A Flow-Sensitive Analysis of Privacy Properties

Nielson, Hanne Riis; Nielson, Flemming

Published in:
Proceedings of Computer Security Foundations Symposium, CFS 2007

Link to article, DOI:
10.1109/CSF.2007.4

Publication date:
2007

Document Version
Publisher's PDF, also known as Version of record

A flow-sensitive analysis of privacy properties

Hanne Riis Nielson and Flemming Nielson
Technical University of Denmark
{riis,nielson}@imm.dtu.dk

Abstract

In this paper we consider service oriented architectures where many components interact with one another using a wireless network. We are interested in questions like:

- *Can I be sure that I do not get unsolicited information from some service? — unless I give my permission?*
- *Can I be sure that information I send to some service never is leaked to another service? — unless I give my permission?*

We shall develop a static program analysis for the π-calculus and show how it can be used to give privacy guarantees like the ones requested above. The analysis records the explicit information flow of the system and keeps track of, not only the potential configurations of the system, but also the order in which they may be encountered.

1. Introduction

In recent years, static program analyses techniques have been developed for analysing security properties for systems expressed in process calculi. The focus has been on analysing the *configurations* of the systems and control flow analyses techniques have played a major role. They have been *flow-insensitive* as well as *context-insensititive* and thus, from a static analysis point of view, the techniques have been fairly simple. Nonetheless, from a security point of view, surprisingly powerful results have been obtained. Examples include analyses of classical access control mechanisms [3], confidentiality [12], information flow [6, 13], firewalls in mobile settings [8] and security protocols [1, 2].

It is characteristic for these analyses that they produce a *single* abstract configuration approximating *all* the concrete configurations of a given system. The security property of interest has then been validated with respect to this abstract configuration: if the abstract configuration could not exhibit the unwanted behaviour, then none of the concrete configurations could exhibit it either.

However, there are cases where this is not enough. To provide guarantees like the ones hinted at in the Abstract, it seems that several abstract configurations are needed in order to distinguish between whether or not certain actions have been performed; also, it seems that we would need information about the order in which these configurations might be encountered when the system is running.

In this paper we develop such an analysis for the π-calculus [4]: given a process it will construct a *finite automaton*, where the states abstract the potential configurations of the system, and the transitions of the automaton faithfully reflect the semantics of the process. We shall show how our analysis can be used to validate privacy properties like the ones mentioned in the Abstract.

The scenario. To illustrate the development of the analysis we shall consider the scenario of Figure 1. It is inspired by a communication device developed for military purposes by the Danish company Mærsk Data Defence [14]; we have reformulated it in a civilian setting and it captures now the essence of one of the case studies of the SENSORIA project[1].

A *car* is equipped with an info-system consisting of a gps device and a message board. The gps device continuously emits the position of the car on the channel gps; this may be for use by a car rental company or an insurance company. At the same time the driver can register with a news agent and obtain information (news, ads, etc.) on the info channel. The registration is performed by sending a password over the login channel. To be a little more concrete, let us specify the processes as follows in the π-calculus:

gps device:	GPS	$\triangleq$	$(\nu\,\mathsf{pos})\,\mathsf{gps!pos.\ GPS}$
message board:	MB	$\triangleq$	login!pwd. info?x. MB

The *service centre* has a gps logger and a news agent. The gps logger does nothing but receive car positions on the

[1] SENSORIA is an EU Integrated Project (IST-2005-016004) funded as part of the 6'th Framework Programme.

20th IEEE Computer Security Foundations Symposium (CSF'07)
0-7695-2819-8/07 $20.00 © 2007 IEEE

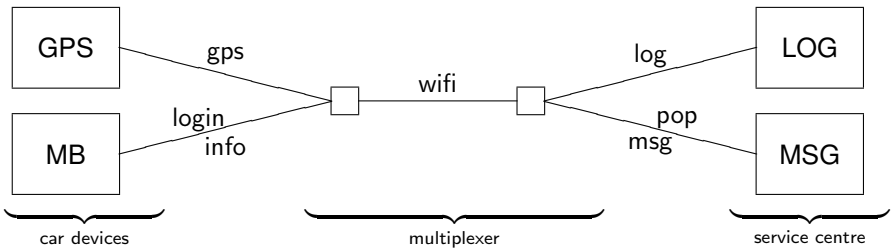


Figure 1. Info-system.

channel log. We shall consider different versions of the news agent; the one specified below requires that the user goes through a login process using the pop channel before any messages are transmitted to him on the msg channel:

$$\begin{array}{lrcl} \text{gps logger:} & \mathsf{LOG} & \triangleq & \mathsf{log?y.\ LOG} \\ \text{news agent:} & \mathsf{NA} & \triangleq & \mathsf{pop?z.} \\ & & & (\nu\,\mathsf{news})\,\mathsf{msg!(news, z).\ NA} \end{array}$$

The car and the service centre are connected using a *wireless network*; both the car and the service centre have a multiplexer facilitating this. We shall specify it as follows:

$$\begin{array}{lrcl} \text{multiplexer:} & \mathsf{MP} & \triangleq & \mathsf{gps?u.\ wifi!(log, u).\ MP} \\ & & + & \mathsf{login?u.\ wifi!(pop, u).\ MP} \\ & & + & \mathsf{msg?(u, v).\ wifi!(info, u).\ MP} \\ & \mathsf{MP'} & \triangleq & \mathsf{wifi?(u, v).\ u!v.\ MP'} \end{array}$$

The process MP specifies how a message received from one of the services in the car or the service centre are transmitted over the wifi channel by encoding the intended recipient by a unique name in the first component of the message. The process MP′ shows how a message received on the wifi channel is redirected to the intended recipient using the first part of the message as a channel over which the second component of the message is transmitted. Thus the overall system illustrated on Figure 1 can be specified by:

$$\mathsf{GPS} \mid \mathsf{MB} \mid \mathsf{MP} \mid \mathsf{MP'} \mid \mathsf{LOG} \mid \mathsf{NA}$$

The security problems. The driver of the car will be interested in protecting his privacy:

- **P1:** He does not want to receive unsolicited ads from the news agent; in particular, he only want to obtain messages *after* he has logged in at the agent.
- **P2:** He only wants his position to be communicated to the gps logger; in particular, it should *never* be revealed to the news agent (where it may be misused by, e.g., paparazzi photographers).
- **P3:** He might, however, choose to disclose his position to the news agent in order to obtain certain benefits (e.g., local news); then he wants to ensure that his position is *only* used for that purpose.

Thus the security of the system relies not only on the correct encoding and subsequent separation and redirection of the information in the multiplexer but also on the order in which the individual actions are performed. The analyses mentioned previously will not be able to capture this. The analysis introduced in this paper presents a first step in rectifying this.

Our results. For inspiration we shall look to data flow analysis [7] where control flow analysis is only a minor and preparatory step in obtaining the information of interest. Flow-sensitivity is often modelled using *Monotone Frameworks*, where each basic block will kill some of the information of interest, while at the same time it will generate some new information. Traditionally formulated for bit-vectors it applies equally well to complete lattices. According to Tarski's fixed point theorem, termination is ensured if the complete lattice satisfies the ascending chain property but there are techniques that apply even if this is not the case: termination can be ensured by utilising the framework of *Abstract Interpretation* and defining a suitable widening operator; this is an upper approximation to the least upper bound operation on the complete lattices that additionally ensures that no infinite ascending chains can be constructed.

The aim of our analysis is to construct a *finite automaton* that faithfully models the potential infinite transition system of the π-calculus process of interest. Each of the states in the automaton will capture two aspects of a configuration of the process:

- a multiset of actions ready for execution, and
- the potential bindings of the names.

The transitions of the automaton approximate the potential transitions of the process, so in particular the analysis will

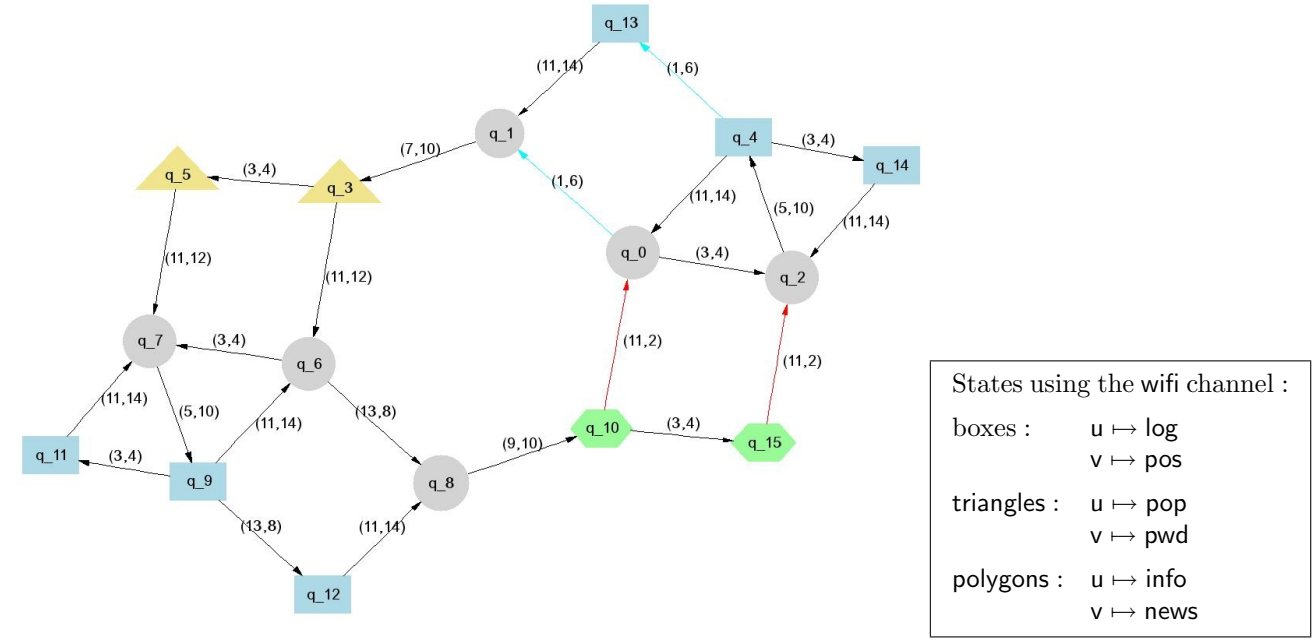


Figure 2. Analysis of the info-system: MB | GPS | MP | MP′ | LOG | NA.

be *flow-sensitive*. It will also be *context-sensitive* because information about the bindings is localised in the states.

To illustrate this consider the automaton of Figure 2 constructed for the process MB | GPS | MP | MP′ | LOG | NA specified above. It shows that the system can be in sixteen different abstract states. The edges connecting the states represent the potential transitions of the system; each edge is annotated with pairs of labels referring to the actions taking part in the actual transition. The initial state q_0 corresponds to the overall process itself. As we shall see later it records that *any* of the eight actions login!pwd, gps!pos, gps?u, login?u, msg?(u, v), wifi?(u, v), log?y and pop?z are ready to be executed. The two outgoing edges from q_0 show that only two transitions are indeed possible from the state: The edge labelled (1,6) captures that the action login!pwd (to be labelled 1 later) of MB may interact with the action login?u (to be labelled 6) of MP and the resulting state will be q_1. Similarly, the edge labelled (3,4) captures that the action gps!pos (to be labelled 3) of GPS may interact with the action gps?u (to be labelled 4) of MP and the resulting state will be q_2. The states captures not only which actions that are ready to be executed but also the potential bindings to the names so as a result of the communication the state q_1 will record that u might be bound to pwd whereas q_2 will record that u might be bound to pos.

