PicNIc - Pi-calculus Non-Interference checker*
(Tool paper)

S. Crafa
Univ. di Padova
Italy

M. Mio
Univ. of Edinburgh
UK

M. Miculan, C. Piazza
Univ. di Udine
Italy

S. Rossi
Univ. Ca' Foscari
di Venezia, Italy

Abstract

PICNIC *is a tool for verifying security properties of systems, namely* non-interference *properties of processes expressed as terms of the π-calculus with two security levels and declassification primitives. More precisely, it checks whether inserting a process into two different high contexts no information leakage to the low level observers occurs. These properties are decidable over finite control processes, but decidability can be extended by compositionality also to some infinite state processes. Notably,* PICNIC *has been developed in Fresh O'CaML, a dialect of CaML with native support for binders and fresh/local names; thus, this work can be seen also as a non-trivial case study about the applicability of these new programming languages.*

1. Introduction

PICNIC (Pi-calculus Non-Interference checker) is a tool for verifying security properties of concurrent systems described as terms of a typed π-calculus with two security levels and declassification primitives. It is freely available, together with some basic instructions and examples, at `http://www.dimi.uniud.it/mio/picnic/`. The basic tool's functionalities are the verification of the *non-interference* properties $\mathcal{NI}(\cong_L)$ and $\mathcal{NI}(\cong_L^{\mathsf{dec}})$ presented in [1] and the compositional property $\mathcal{NI}(\Bumpeq_L^{\mathsf{dec}})$ we introduce here. Intuitively, a system with two security levels, *high* (*confidential*) and *low* (*public*), satisfies the non-interference property $\mathcal{NI}(\cong_L)$ if, when inserted into different high level contexts it always exhibits the same low level behaviour (w.r.t. $\cong_L$). Hence, even if the system interacts with a high level malicious process (e.g., a trojan horse) confidential information is not leaked to the low level. However, in many real-world applications strong non-interference is neither achievable nor desirable. Indeed, partial controlled flows are often part of the intended behaviour of systems. To model intentional information leakages, declassification primitives have been added to the π-calculus and the security property $\mathcal{NI}(\cong_L^{\mathsf{dec}})$ (based on the low level observation $\cong_L^{\mathsf{dec}}$) has been defined to ensure that only explicitly authorized flows occur. The properties $\mathcal{NI}(\cong_L)$ and $\mathcal{NI}(\cong_L^{\mathsf{dec}})$ coincide over the calculus without declassifications.

While $\mathcal{NI}(\cong_L)$ is compositional with respect to the language operators, $\mathcal{NI}(\cong_L^{\mathsf{dec}})$ is not. As a consequence, $\mathcal{NI}(\cong_L)$ can be efficiently verified by decomposing the system into its sub-components, while to check $\mathcal{NI}(\cong_L^{\mathsf{dec}})$ it is necessary to examine the entire system. To improve space/time performances, we introduce the compositional property $\mathcal{NI}(\Bumpeq_L^{\mathsf{dec}})$ that is stronger than $\mathcal{NI}(\cong_L^{\mathsf{dec}})$ and coincides with $\mathcal{NI}(\cong_L)$ when no declassifications occur.

PICNIC allows one also to check the low level observation relations $\cong_L$, $\cong_L^{\mathsf{dec}}$ and $\Bumpeq_L^{\mathsf{dec}}$. This capability can be used, for instance, in the developing of complex systems; indeed the concrete final system can be obtained from an initial abstract specification through stepwise refinements which can be proved to preserve the low level behaviour and the security guarantees of the initial specification.

The tool PICNIC consists of a *graphical interface* (written in JAVA) and a *kernel module* which is composed of three main parts: (1) the *parser* which checks for syntax errors and builds the syntax tree out of the π-calculus processes; (2) the *semantic module* calculating the transition relations from a process to its sequents; (3) the *verifier* which checks the special observation relations used to infer the security properties. As far as $\cong_L^{\mathsf{dec}}$ (and then $\cong_L$) is concerned, the implemented algorithm computes its coinductive characterization, that is the *early weak partial bisimilarity on low level actions* presented in [1]. This is done in an on-the-fly fashion, implementing the ideas presented by Lin in [5]. Moreover, in order to improve the performances, the tool allows one to adopt the compositionality properties of $\mathcal{NI}(\Bumpeq_L^{\mathsf{dec}})$ to both drastically reduce the space/time complexities and check some infinite state processes.

