Compositional Action Refinement and Information Flow Security[⋆]

Annalisa Bossi, Damiano Macedonio, Carla Piazza, and Sabina Rossi

Dipartimento di Informatica, Università Ca' Foscari di Venezia
via Torino 155, 30172 Venezia, Italy
{bossi,mace,piazza,srossi}@dsi.unive.it

Abstract. In the design process of distributed systems we may have to replace abstract specifications of components by more concrete specifications, thus providing more detailed design information. This well-known approach is often referred to as *action refinement*. In this paper we study the relationships between action refinement, compositionality, and information flow security within the Security Process Algebra (SPA). In particular, we first formalize the concept of action refinement in terms of context composition. We study the compositional properties of our notion of refinement and provide conditions under which information flow security properties are preserved through action refinement.

1 Introduction

It is common practice in software development of a complex system first describe it succinctly as a simple abstract specification and then refine it stepwise to a more concrete implementation. This hierarchical specification approach has been successfully developed for sequential systems where abstract-level instructions are expanded to finer-level descriptions until a concrete implementation is reached (see, e.g., [25]).

In the context of process algebra, this refinement methodology amounts to defining a mechanism for replacing abstract actions with more concrete processes. We adopt the terminology *action refinement* to refer to this stepwise development of systems specified as terms of a process algebra. We refer to [18] for a survey on the state of the art of action refinement in process algebra.

In this paper we model action refinement as a function *Ref* taking as arguments an action r to be refined, a system description E on a given level of abstraction and an interpretation of the action r on this level by a more complicated process F on a lower abstraction level. The refined process is obtained as context composition as described by the following simple example.

Let E be the process $r.b.\mathbf{0} + c.\mathbf{0}$ and r be the action we intend to refine by the process $F \equiv a_1.a_2.\mathbf{0}$. The refined process, denoted by $\mathit{Ref}(r, E, F)$, will be

⋆ This work has been partially supported by the MURST project "Modelli formali per la sicurezza" and the EU Contract IST-2001-32617 "Models and Types for Security in Mobile Distributed Systems" (MyThS).

the process $\tau.a_1.a_2.b.\mathbf{0} + c.\mathbf{0}$. We obtain it by context composition as $E'[F'[b.\mathbf{0}]]$ where $E'[X]$ is the context $\tau.X + c.\mathbf{0}$ while $F'[Y]$ is the context $a_1.a_2.Y$.

In practice, we follow the static syntactic approach to action refinement (see, e.g., [21]) reformulating it in terms of context composition. This characterization allows us to prove compositional properties of our notion of refinement by exploiting properties of open terms. Compositional properties are fundamental in the stepwise development of complex systems. They allow us to refine different sub-components of the system, while guaranteeing that the final result does not depend on the order in which the refinements are applied.

In system development, it is important to consider security related issues from the very beginning. Indeed, considering security only at the final step could lead to a poor protection or, even worst, could make it necessary to restart the development from scratch. On the other hand, taking into account security from the abstract specification level, better integrates it in the whole development process, possibly driving some implementation choices.

A security-aware stepwise development requires that the security properties of interest are preserved during the development steps, until a concrete (i.e., implementable) specification is obtained. Following this approach the security properties are guaranteed, and thus verified, by construction.

In this paper we first provide general conditions under which our notion of action refinement preserves security properties. Then we consider a bisimulation-based information flow security property named *P_BNDC* (*Persistent Bisimulation-based non Deducibility on Compositions*) [13]. We show how to both instantiate and extend the general results in the case of *P_BNDC* obtaining computable conditions ensuring that *P_BNDC* is preserved under action refinement.

The paper is organized as follows. In Section 2 we recall some basic notions of the *SPA* language. In Section 3 we define our notion of action refinement. Section 4 is devoted to the analysis of the compositional properties of our refinement. Some general results about the preservation of properties under refinement are presented in Section 5 and further analyzed in Subsection 5.1 for the security property *P_BNDC*. Finally, in Section 6 we discuss related works and draw some conclusions.