We shall use the automaton to validate the security properties of interest. Property **P1** is concerned with the order in which some actions are performed and it amounts to checking whether all the paths in the automaton satisfy a certain property. Property **P2** is concerned with the flow of information and it amounts to checking that all the abstract states disallow certain bindings of names. Finally, property **P3** requires that certain actions are performed before a specific flow of information is possible and thus amounts to a combination of checking the paths of the automaton and the bindings captured in some of the states. Due to the way the automaton is constructed, the security analysis will only be concerned with *explicit* information flow.

Overview of the paper. After having reviewed the π-calculus and its semantics in Section 2 we shall present our analysis in three stages. The actions of the π-calculus are going to play the role of the basic blocks in classical data flow analysis, so in Section 3 we start by introducing the complete lattice of the properties of interest, namely *extended multisets of exposed actions*. The next step is then to specify how this information is modified when an action is executed; in the classical setup this is given by *transfer functions* of the form

$$\mathsf{transfer}_{block}(E) = (E \setminus \mathsf{kill}_{block}) \cup \mathsf{gen}_{block}$$

where kill_{block} is the information that is no longer valid after the block has been executed and gen_{block} is the information that becomes valid upon execution of the block. We shall need analogues of the kill and generate functions in our setup; in doing so we shall *under-approximate* the infor-

mation to be killed and *over-approximate* the information to be generated.

In the second stage, to be presented in Section 4, we construct the appropriate *transfer functions* corresponding to the individual actions. The transfer functions are concerned with not only the extended multisets of exposed actions but also the *bindings of the names*. Obviously, these have to be modified in the case of communication but further precision can be gained when names are matched against one another.

In Section 5 we present the final stage of the analysis, the construction of the automaton. We shall use a simple *work-list algorithm* that, starting from the initial state of the automaton, gradually extends it with more states and transitions. Finiteness of the automaton is achieved using an appropriate *widening operator* together with a *granularity function* that additionally can be used to control the precision of the analysis.

In Section 6 we shall use our analysis to validate a number of interesting security properties for variations of the scenario above.

- **P1:** Sometimes it must be ensured that certain messages are only received after a certain login activity has taken place. — Our analysis will be flow sensitive and will therefore correctly model the sequential order of the actions.
- **P2:** Sometimes it must be ensured that messages only reach the intended recipients. — Our analysis will be context sensitive and will therefore correctly track the flow of information.
- **P3:** Sometimes it is allowed to "declassify" information for a certain purpose. — Our analysis will be both flow sensitive and context sensitive and will therefore accurately model that a certain flow of information takes place after a given action.

Finally, in Section 7 we give our concluding remarks.

2. Review of the π-calculus

The syntax of *processes* P and *actions* α is given by [4]:

$$\begin{array}{rcl} P & ::= & (\nu\, x)\, P \mid P_1 \mid P_2 \mid \Sigma_{i\in I}\alpha_i.P_i \mid A \\ \alpha & ::= & \tau \mid [x{=}y] \mid x!\vec{y} \mid x?\vec{y} \end{array}$$

Here $(\nu\, x)\, P$ introduces the new name x with scope P, parallel composition is modelled using the construct $P_1 \mid P_2$ whereas summations are of the form $\Sigma_{i\in I}\alpha_i.P_i$. Here I is a finite index set and α_i takes one of four forms: either it is a silent action, a match of two names, a polyadic output action or a polyadic input action. If the index set I is empty the summation is simply written as $\mathsf{0}$. The construct A allows us to refer to an instance of a recursively defined process of the form $A \triangleq P$. Finally, a *program* consists of a sequence $A_1 \triangleq P_1, \cdots, A_k \triangleq P_k$ of such definitions together with the main process $P_\star$.

- If Q is obtained from P by alpha-renaming then $P \equiv Q$.
- The Abelian monoid laws hold for parallel:
  - $P \mid Q \equiv Q \mid P$,
  - $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$ and
  - $P \mid \mathsf{0} \equiv P$.
- Summands can be permuted in $\Sigma_{i\in I}\alpha_i.P_i$.
- Recursion can be unfolded.
- The scope extension laws:
  - $(\nu\, x)\, \mathsf{0} \equiv \mathsf{0}$.
  - $(\nu\, x)\, (P \mid Q) \equiv P \mid (\nu\, x)\, Q$ if $x \notin \mathsf{fn}(P)$.
  - $(\nu\, x)\, ([y{=}z]P) \equiv [y{=}z]((\nu\, x)\, P)$ if $x \notin \{y, z\}$.
  - $(\nu\, x)\, (\nu\, y)\, P \equiv (\nu\, y)\, (\nu\, x)\, P$.

Table 1. Structural congruence $P \equiv Q$.

$$\frac{P \rightarrow^{\alpha}_{\tilde{\ell}} Q}{P \mid P' \rightarrow^{\alpha}_{\tilde{\ell}} Q \mid P'} \quad \text{if } \mathsf{bn}(\alpha) \cap \mathsf{fn}(P') = \emptyset$$

$$\frac{P \rightarrow^{x!\vec{y}}_{\ell_1} Q \quad P' \rightarrow^{x?\vec{z}}_{\ell_2} Q'}{P \mid P' \rightarrow^{\tau}_{\ell_1\ell_2} Q \mid Q'[\vec{y}/\vec{z}]} \quad \text{if } |\vec{y}|{=}|\vec{z}|$$

$$\frac{P \rightarrow^{\alpha}_{\tilde{\ell}} P'}{(\nu\, x)\, P \rightarrow^{\alpha}_{\tilde{\ell}} (\nu\, x)\, P'} \quad \text{if } x \notin \mathsf{fn}(\alpha)$$

$$\frac{P \equiv P' \quad P' \rightarrow^{\alpha}_{\tilde{\ell}} Q' \quad Q' \equiv Q}{P \rightarrow^{\alpha}_{\tilde{\ell}} Q}$$

$$\Sigma_{i\in I}\alpha_i^{\ell_i}.P_i \rightarrow^{\alpha_i}_{\ell_i} P_i \qquad [x{=}x]^{\ell}P \rightarrow^{\tau}_{\ell} P$$

Table 2. Operational semantics $P \rightarrow^{\alpha}_{\tilde{\ell}} Q$.

To facilitate the analysis we shall annotate the syntax in two ways: each action will be labelled by an element $\ell \in \mathbf{Lab}$ (as in α^{ℓ}) and each (defining and applied) occurrence of a name will get an index $\imath \in \mathbf{Index}$ (as in $x^{\imath}$) that identifies its defining occurrence. It is important to stress that the annotations only serve as pointers into the syntax; they have no semantic significance whatsoever but will play a major role in the analysis. For the sake of simplicity we shall assume that the annotations of the initial program are unique: all actions are uniquely labelled and similarly all free names and all defining occurrences of names have unique indices. This assumption simplifies the development although it will not be preserved by the semantics.

Following [4, 5] the semantics is specified by a structural

20th IEEE Computer Security Foundations Symposium (CSF'07)
0-7695-2819-8/07 $20.00 © 2007 IEEE

congruence and a transition relation. The structural congruence $P \equiv Q$ is defined as the least congruence generated from the axioms of Table 1. We shall arrange that the indices are left unchanged by the structural congruence — reflecting that they are nothing but pointers into the syntax. In particular this means that alpha-renaming does not modify the indices and neither do substitutions. Similarly, the labels are left unchanged by rearrangements of the structural congruence.

The transition relation takes the form $P \rightarrow^{\alpha}_{\tilde{\ell}} Q$ where $\tilde{\ell}$ lists the labels of the actions involved in the step (which may be one or two depending on the step). The details are given in Table 2. Some of the finer details of the semantics are motivated by the analysis we want to perform, in particular we want the analysis to track silent actions as well as match actions so they are visible actions in the semantics.

3. Step 1: Exposed actions

An *exposed action* is an action that *may* participate in the next interaction. To illustrate this consider the following versions of the message board of the running example:

$$\mathsf{MB}_1 \triangleq \mathsf{login!pwd}^1.\ \mathsf{MB}_1 + \mathsf{info?x}^2.\ \mathsf{MB}_1$$

$$\mathsf{MB}_2 \triangleq \mathsf{login!pwd}^1.\ \mathsf{info?x}^2.\ \mathsf{MB}_2$$

$$\mathsf{MB}_3 \triangleq \mathsf{MB}_2 \mid \mathsf{MB}_2$$

$$\mathsf{MB}_4 \triangleq (\mathsf{login!pwd}^1.\ \mathsf{info?x}^2.\ 0) \mid \mathsf{MB}_4$$

MB_1 can engage in communications over the login channel as well as the info channel and we shall say that one occurrence of each of the actions labelled 1 and 2 are exposed. MB_2, on the other hand, may only engage in communications over the login channel so it has one exposed occurrence of the action labelled 1 and none of the action labelled 2. Also MB_3 is willing to communicate over the login channel but it can do so in two different ways; hence we shall record that two occurrences of label 1 are exposed. Finally MB_4 will have infinitely many exposed occurrences of label 1 since it can be unfolded into any number of parallel compositions of the process $\mathsf{login!pwd}^1.\ \mathsf{info?x}^2.\ 0$.

To capture this we define an *extended multiset* M as an element of:

$$\mathfrak{M} = \mathbf{Lab} \rightarrow \mathbb{N} \cup \{\infty\}$$

The idea is that $M(\ell)$ records the number of occurrences of ℓ; there may be a finite number in which case $M(\ell) \in \mathbb{N}$ or an infinite number in which case $M(\ell) = \infty$. We shall equip $\mathfrak{M}$ with a partial ordering $\leq_{\mathfrak{M}}$ defined by $M \leq_{\mathfrak{M}} M'$ iff $\forall \ell : M(\ell) \leq M'(\ell) \vee M'(\ell) = \infty$. This turns $(\mathfrak{M}, \leq_{\mathfrak{M}})$ into a *complete lattice* with least element $\bot_{\mathfrak{M}}$ given by $\forall \ell : \bot_{\mathfrak{M}}(\ell) = 0$ and largest element $\top_{\mathfrak{M}}$ given by $\forall \ell : \top_{\mathfrak{M}}(\ell) = \infty$. The least upper bound and greatest lower bound operators of $\mathfrak{M}$ are denoted $\sqcup_{\mathfrak{M}}$ and $\sqcap_{\mathfrak{M}}$, respectively, and they are defined by a pointwise extension of the generalised maximum and minimum operators on $\mathbb{N} \cup \{\infty\}$. Note that $(\mathfrak{M}, \leq_{\mathfrak{M}})$ has infinite ascending chains.

In the following we write $M[\ell \mapsto n]$ for the extended multiset that is as M except that ℓ is mapped to $n \in \mathbb{N} \cup \{\infty\}$ and we write $\mathsf{dom}(M)$ for the set $\{\ell \mid M(\ell) \neq 0\}$. In examples we shall often write an extended multiset M by listing the elements of its domain with their counts as superscripts; as an example, we write $\{\ell^1, \ell'^{\infty}\}$ for $\bot_{\mathfrak{M}}[\ell \mapsto 1, \ell' \mapsto \infty]$. Finally, the pointwise addition and subtraction operators of extended multisets are denoted $+_{\mathfrak{M}}$ and $-_{\mathfrak{M}}$, respectively.