Notably, the PICNIC kernel is implemented in FRESH O'CAML, a new programming language extending OBJEC-

*Partially supported by MIUR Projects PRIN 2005 grants 2005015491, 2005015785 and 2005015824.

TIVE CAML with data-types and primitives for dealing with names, binders and the α-equivalence of processes (see [9] and `http://www.fresh-ocaml.org/`). Thus, the present work is also interesting as a testbed and case study for this language.

2. Non-Interference in the π-calculus

The concept of *non-interference* [4] has been introduced to formalize the absence of information flow in multilevel systems: it demands that the low level observation of the system is independent from the behaviour of its high components. Our tool aims at verifying non-interference properties of processes described as terms of a multilevel typed π-calculus with declassification primitives. Types are used to assign secrecy levels to channels and to statically check that low level channels do not carry high level values. The low level behaviour of processes is captured by the observation relation $\cong_L^{\mathsf{dec}}$ (which coincides with $\cong_L$ in the calculus without declassifications). This is a partial equivalence relation (*per* model, see [8]) , that is symmetric and transitive but not reflexive. The security property $\mathcal{NI}(\cong_L^{\mathsf{dec}})$ is defined as the reflexive closure of $\cong_L^{\mathsf{dec}}$ and characterizes the class of processes whose observable behavior is independent from the surrounding high context. Formally, this is expressed by the following contextual property: if P is non-interfering then for all low contexts $C_L[\,]$ (interacting with the process filling the hole just through low channels) and for all high contexts $C_H^1[\,]$ and $C_H^2[\,]$ (interacting with the process filling the hole just through high channels carrying either bound values or high free names)

$$C_L[C_H^1[P]] \;\cong_L^{\mathsf{dec}}\; C_L[C_H^2[P]].$$

For this class of high contexts, which is stricter than the one considered in [1], $\cong_L^{\mathsf{dec}}$ has a coinductive characterization that is a bisimulation-based relation on typed labelled transition systems. This is decidable for the recursion-free calculus and even for the finite-control π-calculus where the number of parallel components in any process is bounded by a constant.

In order to present the coinductive characterizations of our security properties as they are implemented in PICNIC, let us first briefly introduce our calculus.

The calculus we consider is a synchronous, monadic, typed π-calculus where types are used to assign secrecy levels to channels. PICNIC implements the simplest case of two security levels, *high* (H) and *low* (L) with $L \preceq H$. We presuppose a countably-infinite sets of names and a countably-infinite sets of variables, ranged over by $n, \ldots, q$ and $x, \ldots, y$, respectively. We often use a, b, c to range over both names and variables.

The syntax is as follows:

$$\begin{array}{rl} \textit{Types} & T ::= H \mid L \mid H[T] \mid L[T] \\ \textit{Prefixes} & \pi ::= \overline{a}\langle b\rangle \mid a(x{:}T) \\ \textit{Processes} & P ::= \pi.P \mid \mathsf{if}\ a = b\ \mathsf{then}\ P\ \mathsf{else}\ P \mid P|P \\ & \quad \mid (\nu n : T)P \mid\, !P \mid \mathbf{0} \end{array}$$

We consider the type system of [1] which prevents low channels from carrying high values. Indeed, a channel of type $L[H[T]]$ is not admitted, ensuring that confidential information is not explicitly leaked to low. A type environment Γ is a finite mapping from names and variables to types. A judgment of the form $\Gamma \vdash P$ means that the process P uses all channels as input/output devices in accordance with their types, as given in Γ.

Intuitively, the property $\mathcal{NI}(\cong_L)$ states that a process P is secure if whenever P is inserted into two different high contexts $C_H^1[\,]$ and $C_H^2[\,]$, which can use only high level channels, the two processes $C_H^1[P]$ and $C_H^2[P]$ cannot be distinguished, in the sense of $\cong_L$, by any low context $C_L[\,]$, which uses only low level channels. However, this formulation is not effective and our tool PICNIC implements a coinductive characterization of our security properties given in terms of a typed operational semantics.

