. When $Ready(c_{A\downarrow B}) = 0$, the test of point (a) is satisfied and thus the analysis performs the action of point (b). We have actually implemented this sophisticated static analysis algorithm. The C++ source is downloadable from the directory “www.math.unipd.it/˜colussi/Analizer/”.

theoretical foundations

This Section is devoted to the construction of the theoretical foundations of the Efficient Algorithm and of the proof that it is cubic in the size of P (P is always the process under analysis). It mainly contains three things:

(I) The precise definition of the new names that are needed for the Efficient Algorithm together with a partial order on them;

(II) The proof that the value of each new name v depends on that of only two other names $lc(v)$ and $rc(v)$;

(III) The proof that for each pair (A, B) of input/output action one can compute once and for all a set X of names, such that the test of point(a) above is performed by checking that for every name $x \in X$, $\rho(x) \neq \emptyset$. It is also important that $|X|$ is linear in the size of P.

Points (II) and (III) are fundamental for showing that the Efficient Algorithm is cubic in the size of P. Recall that $n(P)$ stands for the set of all the names of P, i.e., $n(P) = fn(P) \cup bn(P)$, where, w.l.g., we assume that $fn(P) \cap bn(P) = \emptyset$. In what follows these names are called *old* to distinguish them from the new ones that we are going to introduce. As explained above, each new name v stands for a set of old names that is indicated with $[v]$. This notation is extended to old names x, letting $[x] = \{x\}$. The set of new names that we create for P is denoted $new(P)$ and consists of two parts $news(P)$ and $newp(P)$. The first part contains new names that corresponds to a single test T of P ('s' stands for single), whereas the second one contains new names that correspond to pairs ('p' stands for pair) as follows: these names correspond either to pairs (A, B) of an input and an output action which are in concurrent position in P or to pairs (T, T') of tests which are in concurrent position in P.

DEFINITION 5 *For each test $T = [a = b]$ of P, $news(P)$ contains a new name c_T that stands for the set of old names $[c_T] = [a]_{COND(T) \cup T}$.*

The following is an easy consequence of Fact 1(2) that is useful for the next Lemma.

FACT 2 *Let c_T be the new name that corresponds to a test $T = [a = b]$, then $[c_T] = [a]_{COND(T)} \cup [b]_{COND(T)}$.*

It is useful to define a partial order on the set $news(P) \cup n(P)$.

DEFINITION 6 *The relation $\preceq$ on $news(P) \cup n(P)$ is defined as follows.*

- *for each $x \in news(P) \cup n(P)$, $x \preceq x$;*
- *the old names in $n(P)$ are unrelated among each other and for each $x \in n(P)$ and $y \in news(P)$, $x \preceq y$;*
- *for any two names c_T and $c_S \in news(P), c_T \preceq c_S$ iff T precedes S in the execution order.*

In what follows we will write $x \prec y$ to denote $x \preceq y$ and $x \neq y$.

In the following Lemma we show point (II) for the names in $news(P)$.

LEMMA 1 *Let c_T be a new name in $news(P)$, where $T = [a = b]$. There are two names (either old or in $news(P)$) $lc(c_T)$ and $rc(c_T)$ such that $[c_T] = [lc(c_T)] \cup [rc(c_T)]$. Moreover, $lc(c_T) \prec c_T$ and $rc(c_T) \prec c_T$.*

Names in $newp(P)$ correspond to pairs $(A, B) \in PAIR(P)$ whose names may interact in some way. Interaction may be of two types: either A and B are an output and an input action that may communicate or A and B are tests and there is a name in A and a name in B that are equated by the tests in $COND(A,B)$.

Recall from Definition 2 that pairs $(A,B) \in PAIR(P)$ are such that A is always to the left of B in P. In this way we avoid the nuisance of having a new name for (A,B) and another for (B,A), while only one of them is enough for the analysis.

DEFINITION 7 *$newp(P)$ contains the following names:*

(a) For each test/test pair $(T,T') \in PAIR(P)$, where $T = [a = c]$ and $T' = [b = d]$, and such that $[a]_{COND(T,T') \cup T} \cap [b]_{COND(T,T') \cup T'} \neq \emptyset$ a new name $c_{T,T'}$ is in $newp(P)$. This name stands for the set of old names $[c_{T,T'}] = [a]_{COND(T,T') \cup T \cup T'}$.