To formalise the notion of exposed actions of processes discussed above we now introduce the function:

$$\mathcal{E}_\star : \mathbf{Proc} \rightarrow \mathfrak{M}$$

In the case of non-recursive processes it is defined by:

$$\begin{aligned} \mathcal{E}_\star[\![(\nu\, x)\, P]\!] &= \mathcal{E}_\star[\![P]\!] \\ \mathcal{E}_\star[\![P \mid P']\!] &= \mathcal{E}_\star[\![P]\!] +_{\mathfrak{M}} \mathcal{E}_\star[\![P']\!] \\ \mathcal{E}_\star[\![\Sigma_{i\in I}\alpha_i^{\ell_i}.P_i]\!] &= +_{\mathfrak{M}i\in I} \bot_{\mathfrak{M}}[\ell_i \mapsto 1] \end{aligned}$$

Here we use the addition operator $+_{\mathfrak{M}}$ to add up the individual contributions from subprocesses both in the case of parallel composition and in the case of sums. In the latter case we simply record that one occurrence of each of the actions $\alpha_i^{\ell_i}$ is exposed while we ignore the continuation processes P_i. The definition is extended to recursive processes using a fixed point formulation.

Example 1 *Consider the following labelling of the running example*[2]*:*

$$\begin{aligned} \mathsf{MB} &\triangleq \mathsf{login!pwd}^1.\ \mathsf{info?x}^2.\ \mathsf{MB} \\ \mathsf{GPS} &\triangleq (\nu\, \mathsf{pos})\mathsf{gps!pos}^3.\ \mathsf{GPS} \\ \mathsf{MP} &\triangleq \mathsf{gps?u}^4.\ \mathsf{wifi!}(\mathsf{log},\mathsf{u})^5.\ \mathsf{MP} \\ &\quad + \mathsf{login?u}^6.\ \mathsf{wifi!}(\mathsf{pop},\mathsf{u})^7.\ \mathsf{MP} \\ &\quad + \mathsf{msg?}(\mathsf{u},\mathsf{v})^8.\ \mathsf{wifi!}(\mathsf{info},\mathsf{u})^9.\ \mathsf{MP} \\ \mathsf{MP}' &\triangleq \mathsf{wifi?}(\mathsf{u},\mathsf{v})^{10}.\ \mathsf{u!v}^{11}.\ \mathsf{MP}' \\ \mathsf{NA} &\triangleq \mathsf{pop?z}^{12}.\ (\nu\, \mathsf{news})\mathsf{msg!}(\mathsf{news},\mathsf{z})^{13}.\ \mathsf{NA} \\ \mathsf{LOG} &\triangleq \mathsf{log?y}^{14}.\ \mathsf{LOG} \end{aligned}$$

The main program is $\mathsf{MB} \mid \mathsf{GPS} \mid \mathsf{MP} \mid \mathsf{MP}' \mid \mathsf{LOG} \mid \mathsf{NA}$.

The exposed actions are $\{1^1, 3^1, 4^1, 6^1, 8^1, 10^1, 12^1, 14^1\}$. *If we replace* MB *with* MB_4 *as discussed above we get* $\{1^\infty, 3^1, 4^1, 6^1, 8^1, 10^1, 12^1, 14^1\}$ *instead.*

[2] For the sake of readability, we have omitted the indices on names as they will not be used in this section; they will be inserted in Example 4.

The set of exposed labels gives a finite handle on a process that is somewhat similar to the set of observable names of a process [5, 11]. To be specific, the set of exposed labels indicate all inputs, outputs or tests that might be involved in the next reduction; the set of observable names on the other hand indicate all the names of channels over which input or output could take place in the next reduction.

3.1. Generated actions

Once an action has been executed, some other actions will become exposed and others will cease to be exposed. As an example, consider the process MP of Example 1. Once action 4 has been executed, neither it nor actions 6 or 8 will be exposed any longer but action 5 will become exposed. We shall say that actions 4, 6 and 8 are *killed* whereas action 5 is *generated*. Similarly, when action 5 is executed it will kill itself but the actions 4, 6 and 8 will be generated.

The aim now is to construct an *over-approximation* of the (extended multiset of) actions generated by an action and (in Subsection 3.2) an *under-approximation* of the (extended multiset of) actions killed by an action. The killed actions will be *removed* from the current multiset of exposed actions and when we have an under-approximation we ensure that we err on the safe side. The generated actions will be *added* to the current multiset of exposed actions so here we shall ensure that we have an over-approximation in order to err on the safe side.

The domain of interest has the functionality:

$$\mathfrak{T} \quad = \quad \mathbf{Lab} \to \mathfrak{M}$$

The ordering $\leq$ on $\mathfrak{T}$ is defined as the pointwise extension of $\leq_{\mathfrak{M}}$ and similarly the other binary operators on $\mathfrak{M}$ are extended to $\mathfrak{T}$. The least element $\bot$ and the greatest element $\top$ of $\mathbf{Lab} \to \mathfrak{M}$ are defined as expected. As usual we shall write $\sqcup$ and $\sqcap$ for the least upper bound and greatest lower bound operations.

Let us now focus on the actions generated when a *single* action is executed just once. First consider the process $\alpha_1^{\ell_1}.P_1 \mid \alpha_2^{\ell_2}.P_2$. When ℓ_i is selected for execution *all* the actions exposed in P_i will be possible candidates for the next interaction; that is, they are generated by ℓ_i. The situation is slightly more complicated in the case where ℓ_1 and ℓ_2 happen to be equal and we shall then take the least upper bound (or maximum) of the two generated multisets — thereby making clear that we construct an over-approximation to the actual set of generated labels. The situation is similar if the two processes are combined using the sum construct rather than the parallel construct.

To formalise this, we shall define the function

$$\mathcal{G}_\star : \mathbf{Proc} \to \mathfrak{T}$$

that, in the case of non-recursive processes, is defined by:

$$\begin{aligned}
\mathcal{G}_\star[\![(\nu\, x)\, P]\!] &= \mathcal{G}_\star[\![P]\!] \\
\mathcal{G}_\star[\![P \mid P']\!] &= \mathcal{G}_\star[\![P]\!] \sqcup \mathcal{G}_\star[\![P']\!] \\
\mathcal{G}_\star[\![\Sigma_{i\in I}\alpha_i^{\ell_i}.P_i]\!] &= \sqcup_{i\in I}(\bot[\ell_i \mapsto \mathcal{E}_\star[\![P_i]\!]] \sqcup \mathcal{G}_\star[\![P_i]\!])
\end{aligned}$$

We are now collecting information for the complete process, i.e. for *all* actions, so in the case of sums we will have contributions from the actions $\alpha_i^{\ell_i}$ as well as the continuations P_i. The definition is generalised to recursive processes using a fixed point construction.

In the definition of $\mathcal{G}$ we make use of the function $\mathcal{E}_\star$ thereby making sure that the exposed actions are computed relative to the complete program of interest. It is worth pointing out that the construction implicitly performs a reachability analysis and only reports on actions that are reachable from the main process.

Example 2 *Returning to Example 1 we get the following generated actions for the* MP *process:*

ℓ	4	5	6	7	8	9
$\mathcal{G}_\star[\![\mathsf{MP}]\!](\ell)$	$\{5^1\}$	G	$\{7^1\}$	G	$\{9^1\}$	G

where $G = \{4^1, 6^1, 8^1\}$.

3.2. Killed actions

We now turn our attention to the actions that are killed when a single action is executed. We go for an under-approximation as it always will be safe to kill too few actions.

Let us first consider a process of the form $P \mid P'$ and assume that some action labelled ℓ is executed. It may be that both P and P' contain an action labelled ℓ, and since we cannot distinguish between the two, we have to be conservative and only kill the actions that definitely would be killed regardless of whether it was the action from P or the one from P' that was executed. Assuming that K and K' (in $\mathfrak{T}$) are mappings with the relevant information for P and P', respectively, we see that it is only safe to kill the actions of $K(\ell) \sqcap_{\mathfrak{M}} K'(\ell)$. Actually, if ℓ only occurs in one of the two branches then we can still use the $\sqcap_{\mathfrak{M}}$ operation provided that we make sure that the default value for a label *not* occurring in a process is $\top_{\mathfrak{M}}$.

The case of summation $\Sigma_{i\in I}\alpha_i^{\ell_i}.P_i$ is considerably more complex. As a simple example consider $\alpha^\ell.\alpha^\ell.0 + \alpha'^{\ell'}.0$. When the first occurrence of α^ℓ is executed it will kill one occurrence of ℓ (namely itself) and one occurrence of ℓ'. However, when the second occurrence of α^ℓ is executed it will kill only one occurrence of ℓ (namely itself) as the occurrence of ℓ' has already been killed previously. Hence in

order to provide an under-approximation we must limit ourselves to claim that when α^ℓ executes there is one copy of ℓ that is killed (namely itself) and nothing else.

To formalise this, we shall define the function

$$\mathcal{K}_\star : \mathbf{Proc} \to \mathfrak{T}$$

that, for the non-recursive processes, is defined by:

$$\begin{aligned}
\mathcal{K}_\star[\![(\nu\, x)\, P]\!] &= \mathcal{K}_\star[\![P]\!] \\
\mathcal{K}_\star[\![P \mid P']\!] &= \mathcal{K}_\star[\![P]\!] \sqcap \mathcal{K}_\star[\![P']\!] \\
\mathcal{K}_\star[\![\Sigma_{i\in I}\alpha_i^{\ell_i}.P_i]\!] &= \sqcap_{i\in I}(\top[\ell_i \mapsto M] \sqcap \mathcal{K}_\star[\![P_i]\!]) \\
&\quad \text{where } M = +_{\mathfrak{M} j\in I} \bot_{\mathfrak{M}}[\ell_j \mapsto 1]
\end{aligned}$$

Since we are constructing an under-approximation there is a number of differences compared to the previous tables. As discussed above we use the greatest lowest bound operation $\sqcap$ in the case of parallelism and sums; note that the clause for sum specialises to $\top$ in case the index set is empty. The definition is generalised to recursive processes using a fixed point construction.

Example 3 *Returning to Example 1 we get the following killed actions for the* MP *process:*

ℓ	4	5	6	7	8	9
$\mathcal{K}_\star[\![\mathsf{MP}]\!](\ell)$	K	$\{5^1\}$	K	$\{7^1\}$	K	$\{9^1\}$

where $K = \{4^1, 6^1, 8^1\}$.

3.3. Semantic properties

The functions $\mathcal{E}_\star$, $\mathcal{G}_\star$ and $\mathcal{K}_\star$ enjoy the following semantic properties:

Lemma 1 *If* $P \equiv Q$ *then* $\mathcal{E}_\star[\![P]\!] = \mathcal{E}_\star[\![Q]\!]$, $\mathcal{G}_\star[\![P]\!] = \mathcal{G}_\star[\![Q]\!]$ *and* $\mathcal{K}_\star[\![P]\!] = \mathcal{K}_\star[\![Q]\!]$.

Lemma 2 *If* $P \rightarrow^\alpha_{\tilde\ell} Q$ *then* $\tilde\ell \in \mathsf{dom}(\mathcal{E}_\star[\![P]\!])$, $\mathcal{G}_\star[\![P]\!] \geq \mathcal{G}_\star[\![Q]\!]$ *and* $\mathcal{K}_\star[\![P]\!] \leq \mathcal{K}_\star[\![Q]\!]$.