Typed actions are parametrized over security levels and take the form $\Gamma \triangleright P \xrightarrow{\alpha}_\delta \Gamma' \triangleright P'$ indicating that in the type environment Γ, the process P performs the action α of level δ and evolves to P' in the possibly modified environment Γ'. Below we report the rules for output and input actions.

$$\frac{\Gamma \vdash n : \delta_1[T] \quad \delta_1 \preceq \delta}{\Gamma \triangleright \overline{n}\langle m\rangle.P \xrightarrow{\overline{n}\langle m\rangle}_\delta \Gamma \triangleright P}$$

$$\frac{\Gamma \vdash n : \delta_1[T] \quad \Gamma \vdash m : T \quad \delta_1 \preceq \delta}{\Gamma \triangleright n(x{:}T).P \xrightarrow{n(m)}_\delta \Gamma \triangleright P\{x := m\}}$$

Notice that if the channel n is low these actions are both low and high, while if n is high they are high level actions.

In the following, we write $\Longrightarrow$ for the reflexive and transitive closure of $\xrightarrow{\tau}_\delta$, and $\overset{\hat{\alpha}}{\Longrightarrow}_\delta$ for $\Longrightarrow$ if $\alpha = \tau$ and for $\Longrightarrow \xrightarrow{\alpha}_\delta \Longrightarrow$ otherwise. The coinductive characterization of $\mathcal{NI}(\cong_L)$ is based on the following notion of partial bisimilarity on low actions.

Partial bisimilarity on low actions is the largest symmetric relation $\dot{\approx}_L$, such that whenever $\Gamma \vDash P \dot{\approx}_L Q$ (i.e., P and Q are $\dot{\approx}_L$-equivalent in the type environment Γ):

(1) if $\Gamma \triangleright P \xrightarrow{\alpha}_L \Gamma' \triangleright P'$, then $\exists\, Q'$: $\Gamma \triangleright Q \overset{\hat{\alpha}}{\Longrightarrow}_L \Gamma' \triangleright Q'$ with $\Gamma' \vDash Q' \dot{\approx}_L P'$;

(2) if $\Gamma \triangleright P \xrightarrow{\alpha}_H \Gamma' \triangleright P'$ with $\alpha \in \{\overline{n}\langle m\rangle, n(m), (\nu p : H)\overline{n}\langle m\rangle, (\nu p : H)n(m)\}$, then $\exists\, Q'$: $\Gamma \triangleright Q \Longrightarrow \Gamma \triangleright Q'$ with $\Gamma' \models Q' \dot{\approx}_L P'$;

(3) if $\Gamma \triangleright P \xrightarrow{\alpha}_H \Gamma' \triangleright P'$ with $\alpha \in \{(\nu p : L)\overline{n}\langle m\rangle, (\nu p : L)n(m)\}$, then $\exists Q'$: $\Gamma \triangleright Q \Longrightarrow \Gamma \triangleright Q'$ with $\Gamma \models Q' \dot{\approx}_L (\nu p : L)P'$.

Intuitively, a process P is secure if, whenever it performs a high level action leading to a state P', it is also able to perform a number of invisible τ-steps reaching a state P'' which is equivalent to P'.

To give an intuition of the different manner to deal with high actions depending on whether they carry high or low bounded names, consider the processes $P = (\nu\ell)(\overline{h}\langle\ell\rangle.\overline{\ell}\langle\rangle.R)$ and $Q = (\nu k)(\overline{h}\langle k\rangle.\overline{k}\langle\rangle.R)$ in the type environment $\Gamma = h{:}H, k{:}H, \ell{:}L$. In both processes a name is extruded along a high level channel, then only high level contexts can receive that name and use it to synchronize on the second action. However, when the extruded name is low the high context cannot read from it, hence no context will ever interact with R. On the other hand, when the extruded name is high the high context can synchronize on the second action and possibly interact with the process R.

In [1] the authors proved that $\mathcal{NI}(\cong_L)$ coincide with the class of processes $\Gamma \triangleright P$ such that $\Gamma \models P \dot{\approx}_L P$. Moreover, $\mathcal{NI}(\cong_L)$ is compositional with respect to the language operators, e.g., if $\Gamma \triangleright P_1$ and $\Gamma \triangleright P_2$ are both in $\mathcal{NI}(\cong_L)$, then also $\Gamma \triangleright P_1|P_2$ is in $\mathcal{NI}(\cong_L)$.