(b) For each input/output pair $(A,B) \in PAIR(P)$, where $\overline{a}\langle u \rangle$ is the output action and $b(w)$ is the input action, $newp(P)$ contains two new names $c_{A,B}$ and $c_{A \downarrow B}$. The name $c_{A,B}$ is intended to stand for the set $[c_{A,B}] = [a]_{COND(A,B) \cup [a=b]}$ of old names, whereas $c_{A \downarrow B}$ stands for the set $[c_{A \downarrow B}] = [u]_{COND(A,B) \cup [a=b]}$.

Notice that in point (b) of the above Definition no assumption is made on which one between A and B is input and which is output. Moreover, $[a = b]$ is not a test in P. We add it to $COND(A,B)$ to mimic the fact that the synchronization of A and B is possible only when this condition is satisfied. Observe that this is coherent with step (2) of the Simple Analysis, cf. Section 1. In what follow with $nn(P)$ we denote $n(P) \cup new(P)$. The partial order $\preceq$ is easily extended to $nn(P)$ as follows.

DEFINITION 8 *The partial order $\preceq$ is extended to $nn(P)$ adding the following points to those of Definition 6:*

- *for each name $x \in newp(P)$, $x \preceq x$;*
- *all old names are smaller than all new names of $newp(P)$;*
- *a name c_T is smaller than every name $c_{X,Y}$ and $c_{X \downarrow Y}$;*
- *if T_1 and T_2 are tests and X_1 and X_2 are either two tests or an input and output action, then $c_{T_1,T_2} \preceq c_{X_1,X_2}$ iff for each $i \in [1,2]$ either T_i precedes X_i or $T_i = X_i$;*
- *for all name x, if $x \preceq c_{A,B} \in newp(P)$, where A and B are an input and an output action, then $x \preceq c_{A \downarrow B}$.*

FACT 3 *The relation $\preceq$ of Definition 8 is a partial order.*

We want now to show point (II) also for the names in $newp(P)$. To this end we follow the same strategy that was used in Lemma 1: for any $v \in newp(P)$, we show that $[v]$ can be split into two parts for which there are corresponding names. The following simple consequence of Fact 1(c) is useful for this.

FACT 4 *Let $c_{T,T'}$ be a new name introduced in step (a) of Definition 7 and let $T = [a = c]$ and $T' = [b = d]$. It is true that $[a]_{COND(T,T') \cup T \cup T'} = [a]_{COND(T,T') \cup T} \cup [b]_{COND(T,T') \cup T'}$.*

LEMMA 2 *Let $v \in newp(P)$, there are names $lc(v)$ and $rc(v)$ in $n(P) \cup news(P) \cup newp(P)$ such that $[v] = [lc(v)] \cup [rc(v)]$ and moreover, these names are smaller than v with respect to the partial order $\prec$.*

- A boolean array $Bound$ indexed on the set $nn(P)$. $Bound[x] = 1$ if there is a name $z \in n(P)$ such that $Rho[z, x] = 1$.
- An array $Ready$ of integers such that for each name $v = c_{A\downarrow B}$, $Ready[v]$ is initially set to the cardinality of $Pred(v)$. For each name $x \in Pred(v)$, there is a list isP_x that contains v and all other names having x in their $Pred$ set.
- Two arrays Lc and Rc indexed on the set $new(P)$ (to store $lc(v)$ and $rc(v)$) and for all $x \in nn(P)$ a list $isLc_x$ (for "x is Left Component of") of all names v such that $x = lc(v)$ and a list $isRc_x$ (for "x is Right Component of") of all those names v such that $x = rc(v)$.
- An array Rho of booleans indexed in $n(P) \times nn(P)$.

Table 1. Main data structures used by the algorithm.

The following Theorem summarizes what we have shown.

THEOREM 1 *For each name v in $new(P)$ there exist names $lc(v)$ and $rc(v)$ (possibly equal) that are strictly smaller than v wrt the partial order $\prec$ defined on $nn(P)$.*

COROLLARY 1 *The relation on $nn(P)$ defined by the $lc(v)$ and $rc(v)$ functions among names is noncircular.*

We turn now to point (III). Consider an input/output pair (A, B) and let a and b be the subjects of the two actions and u the object of the output one. As explained in (III), in order for the Efficient Algorithm to check whether the pair (A, B) can synchronize, all the refined values corresponding to the equivalence classes of $\Pi(PRED(A, B) \cup [a = b])$ should be not empty. In order to perform this test for each such class X there must exist a new or an old name x that corresponds to the class and thus that will hold its refined value. This is shown in the following Lemma.