Writing $T(\tilde\ell)$ as a shorthand for $T(\ell_1) +_{\mathfrak{M}} T(\ell_2)$ when $\tilde\ell = \ell_1\ell_2$ we have:

Proposition 3 *If* $P \rightarrow^\alpha_{\tilde\ell} Q$ *then*

$$\mathcal{E}_\star[\![Q]\!] \leq_{\mathfrak{M}} (\mathcal{E}_\star[\![P]\!] -_{\mathfrak{M}} \mathcal{K}_\star[\![P]\!](\tilde\ell)) +_{\mathfrak{M}} \mathcal{G}_\star[\![P]\!](\tilde\ell)$$

This result provides the key insight for constructing the automaton of interest: Whenever we have a state q_s in the automaton it will be described by some extended multiset E_s of exposed actions. Assume now that actions $\tilde\ell$ of E_s may interact. Then this should be reflected in the automaton by a transition (i.e. an edge) from q_s to some state q_t labelled $\tilde\ell$ and the extended multiset E_t of exposed actions associated with q_t should satisfy:

$$E_t \leq_{\mathfrak{M}} (E_s -_{\mathfrak{M}} \mathcal{K}_\star[\![P_\star]\!](\tilde\ell)) +_{\mathfrak{M}} \mathcal{G}_\star[\![P_\star]\!](\tilde\ell) \qquad (1)$$

where $P_\star$ is the overall process of interest.

The algorithm to be constructed in Section 5 will use a data structure E to record the association of exposed multisets to states and it is listed in Table 3 for the automaton of Figure 2. It is straightforward to verify that (1) holds for each of the transitions of the automaton: for the transition from q_0 to q_1 we e.g. have:

$$\mathsf{E}[q_1] = (\mathsf{E}[q_0] -_{\mathfrak{M}} \mathcal{K}_\star[\![\cdots]\!](1,6)) +_{\mathfrak{M}} \mathcal{G}_\star[\![\cdots]\!](1,6)$$

because $\mathcal{K}_\star[\![\ldots]\!](1,6) = \{1^1, 4^1, 6^1, 8^1\}$ (see Example 3) and $\mathcal{G}_\star[\![\cdots]\!](1,6) = \{2^1, 7^1\}$ (see Examples 2).

q	$\mathsf{E}[q]$
q_0	$\{1^1, 3^1, 4^1, 6^1, 8^1, 10^1, 12^1, 14^1\}$
q_1	$\{2^1, 3^1, 7^1, 10^1, 12^1, 14^1\}$
q_2	$\{1^1, 3^1, 5^1, 10^1, 12^1, 14^1\}$
q_3	$\{2^1, 3^1, 4^1, 6^1, 8^1, 11^1, 12^1, 14^1\}$
q_4	$\{1^1, 3^1, 4^1, 6^1, 8^1, 11^1, 12^1, 14^1\}$
q_5	$\{2^1, 3^1, 5^1, 11^1, 12^1, 14^1\}$
q_6	$\{2^1, 3^1, 4^1, 6^1, 8^1, 10^1, 13^1, 14^1\}$
q_7	$\{2^1, 3^1, 5^1, 10^1, 13^1, 14^1\}$
q_8	$\{2^1, 3^1, 9^1, 10^1, 12^1, 14^1\}$
q_9	$\{2^1, 3^1, 4^1, 6^1, 8^1, 11^1, 13^1, 14^1\}$
q_{10}	$\{2^1, 3^1, 4^1, 6^1, 8^1, 11^1, 12^1, 14^1\}$
q_{11}	$\{2^1, 3^1, 5^1, 11^1, 13^1, 14^1\}$
q_{12}	$\{2^1, 3^1, 9^1, 11^1, 12^1, 14^1\}$
q_{13}	$\{2^1, 3^1, 7^1, 11^1, 12^1, 14^1\}$
q_{14}	$\{1^1, 3^1, 5^1, 11^1, 12^1, 14^1\}$
q_{15}	$\{2^1, 3^1, 5^1, 11^1, 12^1, 14^1\}$

Table 3. The exposed actions of the states of the automaton on Figure 2.

Validating security property P1. With this understanding of the automaton of Figure 2 we can now have a closer look at property **P1** from the Introduction: Here we want to guarantee that the car never receives unsolicited messages from the news agent. We can rephrase this as: is it possible that the action $\mathsf{info?x}^2$ of the message board MB could be performed before the action $\mathsf{login!pwd}^1$. Surely, the answer is obvious from the specification of the process MB but to illustrate our approach let us also validate it using the automaton: does there exist a path in the graph where an edge labelled 2 precedes one labelled 1? It is easy to see that this is not the case and hence the property will be satisfied.

In Section 6 we shall return to this property for a scenario where the answer is less obvious.

4. Step 2: Bindings of names

We shall now turn our attention to the bindings. Recall that each (defining as well as applied) occurrence of a name has an index that identifies its defining occurrences; the indices are merely pointers into the program and thus are neither affected by alpha-renaming nor communication of names.

We shall record the binding using mappings R of the domain:

$$\mathfrak{R} = \mathbf{Index} \to \mathcal{P}(\mathbf{Index})$$

The idea is that $R(\imath)$ approximates which names may occur at the *applied* occurrences of names annotated with $\imath$; these names will be represented by the indices of their *defining* occurrence. This level of indirection is introduced because the structural congruence allows alpha-renaming of bound names and hence the names themselves cannot be used to carry analysis information; on the other hand, the indices record the positions of the names and they will be stable under alpha-renaming. Actually, the bindings $R \in \mathfrak{R}$ to be constructed in the analysis will all satisfy that $R(\imath) = \{\imath\}$ whenever $\imath$ is an index of a free name or a constant name introduced by the construct $(\nu\, x^\imath)\, P$.

We shall equip $\mathfrak{R}$ with a partial ordering $\leq_{\mathfrak{R}}$ that is the pointwise extension of the subset ordering on $\mathcal{P}(\mathbf{Index})$; this turns it into a complete lattice with least element $\bot$ and least upper bound operation $\sqcup$.

4.1. Exposed bindings

Initially, the bindings of the indices will determined by a function:

$$\mathcal{R}_\star : \mathbf{Proc} \to \mathfrak{R}$$

For non-recursive processes it is defined by the clauses:

$$\begin{array}{rcl} \mathcal{R}_\star[\![(\nu\, x^\imath)\, P]\!] & = & \mathcal{R}_\star[\![P]\!] \\ \mathcal{R}_\star[\![P \mid P']\!] & = & \mathcal{R}_\star[\![P]\!] \sqcup \mathcal{R}[\![P']\!] \\ \mathcal{R}_\star[\![\Sigma_{i\in I}\alpha_i.P_i]\!] & = & \sqcup_{i\in I}([\![\alpha_i]\!] \sqcup \mathcal{R}[\![P_i]\!]) \end{array}$$

Here the auxiliary operation $[\![\alpha]\!]$ extracts the relevant information from the action α and it is defined by:

$$\begin{array}{rcl} [\![\tau]\!] & = & \bot \\ [\![[x^\imath = y^\jmath]]\!] & = & \bot[\imath \mapsto \{\imath\}] \sqcup \bot[\jmath \mapsto \{\jmath\}] \\ [\![x^\imath?\vec{y}^{\vec{\jmath}}]\!] & = & \bot[\imath \mapsto \{\imath\}] \\ [\![x^\imath!\vec{y}^{\vec{\jmath}}]\!] & = & \sqcup\{\bot[\jmath \mapsto \{\jmath\}] \mid \jmath \in \{\vec{\jmath}\}\} \sqcup \bot[\imath \mapsto \{\imath\}] \end{array}$$

Note that we only have contributions from applied occurrences of names. The defition of $\mathcal{R}_\star$ is extended to recursive processes using a fixed point definition.

Example 4 *For the running example we shall now add indices as follows (and omit the labels for the sake of readability):*

$$\begin{array}{rcl} \mathsf{MB} & \triangleq & \mathsf{login}^1!(\mathsf{pwd}^2).\ \mathsf{info}^3?(\mathsf{x}^9).\ \mathsf{MB} \\ \mathsf{GPS} & \triangleq & (\nu\, \mathsf{pos}^{10})\mathsf{gps}^4!(\mathsf{pos}^{10}).\ \mathsf{GPS} \\ \mathsf{MP} & \triangleq & \mathsf{gps}^4?(\mathsf{u}^{11}).\ \mathsf{wifi}^5!(\mathsf{log}^6, \mathsf{u}^{11}).\ \mathsf{MP} \\ & + & \mathsf{login}^1?(\mathsf{u}^{12}).\ \mathsf{wifi}^5!(\mathsf{pop}^7, \mathsf{u}^{12}).\ \mathsf{MP} \\ & + & \mathsf{msg}^8?(\mathsf{u}^{13}, \mathsf{v}^{14}).\ \mathsf{wifi}^5!(\mathsf{info}^3, \mathsf{u}^{13}).\ \mathsf{MP} \\ \mathsf{MP}' & \triangleq & \mathsf{wifi}^5?(\mathsf{u}^{15}, \mathsf{v}^{16}).\ \mathsf{u}^{15}!(\mathsf{v}^{16}).\ \mathsf{MP}' \\ \mathsf{NA} & \triangleq & \mathsf{pop}^7?(\mathsf{z}^{17}).\ (\nu\, \mathsf{news}^{18})\mathsf{msg}^6!(\mathsf{news}^{18}, \mathsf{z}^{17}).\ \mathsf{NA} \\ \mathsf{LOG} & \triangleq & \mathsf{log}^6?(\mathsf{y}^{19}).\ \mathsf{LOG} \end{array}$$

Since all the defining occurrences of names have unique indices the result produced by $\mathcal{R}_\star$ will simply map each index occurring in an applied position to the singleton set containing the index itself.

4.2. The transfer function

We shall now define functions

$$\mathsf{transfer}_{\tilde{\ell}} : \mathfrak{M} \times \mathfrak{R} \hookrightarrow \mathfrak{M} \times \mathfrak{R}$$

for each potential interaction $\tilde{\ell}$; recall that $\tilde{\ell}$ will be a single label in the case of a τ-action or a match action and it will be a pair (ℓ_1, ℓ_2) of labels in the case of a communication. The functions will be partial functions since the interaction might not be possible. However, if it could occur, then the result of the function will describe how the multiset of exposed actions and the bindings will be updated as a result of the interaction.