In order to introduce a mechanism into the π-calculus for the secure downgrading of information, in [1] the syntax of the language has been enriched with a family of declassified actions of the form $\overline{\mathsf{dec}_L\, n}\langle m\rangle$ (declassified output on the channel n) and $\mathsf{dec}_L\, n(x : T)$ (declassified input on the channel n), where n is a high level channel. The information arising from declassified actions is allowed to flow downwards to the low level observers. As far as synchronization is concerned, a declassified action can synchronize only with its corresponding declassified co-action, i.e., both the sender and the receiver have to agree on the downgrading of information. Moreover, according to the literature, we assume that only programmers may enable the downgrading of secret information, while external entities cannot synchronize on declassified actions.

The non-interference property $\mathcal{NI}(\cong_L)$scales to the π-calculus with declassification primitives, which also inherits the coinductive proof technique. More precisely, $\mathcal{NI}(\cong_L^{\mathsf{dec}})$ identifies the class of processes $\Gamma \triangleright P$ such that $\Gamma \models P \dot{\approx}_L^{\mathsf{dec}} P$, where $\dot{\approx}_L^{\mathsf{dec}}$ is a partial bisimulation relation defined exactly as $\dot{\approx}_L$, but over a different LTS that now contains also declassified actions. The declassified actions need not to be matched by τ-steps. This capture the intuition that they cannot neither be performed by high level users (since they are controlled only by the programmers) nor be observed by the low level users (they only observe the low level flow caused by the declassification).

Differently form $\mathcal{NI}(\cong_L)$, the property $\mathcal{NI}(\cong_L^{\mathsf{dec}})$ is not compositional with respect to the parallel operator. However, compositionality is desirable to both reduce the space/time complexities and check some infinite state processes. To this end we introduce the compositional property $\mathcal{NI}(\bumpeq_L^{\mathsf{dec}})$ expressed in terms of the partial relation $\bumpeq_L^{\mathsf{dec}}$ that is a slight variation of the early weak *partial bisimilarity on low level actions* presented above. More precisely, the partial relation $\bumpeq_L^{\mathsf{dec}}$ defined exactly as $\dot{\approx}_L$ with the additional condition:

(4) if $\Gamma \triangleright P \xrightarrow{\alpha}_H \Gamma' \triangleright P'$ where α is a declassified action, i.e., α belongs to the set $\{\mathsf{dec}_L n(m),\ \overline{\mathsf{dec}_L\, n}\langle m\rangle,\ (\nu m{:}T)\ \mathsf{dec}_L\ n(m),\ (\nu m{:}T)\ \overline{\mathsf{dec}_L\ n}\langle m\rangle\}$, then $\Gamma' \models P' \bumpeq_L^{\mathsf{dec}} P'$.

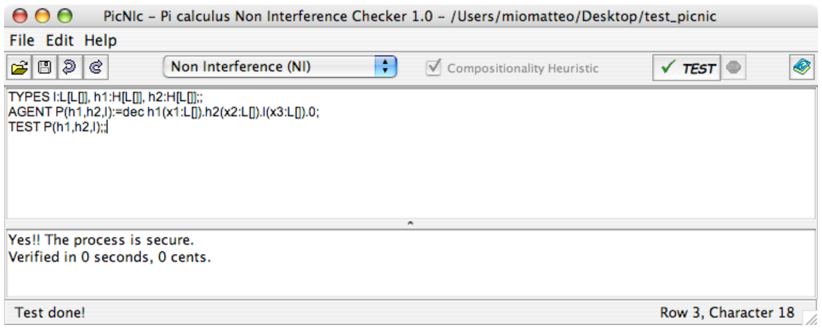
This condition checks that the process P' reached after a declassification is still a secure process, and this leads to compositionality.

We define the property $\mathcal{NI}(\bumpeq_L^{\mathsf{dec}})$ as the class of processes $\Gamma \triangleright P$ such that $\Gamma \models P \dot{\approx}_L^{\mathsf{dec}} P$. Even if there is no contextual characterization of $\mathcal{NI}(\bumpeq_L^{\mathsf{dec}})$, it provides a valuable proof technique for $\mathcal{NI}(\cong_L^{\mathsf{dec}})$. The property $\mathcal{NI}(\bumpeq_L^{\mathsf{dec}})$ is clearly stronger than $\mathcal{NI}(\cong_L^{\mathsf{dec}})$ but it enjoys the desired compositionality properties, i.e., if $\Gamma \triangleright P$ and $\Gamma \triangleright Q$ belong to $\mathcal{NI}(\bumpeq_L^{\mathsf{dec}})$, then also $\Gamma \triangleright P|Q$ and $\Gamma \triangleright !P$ do. Again, in the case of processes without declassifications, the two classes $\mathcal{NI}(\cong_L)$ and $\mathcal{NI}(\bumpeq_L^{\mathsf{dec}})$ coincide.