LEMMA 3 *Let $v = c_{A\downarrow B}$ and let a and b be the subjects of the actions A and B. For each class $X \in \Pi(PRED(A, B) \cup [a = b])$ there is a name $x \in nn(P)$ such that $[x] = X$. Moreover, $x \prec c_{A\downarrow B}$.*

Let us conclude the Section with a notation that will be useful in the next one: For any name $c_{A\downarrow B}$, $Pred(c_{A\downarrow B})$ denotes the names (that were just shown to exist) that correspond to the equivalence classes of $\Pi(PRED(A, B) \cup [a = b])$.

the implementation

The Efficient Algorithm uses several data structures and is composed of two parts: a pre-processing part and the static analysis part. For the sake of brevity, we only describe the static analysis part, in Table 2, and the most important data structures used in that part, in Table 1. The pre-processing part and all other data structures used by the algorithm are described in [6]. Data structures in Table 1 have the following purpose. In the matrix Rho we assume that the first $|n(P)|$ columns correspond to the old names (i.e., those in $n(P)$). This matrix holds, throughout the execution of the algorithm, the name-association computed at each moment.

The name-association ρ_{Rho} on $nn(P)$ that corresponds to a given matrix Rho is as follows: $\forall x \in n(P)$ and $w \in nn(P)$, $x \in \rho_{Rho}(w)$ iff $Rho[x, w] = 1$. The restriction

```
$Compute(P)$
    Set to 0 all entries of arrays $Rho$ and $Bound$.
    Call $Try(u)$ for all $u \in fn(P)$ (i.e., those $u$ that do not occur in $P$ as the object of an input action).

$Try(u)$
    if $Rho[u,u] = 0$ then set $Rho[u,u] = 1$ and call $VisitIsC(u,u)$ .
    if $Bound[u] = 0$ then set $Bound[u] = 1$ and call $VisitIsP(u)$.

$Filter(u,v)$
    set $Rho[u,v] = 1$ and call $VisitIsC(u,v)$.
    if $v = c_{A\downarrow B}$, $Ready[v] = 0$ and $Rho[u,w] = 0$, where $w$ is the object of the input action in the input/output pair $(A,B)$ associated to $v$, then call $Close(u,w)$.
    if $Bound[v] = 0$ set $Bound[v] = 1$ and call $VisitIsP(v)$.

$Transmit(v,w)$
    call $Close(z,w)$ for all $z \in n(P)$ such that $Rho[z,w] = 0$ and $Rho[z,v] = 1$.

$Close(u,v)$
    set $Rho[u,v] = 1$ and call $VisitIsC(u,v)$.
    if $Bound[v] = 0$ then set $Bound[v] = 1$ and call $VisitIsP(v)$.
    call $Close(z,v)$ for all $z \in n(P)$ such that $Rho[z,v] = 0$ and $Rho[z,u] = 1$.
    call $Close(u,z)$ for all $z \in n(P)$ such that $Rho[u,z] = 0$ and $Rho[v,z] = 1$.

$VisitIsC(u,v)$
    call $Filter(u,x)$ for all $x \in isLc_v$ such that $Rho[u,Rc[x]] = 1$ and $Rho[u,x] = 0$.
    call $Filter(u,x)$ for all $x \in isRc_v$ such that $Rho[u,Lc[x]] = 1$ and $Rho[u,x] = 0$.

$VisitIsP(v)$
    for all $x \in isP_v$ set $Ready[x] = Ready[x] - 1$ and in case $Ready[x] = 0$ call $Transmit(x,w)$ where $w$ is the object of the input action in the input/output pair $(A,B)$ associated to $x$ (recall that all $x \in isP_v$ are of type $x = c_{A\downarrow B}$).
```

Table 2. The static analysis part of the Efficient Algorithm

of ρ_{Rho} to the old names in $n(P)$ is denoted $\bar{\rho}_{Rho}$. Lc and Rc specify for each new name v the $lc(v)$ and $rc(v)$. $Bound$ is used to signal when a name x is assigned a not empty value, i.e., $\rho(x) \neq \emptyset$. $Ready$ is defined only for names of the form $c_{A\downarrow B}$. Its initial value is the cardinality of $Pred(c_{A\downarrow B})$. Each time a name x in this set becomes bound, then $Ready[c_{A\downarrow B}]$ is decreased by 1. On the other hand, $isLc_x$ is used to reach all those names v that have x as $lc(v)$ and similarly for $isRc_x$. isP_x lists those names of the form $c_{A\downarrow B}$ such that $x \in Pred(c_{A\downarrow B})$.