In the following, we shall give the definition of $\mathsf{transfer}_{\tilde{\ell}}(E, R)$; there are three cases:

- First assume that ℓ is the label of a τ-action. The function will only succeed if $E(\ell) > 0$ and it will then return a pair (E', R') recording that the action took place:

$$\begin{array}{rcl} E' & = & E -_{\mathfrak{M}} \mathcal{K}[\![P_\star]\!](\ell) +_{\mathfrak{M}} \mathcal{G}[\![P_\star]\!](\ell) \\ R' & = & R \end{array}$$

- Next assume that ℓ is the label of a match $[x^\imath = y^\jmath]$. This action is only possible if $E(\ell) > 0$ and furthermore it must be possible for x and y to denote the same

name, that is, it must be the case that $R(\imath) \cap R(\jmath) \neq \emptyset$. In this case the function will return the pair (E', R') where:

$$E' = E -_{\mathfrak{M}} \mathcal{K}[\![P_\star]\!](\ell) +_{\mathfrak{M}} \mathcal{G}[\![P_\star]\!](\ell)$$

We can simply take $R' = R$ but since we know that the match was successful we can refine the bindings of $\imath$ and $\jmath$ to be:

$$R' = R[\imath \mapsto I][\jmath \mapsto I] \qquad \text{where } I = R(\imath) \cap R(\jmath)$$

- Finally, assume that ℓ is the label of an output action $x^{\imath}!\vec{y}^{\vec{\jmath}}$ and ℓ' is the label of an input action $x^{\prime\imath'}?\vec{y}'^{\vec{\jmath}'}$. The action is only possible if $E(\ell) > 0$ and $E(\ell') > 0$ and furthermore it must be possible for x and x' to denote the same name; that is, it must be the case that $R(\imath) \cap R(\imath') \neq \emptyset$. If these conditions are fulfilled the function will return the pair (E', R') where:

$$E' = E -_{\mathfrak{M}} \mathcal{K}[\![P_\star]\!](\ell\ell') +_{\mathfrak{M}} \mathcal{G}[\![P_\star]\!](\ell\ell')$$

Since we know that the communication was successful, we can refine the bindings of $\imath$ and $\imath'$ in R' as we did in the case of matching, and additionally we have to record that the new potential bindings of names in the input positions $\vec{\jmath}' = \jmath'_1 \cdots \jmath'_n$ are obtained from those of the output positions $\vec{\jmath} = \jmath_1 \cdots \jmath_n$:

$$\begin{array}{l} R' = R[\imath \mapsto I][\imath' \mapsto I][\jmath'_1 \mapsto J_1] \cdots [\jmath'_n \mapsto J_n] \\ \quad \text{where} \quad I = R(\imath) \cap R(\imath') \\ \quad \text{and} \quad J_i = R(\jmath_i) \qquad \text{for } 1 \leq i \leq n \end{array}$$

The analysis presented above can be made more precise by eliminating bindings that are no longer feasible or that are no longer relevant. A basic observation is that after having executed the prefix α of $\alpha.P$, then only the values bound to the free names of P will be interesting – all other names will be *dead* in the terminology of data flow analysis – and hence they can be removed. Due to lack of space we shall refrain from exploring these possibilities in this paper.

4.3. Semantic properties

To conclude, we shall list the correctness property for the transfer function:

Lemma 4 *If $P \equiv Q$ then $\mathcal{R}_\star[\![P]\!] = \mathcal{R}_\star[\![Q]\!]$. If $P \rightarrow^{\alpha}_{\tilde{\ell}} Q$ and $x^{\imath} \in \mathsf{fn}(\alpha)$ then $\imath \in \mathcal{R}_\star[\![P]\!](\imath)$.*

Let us define the ordering $\sqsubseteq$ on $\mathfrak{M} \times \mathfrak{R}$ in a pointwise manner: $(E, R) \sqsubseteq (E', R')$ if and only if $E \leq_{\mathfrak{M}} E'$ and $R \leq_{\mathfrak{R}} R'$. Then:

q	$\mathsf{R}[q](\imath) \setminus \{\imath\}$
q_0	$[\,]$
q_1	$[\mathsf{u}^{12} \mapsto \{\mathsf{pwd}^2\}]$
q_2	$[\mathsf{u}^{11} \mapsto \{\mathsf{pos}^{10}\}]$
q_3	$[\mathsf{u}^{15} \mapsto \{\mathsf{pop}^7\},\ \mathsf{v}^{16} \mapsto \{\mathsf{pwd}^2, \mathsf{u}^{12}\}]$
q_4	$[\mathsf{u}^{15} \mapsto \{\mathsf{log}^6\},\ \mathsf{v}^{16} \mapsto \{\mathsf{pos}^{10}, \mathsf{u}^{11}\}]$
q_5	$[\mathsf{u}^{11} \mapsto \{\mathsf{pos}^{10}\},\ \mathsf{u}^{15} \mapsto \{\mathsf{pop}^7\}, \mathsf{v}^{16} \mapsto \{\mathsf{pwd}^2, \mathsf{u}^{12}\}]$
q_6	$[\mathsf{z}^{17} \mapsto \{\mathsf{pwd}^2, \mathsf{u}^{12}, \mathsf{v}^{16}\}]$
q_7	$[\mathsf{u}^{11} \mapsto \{\mathsf{pos}^{10}\},\ \mathsf{z}^{17} \mapsto \{\mathsf{pwd}^2, \mathsf{u}^{12}, \mathsf{v}^{16}\}]$
q_8	$[\mathsf{u}^{13} \mapsto \{\mathsf{news}^{18}\}]$
q_9	$[\mathsf{u}^{15} \mapsto \{\mathsf{log}^6\},\ \mathsf{v}^{16} \mapsto \{\mathsf{pos}^{10}, \mathsf{u}^{11}\},$ $\mathsf{z}^{17} \mapsto \{\mathsf{pwd}^2, \mathsf{u}^{12}, \mathsf{v}^{16}\}]$
q_{10}	$[\mathsf{u}^{15} \mapsto \{\mathsf{info}^3\},\ \mathsf{v}^{16} \mapsto \{\mathsf{news}^{18}, \mathsf{u}^{13}\}]$
q_{11}	$[\mathsf{u}^{11} \mapsto \{\mathsf{pos}^{10}\},\ \mathsf{u}^{15} \mapsto \{\mathsf{log}^6\},\ \mathsf{v}^{16} \mapsto \{\mathsf{pos}^{10}, \mathsf{u}^{11}\},$ $\mathsf{z}^{17} \mapsto \{\mathsf{pwd}^2, \mathsf{u}^{12}, \mathsf{v}^{16}\}]$
q_{12}	$[\mathsf{u}^{13} \mapsto \{\mathsf{news}^{18}\}, \mathsf{u}^{15} \mapsto \{\mathsf{log}^6\},\ \mathsf{v}^{16} \mapsto \{\mathsf{pos}^{10}, \mathsf{u}^{11}\}]$
q_{13}	$[\mathsf{u}^{12} \mapsto \{\mathsf{pwd}^2\},\ \mathsf{u}^{15} \mapsto \{\mathsf{log}^6\},\ \mathsf{v}^{16} \mapsto \{\mathsf{pos}^{10}, \mathsf{u}^{11}\}]$
q_{14}	$[\mathsf{u}^{11} \mapsto \{\mathsf{pos}^{10}\},\ \mathsf{u}^{15} \mapsto \{\mathsf{log}^6\},\ \mathsf{v}^{16} \mapsto \{\mathsf{pos}^{10}, \mathsf{u}^{11}\}]$
q_{15}	$[\mathsf{u}^{11} \mapsto \{\mathsf{pos}^{10}\},\ \mathsf{u}^{15} \mapsto \{\mathsf{info}^3\}, \mathsf{v}^{16} \mapsto \{\mathsf{news}^{18}, \mathsf{u}^{13}\}]$

Table 4. The bindings of the states of the automaton on Figure 2.

Proposition 5 *If $P \rightarrow^{\alpha}_{\tilde{\ell}} Q$ then*

$$(\mathcal{E}_\star[\![Q]\!], \mathcal{R}_\star[\![Q]\!]) \sqsubseteq \mathsf{transfer}_{\tilde{\ell}}(\mathcal{E}_\star[\![P]\!], \mathcal{R}_\star[\![P]\!])$$

This result is exploited in the construction of the automaton in the next section. For each state q_s we will have information about not only the exposed actions E_s but also the exposed bindings R_s. When the actions $\tilde{\ell}$ of E_s interact we will, as mentioned earlier obtain a transition to some state q_t; it will have exposed actions E_t and bindings R_t satisfying:

$$(E_t, R_t) \sqsubseteq \mathsf{transfer}_{\tilde{\ell}}(E_s, R_s) \tag{2}$$

Note that this does not contradict equation (1).

It shall use a data structure R to associate bindings with the individual states; Table 4 lists the bindings associated with the states of the automaton of Figure 2. We have only listed the interesting part of the binding information, that is, entries where $\mathsf{R}[q](\imath) \neq \{\imath\}$; to ease the readability we have also included the names corresponding to the indices in the initial program. It is straightforward to verify that the bindings are correctly captured in the states: as an example the pair (1,6) of labels records that $\mathsf{login}!(\mathsf{pwd}^2)^1$ and $\mathsf{login}?(\mathsf{u}^{12})^6$ communicate with one another and as a result pwd^2 should be bound to u^{12} exactly as is recorded in state q_1 (and q_{13}).

Validating security property P2. Let us now have a closer look at property **P2** from the Introduction: We are

concerned with the flow of information and want to ensure that the gps position of the driver never is revealed to the news agent. The knowledge of the news agent is represented by the name z^{17} so we may ask whether there exists a state in the automaton where the position pos^{10} of the car is bound to z^{17}. A simple inspection of Table 4 shows that this is not the case.

Actually, it is *only* possible to validate this property because the analysis is able to analyse the use of the multiplexed channel wifi fairly precisely. The channel is used for communication the gps position in the states q_4, q_9, q_{11}, q_{12}, q_{13} and q_{14} (boxes in Figure 2) and from Table 4 we can se that u^{15} and v^{16} will be bound to log and pos, respectively. The wifi channel is used in the login procedure in the states q_3 and q_5 (triangles in Figure 2) and here u^{15} and v^{16} will be bound to pop and pwd, respectively. Finally, it is used for the news in states q_{10} and q_{15} (polygons in Figure 2) and here u^{15} and v^{16} are bound to info and news, respectively. An analysis that is not able to distinguish between these three situations will not be able to validate security property **P2**.

5. Step 3: The finite automaton

For a process $P_\star$, the extended multiset $\mathcal{E}_\star[\![P_\star]\!]$ identifies the exposed actions and the mapping $\mathcal{R}_\star[\![P_\star]\!]$ identifies the exposed bindings. The functions $\mathsf{transfer}_{\tilde{\ell}}$ specifies how this information is modified when actions are executed. We shall now use this to construct a finite automaton $(\mathbf{Q}_\star, \delta_\star, q_\star)$ abstracting the overall behaviour of $P_\star$.

Each *state* $q \in \mathbf{Q}_\star$ will describe a potential configuration of $P_\star$ by an extended multisets of exposed actions, $\mathsf{E}[q] \in \mathfrak{M}$, and an approximation of the relevant name bindings $\mathsf{R}[q] \in \mathfrak{R}$.

A *transition* $(q_s, \tilde{\ell}, q_t) \in \delta_\star$ then reflects a potential transition from q_s to q_t by the action(s) labelled $\tilde{\ell}$; here $\tilde{\ell}$ may be a single label (corresponding to a silent action or a match) or it may be a pair of labels corresponding to a communication.

The automaton will be constructed using the worklist algorithm shown in Table 5. In line (1) it records that the initial state $q_\star$ has $\mathsf{E}[q_\star] = \mathcal{E}_\star[\![P_\star]\!]$ and $\mathsf{R}[q_\star] = \mathcal{R}_\star[\![P_\star]\!]$. This information may be updated as more states and transitions are added by the algorithm. The algorithm uses the data structures Q and Δ to hold the accumulated version of the automaton (the set of states and the transition relation, respectively) while the worklist W keeps track of the states that still have to be processed; Q, Δ and W are initialised in line (2).