3. The tool PicNIc

The tool PicNIc checks the *early weak partial* relations $\dot{\approx}_L^{\mathsf{dec}}$ and $\bumpeq_L^{\mathsf{dec}}$, and as a consequence the security properties $\mathcal{NI}(\cong_L)$, $\mathcal{NI}(\cong_L^{\mathsf{dec}})$, and $\mathcal{NI}(\bumpeq_L^{\mathsf{dec}})$. The algorithms implemented in PicNIc generalize those proposed by Lin in [5] for computing bisimulations on-the-fly over the finite control π-calculus.

First, since we deal with a typed semantics, a PicNIc algorithm computes, at each step, the current type environment and checks its consistency. Then, when comparing two processes P and Q the algorithm generates a transition from P and tries to simulate it with the successors of Q. Since the observation relations checked by the tool are *weak*, the computation of Q-successors requires the computation of a *transitive closure of* τ *actions*. In the current version this computation is performed each time a process is considered; we are working on a more efficient implementation in which the τ-successors of a process are stored and reused. Notice that we cannot simply store the τ-successors of Q, but we also need to check their consistency with the current environment. As far as input transitions are concerned the algorithm instantiates the input variable with all


PicNIc – Pi calculus Non Interference Checker 1.0 – /Users/miomatteo/Desktop/test_picnic
File Edit Help
Non Interference (NI)
Compositionality Heuristic
TEST

```
TYPES l:L[L[]], h1:H[L[]], h2:H[L[]];;
AGENT P(h1,h2,l):=dec h1(x1:L[]).h2(x2:L[]).l(x3:L[]).0;
TEST P(h1,h2,l);;
```

```
Yes!! The process is secure.
Verified in 0 seconds, 0 cents.
```

Test done! Row 3, Character 18

Figure 1. PicNIc **screen-shot.**

the free names of the process and with a symbolic new name. Finally, the algorithm keeps track of the already visited nodes and of the pairs of nodes assumed to be bisimilar to guarantee termination over finite state processes. Some basic optimizations have been introduced in the implementation to reduce the branching when actions visible at both high and low levels are considered.

To assess the time complexity of our algorithms, we recall that the complexity of Lin's algorithm for strong bisimulation is bounded by $O(N^2M^2)$, where N and M are the number of states in the symbolic LTS's of the analyzed processes. Hence, due to the τ-successors computations, our bound for time complexity to check both $\Gamma \vDash P \cong_L^{\mathsf{dec}} P$ and $\Gamma \vDash P \approx_L^{\mathsf{dec}} P$ is $O(N^6)$, where N is the size of the symbolic LTS of P.

In order to improve efficiency, our implementation exploits the compositionality of $\mathcal{NI}(\approx_L^{\mathsf{dec}})$ with respect to the parallel composition and replication operators. Indeed, to prove that $P|Q$ is secure we can first check the security of P and Q, and then if these tests are positive we can conclude that $P|Q$ is secure. These considerations drastically reduce the size of the symbolic LTS's that we need to consider and allow us to verify also some infinite state processes.