3 Correctness of the Efficient Algorithm

In what follows we show that the Efficient Algorithm computes the same name-association as the Simple Analysis of Section 1. In the following Lemma we list some important facts that are true about the Efficient Algorithm.

LEMMA 4

1 *For all* $u \in n(P)$ *and* $z \in nn(P)$*,* $\mathit{Rho}[u, z]$ *is set to 1 iff* $\mathit{Rho}[u, v] = 1$ *for all* $v \in [z]$*.*

2 *Consider any* $v = c_{A \downarrow B}$*, and let* a *and* b *be the subjects of actions* A *and* B*. The following holds: Initially* $\mathit{Ready}[v] > 0$ *and* $\mathit{Ready}[v]$ *became 0 as soon as* $\rho_{\mathit{Rho}} \models \mathit{PRED}(A, B) \cup [a = b]$*.*

3 *Consider any* $u, w \in n(P)$ *where* w *is the object of an input action. Initially* $\mathit{Rho}[u, w] = 0$ *and* $\mathit{Rho}[u, w]$ *is set to 1 as soon as* $\mathit{Rho}[u, v] = 1$ *and* $\mathit{Ready}[v] = 0$ *for some* $v = c_{A \downarrow B}$ *such that the object of the input action is* w*.*

Using the above facts we can now show the correctness of the Efficient Algorithm. Let ρ_{EA} be the name-association computed by the Efficient Algorithm and ρ_{SA} that computed by the Simple Analysis. Clearly, ρ_{EA} is ρ_{Rho}, where Rho is the final matrix produced by the Efficient Algorithm. Recall that $\bar{\rho}_{EA}$ is its restriction to $n(P)$.

THEOREM 2 $\bar{\rho}_{EA} = \rho_{SA}$

Proof. For this proof it is convenient to consider that ρ_{SA} is extended to $nn(P)$ by setting for each new name v, $\rho_{SA}(v) = \bigcap_{x \in [v]} \rho_{SA}(x)$. Moreover, it will be useful to consider the computation of the Efficient Algorithm and of the Simple Analysis and the sequence of name-associations produced by the two processes. With ρ_{EA_i} we denote the name-associations obtained by the Efficient Algorithm after the first i changes operated to the initial name-association which is the empty matrix Rho and thus the empty name-association. Rho_i is the corresponding matrix. Similarly, ρ_{SA_i} denotes the name-association computed by the Simple Analysis after i changes operated on the initial name-association ρ_0. Recall that ρ_0 is the identity for the free names of P and the empty set for the other names.

Let us first show that $\rho_{EA} \subseteq \rho_{SA}$. We reason by contradiction. Assume that there are old names $x, y \in n(P)$ such that $y \in \rho_{EA_{i+1}}(x)$, but that $y \notin \rho_{SA}(x)$. We assume that the (i+1)-th step introduces this difference for the first time and thus $\rho_{EA_i} \subseteq \rho_{SA}$. By Lemma 4(3), from the fact that $\mathit{Rho}_i[y, x] = 1$, it follows that there must be a new name $v = c_{A \downarrow B}$ such that $\mathit{Rho}_i[y, v] = 1$ and $\mathit{Ready}[v] = 0$. Let also u be the object of the output action in (A, B). From Lemma 4(2), it follows that $\mathit{Ready}[v] = 0 \Rightarrow \rho_{EA_i} \models \mathit{PRED}(A, B) \cup [a = b] \Rightarrow \rho_{SA} \models \mathit{PRED}(A, B) \cup [a = b]$. Let $\rho'_{SA} = \rho_{SA}|_{\mathit{PRED}(A,B) \cup [a=b]}$. Observe now that, since $\mathit{Rho}_i[y, v] = 1$, from Lemma 4(1), it follows that $y \in \rho_{EA_i}(d), \forall d \in [v] = [u]_{\mathit{PRED}(A,B) \cup [a=b]}$. Hence, $y \in \rho'_{SA}(u)$ and therefore, $y \in \rho_{SA}(x)$. This clearly contradicts the initial hypothesis.

Let us now prove that $\rho_{EA} \supseteq \rho_{SA}$. Observe that the Simple Analysis starts from ρ_0. It suffices to look at the function $\mathit{Compute}$ of Table 2 to see that $\rho_{EA} \supseteq \rho_0$.