Line (3) contains the classical loop inspecting the contents

$\mathsf{E}[q_\star] := \mathcal{E}_\star[\![P_\star]\!]; \mathsf{R}[q_\star] := \mathcal{R}_\star[\![P_\star]\!];$	(1)
$\mathsf{W} := \{q_\star\}; \mathsf{Q} := \{q_\star\}; \Delta := \emptyset;$	(2)
$\mathsf{while}\ \mathsf{W} \neq \emptyset\ \mathsf{do}$	(3)
$\quad \mathsf{select}\ q\ \mathsf{from}\ \mathsf{W}; \mathsf{W} := \mathsf{W} \setminus \{q\};$	(4)
$\quad \mathsf{for\ each}\ \tilde{\ell} \in \mathsf{enabled}(\mathsf{E}[q])\ \mathsf{do}$	(5)
$\quad\quad \mathsf{case}\ \mathsf{transfer}_{\tilde{\ell}}(\mathsf{E}[q], \mathsf{R}[q])$	(6)
$\quad\quad \mathsf{of}\ (E', R') : \mathsf{update}(q_s, \tilde{\ell}, (E', R'))$	(7)

Table 5. Worklist algorithm.

of the worklist. A state q is selected and removed from the worklist in line (4) and the set of enabled transitions are constructed using the function $\mathsf{enabled}(\mathsf{E}[q])$ in line (5). For each $\tilde{\ell}$ in this set, the function $\mathsf{transfer}_{\tilde{\ell}}(\mathsf{E}[q], \mathsf{R}[q])$ is called in line (6). If it fails then no further action is taken as this means that the transition was not possible after all. If the function succeeds then it determines a possible new state by returning a pair (E', R'), and the function $\mathsf{update}(q_s, \tilde{\ell}, (E', R'))$ will be called in line (7) to update the automaton to reflect this.

The main challenge in the construction of the worklist algorithm is to ensure that it terminates. As we shall see shortly, judicious choices in the definition of the update function will take care of this; however let us first have a closer look at the details of the function enabled.

5.1. Enabled actions

Based on the exposed actions the operation $\mathsf{enabled}(E)$ will return a set of *potential* interactions. Motivated by the definition of $\mathsf{transfer}_{\tilde{\ell}}$ in Section 4 we may simply take:

- if $E(\ell) \geq 1$ and ℓ is the label of a τ action or a match action then $\ell \in \mathsf{enabled}(E)$, and
- if $E(\ell) \geq 1$ and $E(\ell') \geq 1$ and ℓ and ℓ' are labels of an output and an input action, resp., then $(\ell, \ell') \in \mathsf{enabled}(E)$.

The assumption that the initial program is uniquely labelled ensures that each label is associated with exactly one kind of action (and this property is preserved by the semantics).

However, we may do slightly better by observing that that parallelism and choice behave differently: for $P \mid P'$ we have the possibility of an action from P interacting with one from P' whereas this situation never will occur for $P + P'$. We omit the details.

if some $q'' \in \mathsf{Q}$ with $H(\mathsf{E}[q''], \mathsf{R}[q'']) = H(E, R)$	(1)
then $q' := q''$	(2)
else select q' from outside Q;	(3)
$\mathsf{Q} := \mathsf{Q} \cup \{q'\}; \mathsf{E}[q'] := \bot_{\mathfrak{M}}; \mathsf{R}[q'] := \bot;$	(4)
if $\neg(E \leq_{\mathfrak{M}} \mathsf{E}[q'] \wedge R \leq_{\mathfrak{R}} \mathsf{R}[q'])$	(5)
then $\mathsf{E}[q'] := \mathsf{E}[q'] \nabla_{\mathfrak{M}} E;\ \ \mathsf{R}[q'] := \mathsf{R}[q'] \sqcup R;$	(6)
$\mathsf{W} := \mathsf{W} \cup \{q'\};$	(7)
$\Delta := \Delta \backslash \{(q, \tilde{\ell}, q'') \mid q'' \in \mathsf{Q}\} \cup \{(q, \tilde{\ell}, q')\};$	(8)
clean-up$(\mathsf{Q}, \mathsf{W}, \Delta)$	(9)

Table 6. Processing enabled actions: update$(q, \tilde{\ell}, (E, R))$.

5.2. Updating the automaton

The function transfer$_{\tilde{\ell}}$ will inspect all the candidate interactions discovered by enabled and, if it succeeds, the function update will update the automaton with the new transition. This function will decide whether it is possible to reuse an existing state. If it is, then its description should be updated and otherwise a new state will have to be created. In both cases, the data structures must be updated to reflect the extension of the automaton.

The procedure update$(q, \tilde{\ell}, (E, R))$ is specified in Table 6. Recall that (E, R) describes the new state to which there should be a transition labelled $\tilde{\ell}$ that emerges from q. In line (1) it is checked whether or not there already exists a state q'' in Q that should be used for describing (E, R), in which case we shall reuse it as indicated in line (2); otherwise, in lines (3-4) we introduce a new state and initialise the corresponding entries in E and R to be the least elements of $\mathfrak{M}$ and $\mathfrak{R}$, respectively.

Line (1) makes use of a *granularity function*:

$$H : \mathfrak{M} \times \mathfrak{R} \to \mathfrak{H}$$

The most obvious choice might be the identity function, i.e. $H(E, R) = (E, R)$, but it turns out that this choice may lead to non-termination of the worklist algorithm. Some more interesting choices that all will lead to termination are:

$$\begin{aligned} H^{\mathsf{dom}}(E, R) &= \mathsf{dom}(E) \\ H^{\mathsf{dom} \cap L}(E, R) &= \mathsf{dom}(E) \cap L \text{ for } L \subseteq_{\mathsf{fin}} \mathbf{Lab} \\ H^{\mathsf{dom}}_{\mathsf{id}}(E, R) &= (\mathsf{dom}(E), R) \end{aligned}$$

As an example, H^{dom} expresses that only the domain of the extended multiset is of interest – and in this case it is ensured that all the states will have distinct domains for their extended multisets and hence that there is at most one candidate for q' in line (2).

In line (5) it is then checked whether q' already captures the required information given by the parameters E and R. If not, the entries in the tables E and R must be updated as shown in line (6) and furthermore q' will have to be put on the worklist (line (7)) so that it can be processed again. If q' is a new state then the test of line (5) is likely to succeed and lines (6-7) will be executed. However, it may be that the test of line (1) succeeded in finding a suitable old state q' except that it does not capture the information required and then lines (6-7) will also have to be executed — this is for example likely to occur frequently when $H(E, R) = \mathsf{dom}(E)$.

The widening operator $\nabla_{\mathfrak{M}}$ of line (6) makes sure to combine the old and the new extended multiset in such a way that termination of the overall algorithm is ensured; it is defined by

$$(M_1 \nabla_{\mathfrak{M}} M_2)(\ell) = \begin{cases} M_1(\ell) & \text{if } M_2(\ell) \leq M_1(\ell) \\ M_2(\ell) & \text{if } M_1(\ell) = 0 \wedge M_2(\ell) > 0 \\ \infty & \text{otherwise} \end{cases}$$

In a similar way the least upper bound operation $\sqcup$ combines the old and the new binding information in a pointwise manner.

Finally, line (8) expresses that the transition relation is updated and that all previous $\tilde{\ell}$ transitions from state q are removed; the reason is that their destination may no longer be correct. As a consequence the automaton may contain unreachable parts and the procedure clean-up$(\mathsf{Q}, \mathsf{W}, \Delta)$ will remove those parts of Q, W and Δ that cannot be reached from the initial state $q_\star$.

5.3. Semantic properties

We now develop the simulation result showing the correctness of the automaton constructed by Table 5. A state denoting the pair (E, R) is said to represent the process P, written $P \rhd (E, R)$, which is defined by

$$P \rhd (E, R) \qquad \text{iff} \qquad \mathcal{E}_\star[\![P]\!] \leq_{\mathfrak{M}} E \wedge \mathcal{R}_\star[\![P]\!] \leq_{\mathfrak{R}} R$$

Lemma 6 *If $P \equiv Q$ then $P \rhd (E, R)$ if and only if $Q \rhd (E, R)$.*

We can now establish the main result which is independent of the choice of the granularity function $H : \mathfrak{M} \times \mathfrak{R} \to \mathfrak{H}$. It only requires that it is *finitary*, meaning that for all

choices of finite sets $\mathbf{Lab}_\mathsf{f} \subseteq \mathbf{Lab}$ and $\mathbf{Index}_\mathsf{f} \subseteq \mathbf{Index}$, H specialises to

$$H : (\mathbf{Lab}_\mathsf{f} \to \mathbb{N} \cup \{\infty\}) \times (\mathbf{Index}_\mathsf{f} \to \mathcal{P}(\mathbf{Index}_\mathsf{f})) \to \mathfrak{H}_\mathsf{f}$$

for some finite subset $\mathfrak{H}_\mathsf{f} \subseteq \mathfrak{H}$.

Theorem 7 *In the granularity function is finitary, then the algorithm of Table 5 terminates and produces a finite automaton* $(\mathsf{Q}, q_\star, \Delta)$ *together with the tables* E *and* R*. If*

$$P \rhd (\mathsf{E}[q], \mathsf{R}[q]) \text{ and } P \to^\tau_{\tilde{\ell}} Q$$

then there exists a unique $q' \in \mathsf{Q}$ *such that*

$$Q \rhd (\mathsf{E}[q'], \mathsf{R}[q']) \text{ and } (q, \tilde{\ell}, q') \in \Delta$$

The choice of granularity function is the main parameter controlling the complexity of the algorithm and thereby the size of the resulting automaton. Indeed, the number of states of the automaton will be dominated by the cardinality of the range of the granularity function, that is, the size of $\mathfrak{H}_\mathsf{f}$.

6. Worked examples

The analysis has been implemented and used to analyse a number of variants of the scenario presented in the Introduction. We have already discussed the properties **P1** and **P2** for the scenario:

$$\mathsf{MB} \mid \mathsf{GPS} \mid \mathsf{MP} \mid \mathsf{MP}' \mid \mathsf{LOG} \mid \mathsf{NA}$$

We shall now consider variants of the scenario, in particular we shall focus on the formulation of the news agent.

6.1. An intrusive news agent

In the scenario considered so far the message board of the car as well as the news agent insist on the login procedure to be completed before the news can be exchanged. We shall now study more relaxed versions of these processes where neither the message board nor the news agent insist on the sequentialisation:

$$\begin{array}{rcl} \mathsf{MB}_1 & \triangleq & \mathsf{login!pwd}^1.\ \mathsf{MB}_1 + \mathsf{info?x}^2.\ \mathsf{MB}_1 \\ \mathsf{NA}_1 & \triangleq & (\nu\,\mathsf{news})(\mathsf{pop?z}^{12}.\ \mathsf{NA}_1 \\ & & \qquad + \mathsf{msg!}(\mathsf{news}, \cdot)^{13}.\ \mathsf{NA}_1) \end{array}$$

The main program is $\mathsf{MB}_1 \mid \mathsf{GPS} \mid \mathsf{MP} \mid \mathsf{MP}' \mid \mathsf{LOG} \mid \mathsf{NA}_1$ where GPS, MP, MP′ and LOG are exactly as before.