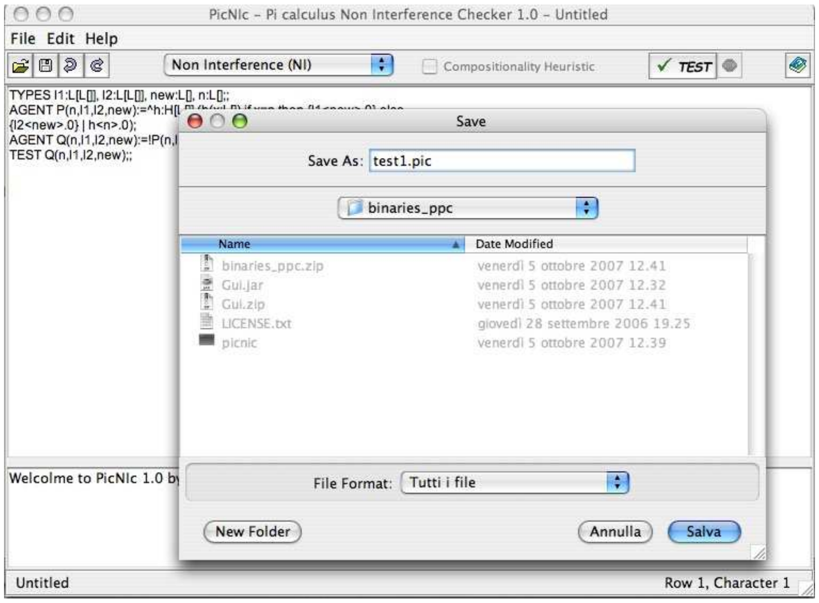
To model recursive processes PicNIc allows also the use of recursive definitions with constants occurring on both the left and right hand sides of a definition. Recursive definitions are not present in the syntax described in the previous section and their introduction does not augment the expressive power of the language. However, in many cases, their use simplify the modeling phase.

A screen-shot of the tool is shown in Figure 1: a process has been typed in the edit pane on the top and the compositionality heuristic has been selected. The process is secure as shown in the log panel on the bottom.

The kernel of PicNIc is written in Fresh Objective Caml [9], a programming language extending Objective Caml with a special support for data-types with binders, based on recent developments on the so called *nominal syntax* [3]. More precisely, Fresh O'Caml offers specific *types of names* for representing object-level bindable names (such as names or channels of the π-calculus); *abstraction expressions* for representing object-level binding (such as input prefixes, or restrictions), and *pattern-matching* for deconstructing abstraction values. Notably, two α-equivalent terms are *automatically equated* by the language; e.g., the terms `(in c [x](out c x))` and `(in c [y](out c y))` (representing the π-terms $c(x).\bar{c}\langle x\rangle.0$ and $c(y).\bar{c}\langle y\rangle.0$) are considered equal. Also α-conversion is automatically performed, to avoid name clashing/capture. Still, we can make pattern matching and recursion over objects of nominal data-types. Hence, using Fresh O'Caml the programmer is freed from the burden of implementing α-conversions, or to work with awkward representation of binders such as de Bruijn indexes, without renouncing to performance. In fact, supported by our positive experience in effectiveness, easiness, and efficiency, we can definitely assert that languages with native support for binders, such as Fresh O'Caml, should be the natural choice for implementing algorithms dealing with the π-calculus and similar calculi.

Some Installation and Use Instructions. The PicNIc web page contains several pre-compiled versions. The sources of the kernel are also available; they can be compiled with Fresh O'Caml which is available at `http://www.fresh-ocaml.org/`. The compiled GUI (a jar file) should work well in every Java compatible system. The GUI contains some working examples which are available in the Syntax Help window. The GUI offers two text areas. The largest one is the edit pane where π-calculus processes can be either typed or loaded from a file or saved into a file (Figure 2). The smallest one, at the bottom, is used to display the output.

The menu bar on the top allows one to select either $\mathcal{NI}(\cong_L^{\mathsf{dec}})$ or $\mathcal{NI}(\approx_L^{\mathsf{dec}})$, to enable the compositionality heuristic (that works only in the case of $\mathcal{NI}(\approx_L^{\mathsf{dec}})$), to start the verification pressing the button "TEST", and to stop the computation. In particular, the compositionality heuristic



```
TYPES l1:L[L[]], l2:L[L[]], new:L[], n:L[];;
AGENT P(n,l1,l2,new):=^h:H[L
{l2<new>.0} | h<n>.0);
AGENT Q(n,l1,l2,new):=!P(n,l
TEST Q(n,l1,l2,new);;
```

Save As: test1.pic

binaries_ppc

Name	Date Modified
binaries_ppc.zip	venerdì 5 ottobre 2007 12.41
Gui.jar	venerdì 5 ottobre 2007 12.32
Gui.zip	venerdì 5 ottobre 2007 12.41
LICENSE.txt	giovedì 28 settembre 2006 19.25
picnic	venerdì 5 ottobre 2007 12.39

File Format: Tutti i file

Welcolme to PicNIc 1.0 b

Figure 2. The Editor panel.

exploits the compositinal property $\mathcal{NI}(\approx_L^{\mathsf{dec}})$ allowing one to reduce, in many cases, the search space and the time needed for the verification process. It allows also to verify some infinite state processes. The Settings option in the Edit menu allows one to (1) set the path of the kernel, (2) set the state space limit (i.e., the maximum number of π-calculus states to be visited) and (3) view the successors of the process under testing to better understand the process behaviour. The Help menu contains the Syntax Help item. It can be used to view the PicNIc syntax grammar and some working examples. More detailed instructions and suggestions can be found in the tool's site.