Make the following Assumption (*): for the first time at the (i+1)-th step the Simple Analysis adds y to $\rho_{SA_i}(x)$ such that $y \notin \rho_{EA}(x)$. In order to meet Assumption (*) the

Simple Analysis must consider an input/output pair (A, B). Assume that the subjects of the 2 actions are a and b, whereas the object of the output one is u, whereas that of the input, from the hypothesis, must be x. Moreover, it must be that **(A)** $\rho_{SA_i} \models PRED(A,B) \cup [a=b]$ and if $\rho'_{SA_i} = \rho_{SA_i}|_{PRED(A,B)\cup[a=b]}$, then **(B)** $y \in \rho'_{SA_i}(u)$.

From Assumption (*) and statement **(A)**, it follows that $\rho_{EA} \models PRED(A,B) \cup [a=b]$ and, by Fact 4(2), we derive that **(C)** $Ready[c_{A\downarrow B}] = 0$. From **(B)** and Assumption (*), it follows that $\forall d \in [c_{A\downarrow B}] = [u]_{PRED(A,B)\cup[a=b]}$, $Rho[y,d] = 1$, and thus, by Fact 4(1), that **(D)** $Rho[y, c_{A\downarrow B}] = 1$. From **(C)** and **(D)**, by Fact 4(3), we can conclude that $Rho[y,x] = 1$ in contradiction with our initial assumption. □

4 Complexity of the algorithm

It is quite simple to prove that the Efficient Algorithm requires time $O(n^3)$ (where n is the size of the π-expression P in input): for each function we find a bound for the number of times it is called and a bound for the time required to execute the function. The execution time of each function does not include the time required to execute the function calls it may contain. At the end it suffices to sum everything up in order to obtain a bound for the total time required by whole Efficient Algorithm.

Observe that there are at most $O(n)$ actions or tests in P and at most $O(n)$ old names in $n(P)$ while the cardinality of $nn(P)$ can be $O(n^2)$. The function $Compute$ is called only once and requires $O(n^3)$ time. This time is needed fundamentally for initializing matrix Rho. Try is called $O(n)$ times (at most once for each name in $n(P)$) and its execution requires time $O(1)$. $Filter$ is called $O(n^3)$ times (at most once for each entry of Rho) and it requires time $O(1)$. $Transmit$ is called $O(n^2)$ times (at most once for each name $c_{A\downarrow B}$) and it requires time $O(n)$. $Close$ is called $O(n^2)$ times (at most once for each pair of names in $n(P)$) and it requires time $O(n)$. $VisitIsC$ is called at most once for each pair (u,v) and requires time proportional to the length of lists $isLc_v$ and $isRc_v$. Since the sum of the lengths of all lists $isLc$ and $isRc$ is $O(n^2)$ the total time required is $O(n^3)$. For function $VisitIsP$ the reasoning is more subtle. $VisitIsP$ is called at most once for each name $v \in nn(P)$ and requires time proportional to the length of the list isP_v. Since the sum of the lengths of all lists isP_v is $O(n^3)$ (because each name in $c_{A\downarrow B}$ can be inserted in at most $O(n)$ such lists), the total time required by this function is $O(n^3)$.

Since the above functions use the data structures shown in Table 1, it is important to consider also the cost of constructing these structures in a pre-processing phase.

The pre-processing can be done in time $O(n^3)$. A detailed description of the pre-processing and the proof that it require time $O(n^3)$ is given in [6]. Here we explain why this is the case on a more intuitive level. The pre-processing consists of a double visit of the parse tree of the π-expression P in input. For each action or test A encountered in the first visit we do a second visit to find all action or test B that is in concurrent position with A. At each step we update a disjoint-set data structure cls that holds the classes in $\Pi(PRED(A,B))$. The data structure cls is augmented by the name of classes and a list $Pred$ that links names of classes in cls. Since cls and $Pred$ can be updated in many different ways, they must be copied before an update takes place. Double visiting the parse tree takes time $O(n^2)$ and copying the structures cls requires time $O(n)$. Thus the total time used is $O(n^3)$.

5 Related work and perspectives

Bodei et al. in [4] proposed a static analysis of the π-calculus that in [14] was shown to have a cubic time complexity. This analysis considers that any input/output pair (A, B) can synchronize only when all tests that precede them are satisfied by the name-association computed so far, but the tests are considered one at the time and not together as our analysis does. In [13] the analysis of [4] is extended to the $spi-$calculus [2] maintaining the same time complexity. This extended version still handles the cryptographic primitives one at the time as before.