The resulting automaton is shown in Figure 3. The path from q_0 to q_3, q_5 and then q_0 clearly shows that the message board may receive news from the news agent before the driver has logged in: the news agent succeeds in sending the message to the multiplexer which then forwards it to the car. Thus property **P1** is *not* fulfilled.

Turning our attention to property **P2** it turns out that it *is* fulfilled: the news agent will never get hold of the position of the car. As in Section 4 we inspect the bindings of the states in order to validate this property. Again the analysis is able to distinguish between the different uses of the multiplexer channel; the shape of the states of the automaton in Figure 3 provides the interesting information: in the states q_4, q_6, q_7 and q_8 (boxes) the multiplexer channel is used for communication the gps position, in the states q_{12}, q_{13}, q_{14} and q_{15} (triangles) it is used for communicating the password and in the states q_5, q_9, q_{10} and q_{11} (polygons) it is used for forwarding the news.

6.2. A news agent with ads

Let us replace the news agent with an agent that issues unsolicited ads so the overall process becomes $\mathsf{MB} \mid \mathsf{GPS} \mid \mathsf{MP} \mid \mathsf{MP}' \mid \mathsf{LOG} \mid \mathsf{NA}_2$ where

$$\begin{array}{rcl} \mathsf{NA}_2 & \triangleq & \mathsf{pop?z}^{12}.\ (\nu\,\mathsf{news})\mathsf{msg!}(\mathsf{news}, \mathsf{z})^{13}.\ \mathsf{NA}_2 \\ & + & \mathsf{msg!}(\mathsf{ad}, \cdot)^{14}.\mathsf{NA}_2 \end{array}$$

and all the other processes are as before (except that the labelling of LOG is changed to $\mathsf{LOG} \triangleq \mathsf{log?y}^{15}.\ \mathsf{LOG}$).

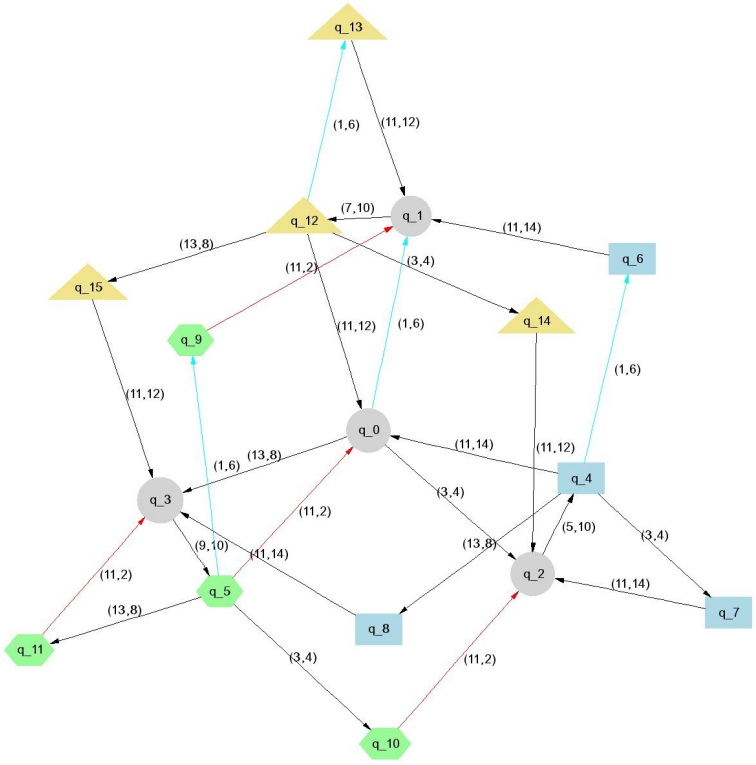
The resulting automaton is displayed in Figure 4. The upper right part of the automaton corresponds to the system displayed in Figure 2. The interactions labelled (14,8) records that the news agent successfully has interacted with the multiplexer channel and after a few more transitions the ads may reach the car.

The automaton suggests that the system may have a number of deadlock states. Since the automaton represents an over approximation of the behaviour of the system we cannot be sure that the system indeed will be able to reach a deadlock but in this particular setting it is possible as evidenced by this derivation sequence:

$$\begin{array}{ll} & \mathsf{MP} \mid \mathsf{MP}' \mid \mathsf{NA}_2 \mid \cdots \\ \to^\tau_{14,8} & \mathsf{wifi!}(\mathsf{info}, \mathsf{ad}).\ \mathsf{MP} \mid \mathsf{MP}' \mid \mathsf{NA}_2 \mid \cdots \\ \to^\tau_{9,10} & \mathsf{MP} \mid \mathsf{info!ad}.\ \mathsf{MP}' \mid \mathsf{NA}_2 \mid \cdots \\ \to^\tau_{14,8} & \mathsf{wifi!}(\mathsf{info}, \mathsf{ad}).\ \mathsf{MP} \mid \mathsf{info!ad}.\ \mathsf{MP}' \mid \mathsf{NA}_2 \mid \cdots \end{array}$$

The process of the last line above have no next step since the message board MB is not ready to receive on the channel info yet. A more sophisticated coding of the message board is needed to overcome this problem.

As before property **P1** is concerned with the *order* in which the actions occur and we have to inspect the paths in the automaton. This shows that the condition is indeed fulfilled so



States using the wifi channel :

boxes :	u ↦ log
	v ↦ pos
triangles :	u ↦ pop
	v ↦ pwd
polygons :	u ↦ info
	v ↦ news

Figure 3. Analysis of the info-system with an intrusive news agent: $\mathsf{MB}_1 \mid \mathsf{GPS} \mid \mathsf{MP} \mid \mathsf{MP}' \mid \mathsf{LOG} \mid \mathsf{NA}_1$.

the analysis guarantees that the car will never get messages from the news agent *before* it has registered. However, the analysis result also shows that *after* having completed the login process the car may receive news as well as ads.

Just as in the previous versions of the system the analysis guarantees that the the location of the car is not revealed to the news agent.

6.3. A news agent with local news

As the final variation of the scenario we shall consider a news agent that provides general news as well as local news; it is specified as follows and further explained below:

$$
\begin{array}{rcl}
\mathsf{MB}_5 & \triangleq & \mathsf{login!pwd}^{1}.\ \mathsf{info?x}^{2}.\ \mathsf{MB}_5 \\
 & + & \mathsf{gps?x}^{3}.\ \mathsf{login!x}^{4}.\ \mathsf{info?x}^{5}.\ \mathsf{MB}_5 \\
\mathsf{GPS}_1 & \triangleq & \mathsf{gps!pos}^{6}.\ \mathsf{GPS}_1 \\
\mathsf{MP} & \triangleq & \mathsf{gps?u}^{7}.\ \mathsf{wifi!}(\mathsf{log},\mathsf{u})^{8}.\ \mathsf{MP} \\
 & + & \mathsf{login?u}^{9}.\ \mathsf{wifi!}(\mathsf{pop},\mathsf{u})^{10}.\ \mathsf{MP} \\
 & + & \mathsf{msg?}(\mathsf{u},\mathsf{v})^{11}.\ \mathsf{wifi!}(\mathsf{info},\mathsf{u})^{12}.\ \mathsf{MP} \\
\mathsf{MP}' & \triangleq & \mathsf{wifi?}(\mathsf{u},\mathsf{v})^{13}.\ \mathsf{u!v}^{14}.\ \mathsf{MP}'
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{NA}_3 & \triangleq & \mathsf{pop?z}^{15}. \\
 & & \quad [\mathsf{z}=\mathsf{pwd}]^{16}(\nu\,\mathsf{news})\,\mathsf{msg!}(\mathsf{news},\mathsf{z})^{17}.\ \mathsf{NA}_3 \\
 & & +[\mathsf{z}=\mathsf{pos}]^{18}(\nu\,\mathsf{local})\,\mathsf{msg!}(\mathsf{local},\mathsf{z})^{19}.\ \mathsf{NA}_3 \\
\mathsf{LOG} & \triangleq & \mathsf{log?y}^{20}.\ \mathsf{LOG}
\end{array}
$$

In order to obtain the local news the car must send its gps position to the news agent; it will obtain that directly from its gps device as shown in the process MB_5. The news agent will, once it receives a password, determine whether or not it contains gps information and then either send general news or local news. In order not to make the example overly complex we shall simply check whether or not the received message z equals pwd or pos. The resulting process is called NA_3 above.

The automaton constructed by the analysis when applied to the system $\mathsf{MB}_5 \mid \mathsf{GPS}_1 \mid \mathsf{MP} \mid \mathsf{MP}' \mid \mathsf{LOG} \mid \mathsf{NA}_2$ is shown in Figure 5. It basically consists of three parts, the leftmost part corresponding to handling the local news, the rightmost part corresponding to handling the general news and an upper part where the news agent is not involved. With this understanding it is not surprising that the left and the right part of the graph are almost symmetric. As in the previous scenarios the shape of the nodes indicate which messages might be communicated over the multiplexer channel in the

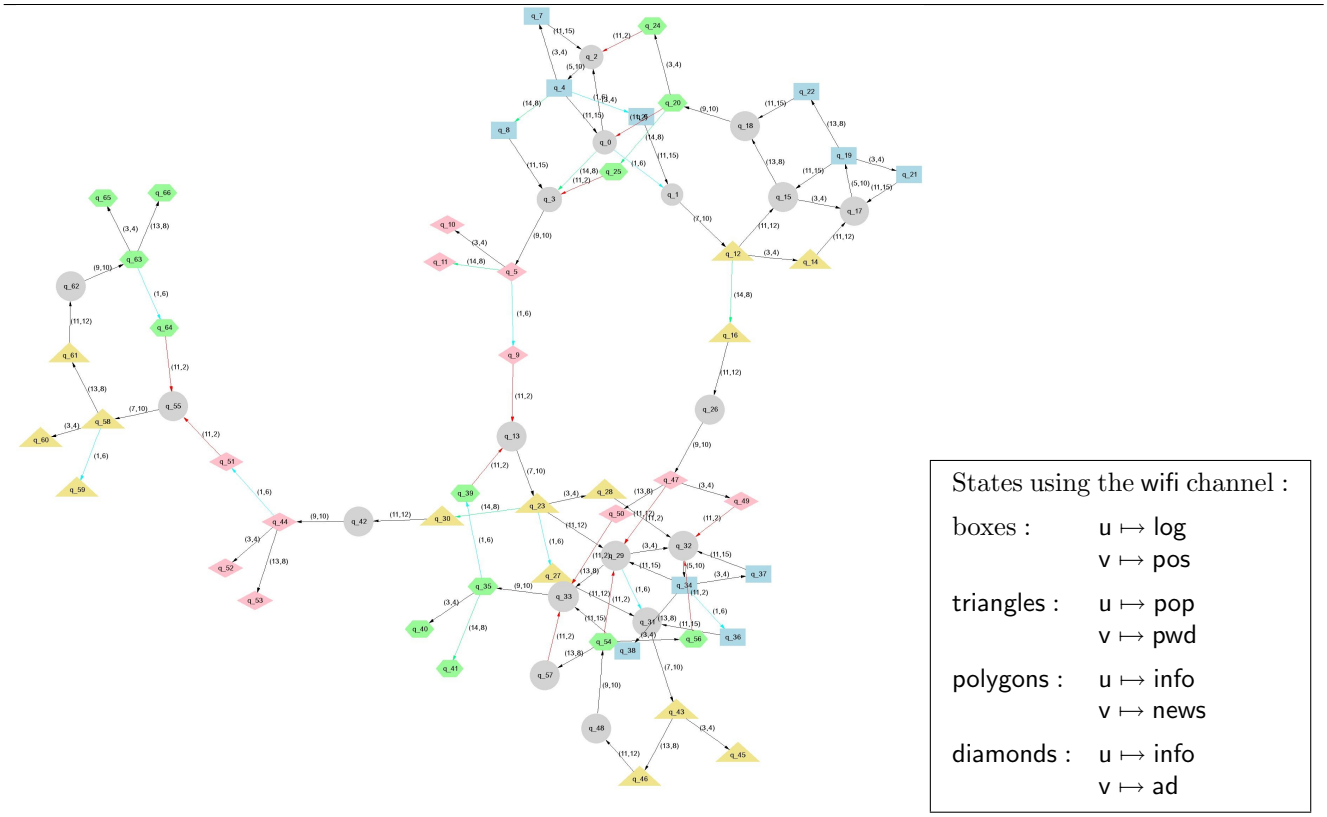


Figure 4. Analysis of the info-system with a news agent with ads: MB | GPS | MP | MP′ | LOG | NA_2.

various states: In the rightmost part of the automaton the states q_{18} and q_{20} (triangles) record that the password pwd is communicated over wifi and the states q_{37} and q_{39} (polygons) that the general news news is communicated over wifi. The states drawn as boxed record that the gps position is being communicated over wifi but we cannot directly see who is going to receive it; in most cases it will be the gps logger but closer inspection of the labels on the edges of the graph shows that in the states q_8 and q_9 it will indeed be received by the news agent. The states q_{36} and q_{38} (diamonds) then records that local news are communicated on the multiplexer channel.