4. Two Illustrating Examples

Consider the following communication protocol in which an agent A intending to send a message m to B refers to a system S to create a communication channel with B. In particular, A sends to S over a private channel cas the name of the new channel cab. The system S sends to B over a private channel cbs the name cab. The process B receives cab and waits for a message over it. The process A sends to B the message m over the channel cab. Since the channels cas and cbs with a trusted system S are private we model them as high level channels, while we imagine that the new channel cab and the message m are low. Using the syntax of PicNIc the protocol can be modeled as:

```
TYPES cas:H[L[L[]]], cab:L[L[]],
      cbs:H[L[L[]]], m:L[], n:L[];;
AGENT A(cas,m):=
      ^cab:L[L[]].(cas<cab>.cab<m>);
AGENT S(cas,cbs):= cas(x:L[L[]]).cbs<x>.0;
AGENT B(cbs,n):=cbs(x:L[L[]]).x(y:L[]);
AGENT P(m,n,cas,cbs):=
      A(cas,m)|S(cas,cbs)|B(cbs,n);
AGENT Q(m,n):=^cas:H[L[L[]]].^cbs:H[L[L[]]].
               P(m,n,cas,cbs);
```

The agent P(m,n,cas,cbs) is not secure, since the high level channels cas and cbs are visible to possible malicious high level processes which can use them to modify the low level behaviour of P(m,n,cas,cbs). On the other hand, the process Q(m,n) is secure since cas and cbs can be used only by A, B, and S.

As a second example, consider a simplified version of a protocol in which a client sends its data to the bank (e.g., the pin code); if the data are correct the bank sends a positive acknowledgement together with the money to the client, otherwise it sends a negative answer and no money. We start by assuming that all the involved channels are high, but the acknowledgement one.

```
TYPES ck:H[H[]], id:H[],ack:L[L[]], money:H[L[]],
      ten:L[], zer:L[], trans:H[L[]],
      ok:L[], no:L[];;
AGENT Client(ck,id,ack,money):=
      ck<id>.ack(x:L[]).money(y:L[]).0;
AGENT Bank(ck,id,ack,ok,no,trans,money,ten,zer):=
      ck(z:H[]).if z=id then {ack<ok>.trans<ok>.0}
                 else {ack<no>.trans<no>.0}|
      trans(w:L[]).if w=ok then {money<ten>.0}
                 else {money<zer>.0};
AGENT System(ck,id,ack,ok,no,trans,money,ten,zer):=
      Bank(ck,id,ack,ok,no,trans,money,ten,zer)|
      Client(ck,id,ack,money);
TEST System(ck,id,ack,ok,no,trans,money,ten,zer);;
```

The agent System(ck,id,ack,ok,no,trans,money, ten,zer) is not secure. However, if we replace both the occurrences of ck with dec ck we get a secure system.

5. Comparisons and Time Performance

In the literature there are other tools that compute bisimulation relations in the π-calculus, namely, the MOBILITY WORKBENCH (`http://www.it.uu.se/research/group/mobility/mwb`) and the ABC (`http://lamp.epfl.ch/~sbriais/abc/abc.html`) tools. They both check *open* bisimulation equivalence relations; differently, our security properties are based on weak *early* bisimulation relations.

Another related tool is MIHDA (`http://www.cs.le.ac.uk/people/et52`) which aims at reducing the finite control π-calculus processes with respect to the early bisimulation using a partition refinement algorithm. We noticed that PICNIC is always faster than MIHDA on our test-cases (e.g., MIHDA on 8 copies of the process P4 in the table below terminates after 2.964 seconds). However, the purpose of Mihda is that of computing the labeled transition system of the reduced process. Hence, it cannot stop the computation when two non-bisimilar states are found.