Also Venet [17] and Feret [8, 9] have proposed static analyses of the π-calculus that are formulated in the abstract interpretation framework, [7]. They first introduce non standard semantics and then define their analyses as abstractions of these semantics. The semantics they propose are expressive enough to encompass *non uniform* analyses, that is analyses able to distinguish among the different copies of a same replicated subprocess and among the names that these copies can define and transmit. In fact, these analyses are useful, for instance, for evaluating the resource usage inside a system. These works are rather different from the present one. They focus on the expressivity of the analyses rather than on their efficient implementation.

Clearly, many other methods, different from ours and from those mentioned before, have also been used for proving properties of protocols. These methods include model checking [12], type systems [2], the use of theorem provers [3, 1]. Often these proposals try to establish more sophisticated properties of protocols than what our static analysis can compute. However, we believe that any method for inferring properties of protocols must lay on a precise knowledge of the name-association the protocol actually produces and this is precisely what our static analysis computes with high precision and also efficiently.

In the future we intend to substantiate the above statement by extending our analysis in various ways. First of all we will further enhance the precision of our analysis by including into it the detection of "blocked" output actions, i.e., outputs that cannot synchronize with any input and that, therefore, block the successive actions. Our approach improves precision by considering the global condition $COND(A) \cup COND(B) \cup [a = b]$ under which transmission of a name can take place from the output action $A = \overline{a}\langle u \rangle$ to the input action $B = b(w)$. It is possible to further improve the precision of the analysis by considering the transmission of each name through sequences of synchronizing pairs of input/output actions, evaluating together all tests that precede these actions. We obviously expect that the complexity of this improved analysis will grow with the length of the action sequences considered.

Finally, we will apply our method to the analysis of extensions of the π-calculus that include various cryptographic primitives.

References

[1] Martìn Abadi and Bruno Blanchet. Computer assisted verification of a protocol for certified email. In *Proceedings of 10th SAS*, number 2694 in LNCS, pages 316–335, 2003.

[2] Martìn Abadi and Andrew Gordon. A calculus for cryptographic protocols—the spi calculus. *Information and Computation*, 148(4):1–70, 1999.

[3] G. Bella and L.C. Paulson. Kerberos version iv:inductive analysis of the secrecy goals. In *Proceedings of ESORICS 98*, number 1485 in LNCS, pages 361–375, 1998.

[4] Chiara Bodei, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Static analysis for the π-calculus with applications to security. *Information and Computation*, 165:68–92, 2001.

[5] C. Priami C.Bodei, P.Degano and N. Zannone. An enhanced cfa for security policies. In *Proceedings of WITS'03, pp.131-145, Warszawa*, 2003.

[6] L. Colussi, G. Filè and A. Griggio. Precise Analysis of π-calculus in cubic time. Preprint n. 20 *Dipartimento di Matematica Pura ed Applicata, University of Padova*, 2003.

"www.math.unipd.it/~colussi/DMPA-Preprint20-2003.ps"

[7] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of 4th ACM POPL*, pages 238–252, 1977.

[8] Jérôme Feret. Confidentiality analysis of mobile systems. In *Proceedings of 7th SAS*, number 1824 in LNCS, 2000.

[9] Jérôme Feret. Occurrence counting analysis. In *Proceedings of GETCO 2000, appeared on ENTCS*, number 39, 2001.

[10] Robin Milner. *Communicating and mobile systems: the π-calculus*. Cambridge University Press, 1999.

[11] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes (i and ii). *Information and Computation*, 100(1):1–77, 1992.

[12] J.C. Mitchell, V. Shmatikov, and U. Stern. Finite state analysis of ssl 3.0. In *Proceedings of 7th USENIX Security Symposium*, pages 201–216, 1998.

[13] Flemming Nielson, Hanne Riis Nielson, and Helmut Seidl. Cryptographic analysis in cubic time. In *ENTCS*, number 62 in ?, 2002.

[14] Flemming Nielson and Helmut Seidl. Control flow analysis in cubic time. In *Proceedings of ESOP '01*, number 2028 in LNCS, pages 252–268, 2001.

[15] Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[16] R. L. Rivest T. H. Cormen, C. E. Leiserson and C. Stein. *Introduction to Algorithms*. The Mit Press, 1998.

[17] Arnaud Venet. Automatic determination of communication topologies in mobile systems. In *Proceedings of 5th SAS*, number 1503 in LNCS, pages 152–167, 1998.