The security property **P1** requires that the driver only receives news *after* he has logged into the news agent. There are two ways to log into the system so we have to ensure that independently of which one is chosen it is not possible to get any news (general or local); a closer inspection of the automaton on Figure 5 shows that this is indeed the case. Thus property **P1** can be validated.

Security property **P2** insists that the news agent never will learn the position of the car. This does *not* hold for the simple reason that it is sent to the news agent and a closer inspection of the bindings associated with the states q_9 and q_{10} show that here the name z used in NA_3 might indeed be bound to pos.

In order for property **P3** to be fulfilled we have to ensure that *only after* the news agent has received the gps position from the car it has knowledge it and thereby that this is the only way the news agent can obtain this information. To validate this we need to combine the techniques we have used before: First we have to inspect the transitions of the automaton to determine whether or not the car has submitted its position to the news agent; this is recorded by the presence of a transition labelled (4,9). The left most part of the automaton will then be reachable and as noted above the local news may be received by the car. The news agent forgets the knowledge of the position of the car as soon as the local news is sent to the multiplexer (cf. the recursive formulation of the process NA_3) and hence in the state q_{19} this information is no longer available and as we already have mentioned in states q_{36} and q_{38} the local news is ready to be forwarded to the car.

We now have to inspect the part of the automaton that can be reached without the message board of car transmitting the gps position to the multiplexer. This is the topmost and the rightmost part of the automaton of Figure 5. We have

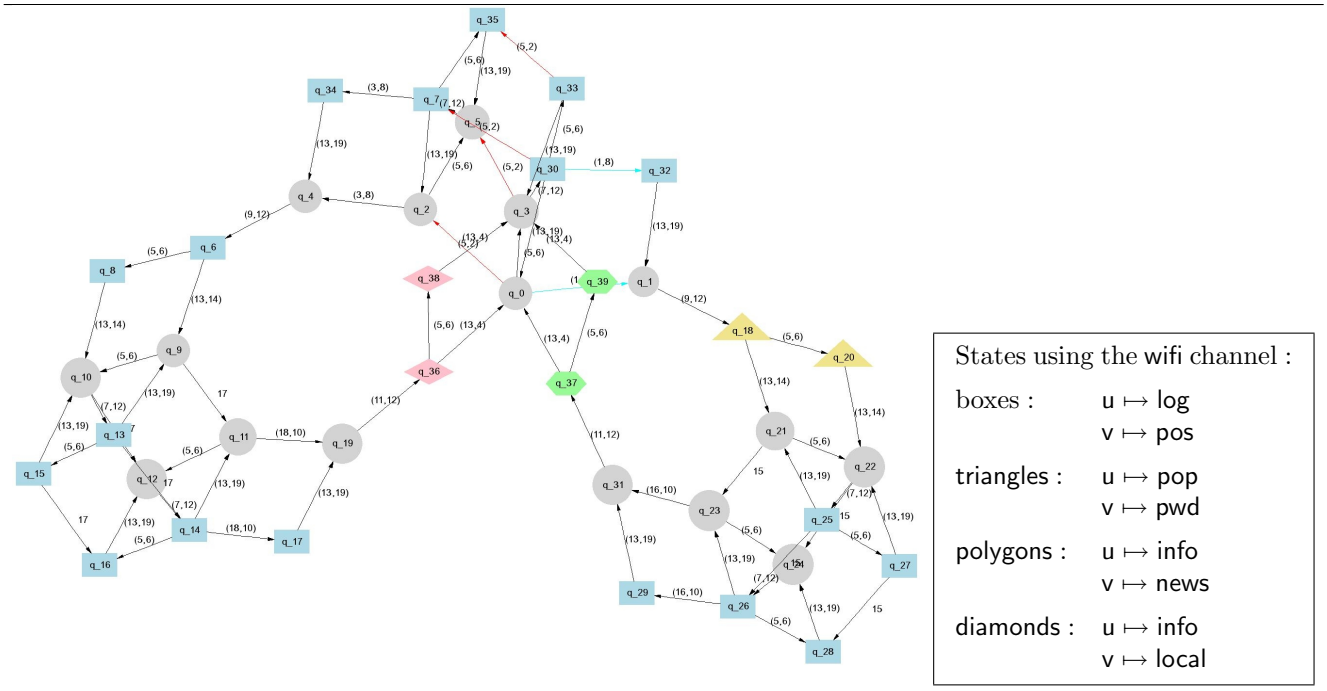


Figure 5. Analysis of the info-system with a news agent with local news: $\mathsf{MB}_5 \mid \mathsf{GPS}_2 \mid \mathsf{MP} \mid \mathsf{MP}' \mid \mathsf{LOG} \mid \mathsf{NA}_2$.

to ensure that the news agent does not have any information about the gps position in these states and as in the previous cases this amounts to a simple inspection of whether pos can be bound to the name z occurring in NA_3. Since this is not the case the property **P3** has been validated.

7. Conclusion

We have presented an analysis of the π-calculus that is flow sensitive as well as context sensitive. This has allowed us to validate a number of interesting properties related to privacy:

- **P1:** Sometimes it must be ensured that certain messages are only received after a certain login activity has taken place. — Our analysis is flow sensitive and can therefore correctly model the sequential order of the actions.
- **P2:** Sometimes it must be ensured that messages only reach the intended recipients. — Our analysis is context sensitive and can therefore correctly track the flow of information.
- **P3:** Sometimes it is allowed to "declassify" information for a certain purpose. — Our analysis is both flow sensitive and context sensitive and can therefore accurately model that a certain flow of information takes place after a given action.

As shown in the worked examples it is rather straightforward to validate the properties by inspecting the transitions of the automaton as well as information associated with the states.

More generally, our approach will scale to determine approximations to safety properties; because of the nature of the over-approximation performed during the development we cannot deal with properties that falls outside this class.

Our analysis is guaranteed to terminate for all processes, even those with an infinite state space. By carefully selecting the granularity function used in the analysis, it is possible to control the size of resulting automaton – and thereby both the complexity of the analysis and the precision of the analysis result. The resulting automata, as displayed in this paper, are generated in a fully automatic manner by our analysis engine programmed in Standard ML.

The analysis problem for the original military communication device, which inspired the development of the present scenario, was formulated in terms of a mandatory security policy to be validated against the Common Criteria: Two sources (corresponding to GPS and MB) are emitting data classified as *high* and *low*, respectively, and it is communicated over a multiplexed channel and must be correctly separated to a high and a low agent (corresponding to LOG

and NA). The security problem is thus to guarantee that the low data only is received by the low agent and similarly the high data is only received by the high agent. The analysis technique developed in this paper surely is powerful enough to provide the required guarantees; further work it needed to develop it for hardware programming language VHDL, in which the original system was developed.

Our contribution. Some of the basic ideas behind the analysis have previously been developed for CCS [9]. Our work substantially generalises the language primitives that can be dealt with: rather than only dealing with synchronisation we can deal with full polyadic communication.

Previous attempts [10] at dealing with communication have used a global control flow analysis for providing the bindings that may occur as the result of communications. Our work substantially improves the presision of this information by developing a transfer function that not only applies to exposed actions but also exposed bindings thereby essentially providing a localised control flow analysis in each state of the automaton.

Acknowledgements. We should like to thank the members of the *Language Based Technology* group at DTU for fruitful discussions. This work is supported by the SENSORIA project, an EU Integrated Project (IST-2005-016004) funded as part of the 6'th Framework Programme.

References

[1] M. Abadi and B. Blanchet. Computer-Assisted Verification of a Protocol for Certified Email. *Science of Computer Programming*, 58(1–2):3–27, October 2005. Special issue SAS'03.

[2] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Static validation of protocol narration. *Journal of Computer Security*, 13,3:347–390, 2005.

[3] C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Static analysis for the π-calculus with applications to security. *Information and Computation*, 168:68–92, 2001.

[4] R. Milner. *Communicating and Mobile Systems: The pi-Calculus*. Cambridge University Press, 1999.

[5] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, i. *Inf. Comput.*, 100(1):1–40, 1992.

[6] A. C. Myers and B. Liskov. A decentralized model for information flow control. In *Symposium on Operating Systems Principles*, pages 129–142, 1997.

[7] F. Nielson, H. Riis Nielson, and C. L. Hankin. *Principles of Program Analysis*. Springer, 1999. Second printing, 2005.

[8] F. Nielson, H. Riis Nielson, and R. R. Hansen. Validating firewalls using flow logics. *Theoretical Computer Science*, 283(2):381–418, 2002.

[9] H. Riis Nielson and F. Nielson. Data flow analysis for CCS. In T. Reps, M. Sagiv, and J. Bauer, editors, *Program Analysis and Compilation, Theory and Practice: Essays dedicated to Reinhard Wilhelm*, LNCS 4444, pages 311–327. Springer-Verlag, 2007.

[10] H. Pilegaard, F. Nielson, and H. Riis Nielson. Pathway Analysis for BioAmbients. *Submitted for publication*, 2007.

[11] F. Pottier. A simple view of type-secure information flow in the π-calculus. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 320–330, Cape Breton, Nova Scotia, June 2002.

[12] A. Sabelfeld and H. Mantel. Static confidentiality enforcement for distributed programs. In *Proc. Symp. on Static Analysis*, volume 2477 of *LNCS*, pages 376–394. Springer-Verlag, September 2002.

[13] G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In *Conference Record of POPL 98: The 25TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, California*, pages 355–364, New York, NY, 1998.

[14] T. K. Tolstrup, F. Nielson, and H. Riis Nielson. Information Flow Analysis for VHDL. In Victor E. Malyshkin, editor, *Proc. Eighth International Conference on Parallel Computing Technologies*, volume 3606 of *Lecture Notes in Computer Science*, pages 79–98. Springer-Verlag, September 2005.