2 Basic Notions

We assume the reader familiar with the basic notions of Milner's CCS [20]. The *Security Process Algebra* (SPA) [11] is a variation of CCS where the set of visible actions is partitioned into two security levels, high and low, in order to specify multilevel systems. SPA syntax is based on the same elements as CCS, i.e.: a set $\mathcal{L} = I \cup O$ of *visible* actions where $I = \{a, b, \ldots\}$ is a set of *input* actions and $O = \{\bar{a}, \bar{b}, \ldots\}$ is a set of *output* actions; a special action τ which models internal computations, not visible outside the system; a function $\bar{\cdot} : \mathcal{L} \to \mathcal{L}$, such that $\bar{\bar{a}} = a$, for all $a \in \mathcal{L}$. $Act = \mathcal{L} \cup \{\tau\}$ is the set of all *actions*. The set of visible actions is partitioned into two sets, H and L, of high security actions and low

Prefix $$\frac{-}{a.E \xrightarrow{a} E}$$

Sum $$\frac{E_1 \xrightarrow{a} E_1'}{E_1 + E_2 \xrightarrow{a} E_1'} \qquad \frac{E_2 \xrightarrow{a} E_2'}{E_1 + E_2 \xrightarrow{a} E_2'}$$

Parallel $$\frac{E_1 \xrightarrow{a} E_1'}{E_1|E_2 \xrightarrow{a} E_1'|E_2} \qquad \frac{E_2 \xrightarrow{a} E_2'}{E_1|E_2 \xrightarrow{a} E_1|E_2'} \qquad \frac{E_1 \xrightarrow{\ell} E_1' \quad E_2 \xrightarrow{\bar{\ell}} E_2'}{E_1|E_2 \xrightarrow{\tau} E_1'|E_2'}$$

Restriction $$\frac{E \xrightarrow{a} E'}{E \setminus v \xrightarrow{a} E' \setminus v} \text{ if } a \notin v$$

Relabelling $$\frac{E \xrightarrow{a} E'}{E[f] \xrightarrow{f(a)} E'[f]}$$

Recursion $$\frac{T[recZ.T[Z]] \xrightarrow{a} E'}{recZ.T[Z] \xrightarrow{a} E'}$$

Fig. 1. The operational rules for SPA

security actions such that $\overline{H} = H$ and $\overline{L} = L$. The syntax of SPA *terms* is:

$$T ::= \mathbf{0} \mid Z \mid a.T \mid T + T \mid T|T \mid T \setminus v \mid T[f] \mid recZ.T$$

where Z is a variable, $a \in Act$, $v \subseteq \mathcal{L}$, $f : Act \rightarrow Act$ is such that $f(\bar{l}) = \overline{f(l)}$ for $l \in \mathcal{L}$, $f(\tau) = \tau$, $f(H) \subseteq H \cup \{\tau\}$, and $f(L) \subseteq L \cup \{\tau\}$.

A SPA *process* is a SPA term without free variables. We denote by $\mathcal{E}$ the set of all SPA processes.

The distinction between high and low security actions does not affect the operational semantics of SPA processes (see Figure 1) with respect to the corresponding CCS one . We denote by $(\mathcal{E}, Act, \rightarrow)$ the *Labelled Transition System* (LTS) which identifies it. We use the notations: $E \xrightarrow{a} E'$ to denote the transition labelled by a from E to E', $E \overset{a}{\Longrightarrow} E'$ to denote any sequence of transitions $E(\xrightarrow{\tau})^* \xrightarrow{a} (\xrightarrow{\tau})^* E'$ where $(\xrightarrow{\tau})^*$ denotes a (possibly empty) sequence of τ labelled transitions, and $E \overset{\hat{a}}{\Longrightarrow} E'$ which stands for $E \overset{a}{\Longrightarrow} E'$ if $a \in \mathcal{L}$, and for $E(\xrightarrow{\tau})^* E'$ if $a = \tau$.