COSEC [2] and COPS (`http://www.dsi.unive.it/~mefisto/CoPS`) are two tools that check non-interference security properties for CCS processes. The security property $\mathcal{NI}(\cong_L)$ is a generalization to the π-calculus of the P_BNDC property defined over CCS and checked by both COSEC and COPS, while the property $\mathcal{NI}(\approx_L^{\mathsf{dec}})$ is an extension to the π-calculus of the CCS property DP_BNDC. Finally the tool COPS [7] checks a property similar to $\mathcal{NI}(\cong_L)$ over the set of CCS processes exploiting Paige-Tarjan bisimulation algorithm [6]. Some comparisons between COSEC and COPS can be found in [7].

Interestingly, we run some benchmarks to compare the time performance of PICNIC and COPS. We considered five simple processes belonging to both CCS and π-calculus (i.e., we used only the prefix operator and the recursion one) and we composed them through parallel composition to build larger processes. In both tools we disabled the compositionality heuristics. The results of these tests are reported in tables below. We can notice that in the case of insecure processes PICNIC outperforms COPS, while, as far as secure processes is concerned, PICNIC is slower than COPS only in one case. Indeed, since COPS exploits the Paige-Tarjan bisimulation algorithm [6], it always builds the complete LTS of the process before starting the bisimulation computation. Hence, it has the same performance both over insecure processes (P1 and P2) and over secure processes (P3 and P4). On the other hand, PICNIC using an on-the-fly algorithm does not generate the whole LTS. This fact in the case of insecure processes allows to save time when transitions which cannot be simulated are found early during the computation.

Process	CCS	π-calculus
P1	$h.l.0$	$h\langle new\rangle.l\langle new\rangle.0$
P2	$h.l.$P2	$h\langle new\rangle.l\langle new\rangle.$P2
P3	$h.h.0$	$h\langle new\rangle.h\langle new\rangle.0$
P4	$h.h.$P4	$h\langle new\rangle.h\langle new\rangle.$P4
P5	$l.h.0$	$l\langle new\rangle.h\langle new\rangle.0$

where l, new are low level actions (names); h is high

Process	#Nodes	#Edges	CoPS	PICNIC
10 copies P1	2,047	18,434	44.42 s	0.06 s
60 copies P1	–	–	$>$ 5 m	22.86 s
11 copies P2	2,049	22,539	45.61 s	4.57 s
13 copies P2	8,193	106,509	$>$ 5 m	21.77 s
8 copies P3	511	3586	1.46 s	8.80 s
8 copies P4	1,050	10,250	6.81 s	0.15 s
10 copies P5	2,047	18,434	50.69 s	1.18 s

Finally, we mention the tool PIET (`http://piet.sourceforge.net/`) developed by Matteo Mio. PIET allows one to verify ten different bisimulations on the π-calculus. PIET and PICNIC works on different calculi and different semantics, they only share the idea of implementing on-the-fly bisimulation algorithms in FRESH O'CAML.

References

[1] S. Crafa and S. Rossi. P-Congruences as Non-Interference for the Pi-calculus. In *Proc. of the ACM work. on Formal Meth. in Security Engineering (FMSE'06)*, pages 13–22, 2006.

[2] R. Focardi and R. Gorrieri. The Compositional Security Checker: A Tool for the Verification of Information Flow Security Properties. *IEEE Transactions on Software Engineering*, 23(9):550–571, 1997.

[3] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2002.

[4] J. A. Goguen and J. Meseguer. Security Policies and Security Models. In *Proc. of the IEEE Symp. on Security and Privacy (SSP'82)*, pages 11–20. IEEE Computer Society Press, 1982.

[5] H. Lin. Computing Bisimulations for Finite-Control π-Calculus. *Journal of Computer Science and Technology*, 15(1):1–9, 2000.

[6] R. Paige and R. E. Tarjan. Three Partition Refinement Algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[7] C. Piazza, E. Pivato, and S. Rossi. Cops - Checker of Persistent Security. In *Proc. of Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*, volume 2988 of *LNCS*, pages 144–152. Springer, 2004.

[8] A. Sabelfeld and D. Sands. A Per Model of Secure Iinformation Flow in Sequential Programs. In *Proc. of European Symposium on Programming (ESOP'99)*, volume 1576 of *LNCS*, pages 40–58. Springer-Verlag, 1999.

[9] M. R. Shinwell. Fresh o'caml: Nominal abstract syntax for the masses. *Electr. Notes Theor. Comput. Sci.*, 148(2):53–77, 2006.