The concept of *observation equivalence* is used to establish equalities among processes and it is based on the idea that two systems have the same semantics if and only if they cannot be distinguished by an external observer. This is obtained by defining an equivalence relation over $\mathcal{E}$ equating two processes when they are indistinguishable. In this paper we consider the relations named *weak bisimulation*, $\approx_B$, and *strong bisimulation*, $\sim_B$, defined by Milner for CCS [20].

They equates two processes if they are able to mutually simulate their behavior step by step. Weak bisimulation does not care about internal τ actions while strong bisimulation does. For the sake of completeness, we report its definition.

Definition 1 (Weak Bisimulation). *A symmetric binary relation $\mathcal{R} \subseteq \mathcal{E} \times \mathcal{E}$ over processes is a* weak bisimulation *if $(E, F) \in \mathcal{R}$ implies, for all $a \in Act$, if $E \xrightarrow{a} E'$, then there exists F' such that $F \overset{\hat{a}}{\Longrightarrow} F'$ and $(E', F') \in \mathcal{R}$.*

Two processes $E, F \in \mathcal{E}$ are weakly bisimilar*, denoted by $E \approx_B F$, if there exists a weak bisimulation $\mathcal{R}$ containing the pair (E, F).*

The relation $\approx_B$ is the largest weak bisimulation and it is an equivalence relation. *Strong bisimulation* [20] is similar to weak bisimulation, but it considers also τ actions. To do that it is sufficient use $\xrightarrow{a}$ instead of $\overset{\hat{a}}{\Longrightarrow}$ in the above definition.

We apply the standard notions of *free* and *bound* (occurrences of) variables in a SPA term. More precisely, all the occurrences of the variable Z in $recZ.T$ are *bound*; while Z is *free* in a term T if there is an occurrence of Z in T which is not bound. A SPA term with free variables can be seen as an environment with holes (the free occurrences of its variables) in which other SPA terms can be inserted. The result of this substitution is still a SPA term, which could be a process. For instance, in the term $h.\mathbf{0}|(l.X + \tau.\mathbf{0})$ we can replace the variable X with the process $\bar{h}.\mathbf{0}$ obtaining the process $h.\mathbf{0}|(l.\bar{h}.\mathbf{0} + \tau.\mathbf{0})$; or we can replace X by the term $a.Y$ obtaining the term $h.\mathbf{0}|(l.a.Y + \tau.\mathbf{0})$. When we consider a SPA term as an environment we call it *context.*

Definition 2 (Contexts). *A SPA* context *is a SPA term in which free variables may occur.*

Given a context C, we use the notation $C[Y_1, \ldots, Y_n]$ to stress the fact that we are interested only in the free occurrences of the variables $Y_1, \ldots, Y_n$ in C. The term $C[T_1, \ldots, T_n]$ is obtained from $C[Y_1, \ldots, Y_n]$ by replacing all the free occurrences of $Y_1, \ldots, Y_n$ with the terms $T_1, \ldots, T_n$, respectively. For instance, we can write $C[X] \equiv h.\mathbf{0}|(l.X + \tau.\mathbf{0})$ or $D[X] \equiv (l.X + \tau.\mathbf{0})|Y$ or $C'[X] \equiv Y|h.\mathbf{0}$. Hence, the notation $C[\bar{h}.\mathbf{0}]$ stands for $h.\mathbf{0}|(l.\bar{h}.\mathbf{0} + \tau.\mathbf{0})$, while $D[\bar{h}.\mathbf{0}] \equiv (l.\bar{h}.\mathbf{0} + \tau.\mathbf{0})|Y$ and $C'[\bar{h}.\mathbf{0}] \equiv Y|h.\mathbf{0}$. Note that the notation $C[Y_1, \ldots, Y_n]$ implies neither that all the variables $Y_1, \ldots, Y_n$ occur free in the context nor that they include all the variables occurring free in the context.

Following [20] we extend binary relations on processes to contexts as follows.

Definition 3 (Relations on Contexts). *Let $\mathcal{R}$ be a binary relation over processes, i.e., a subset of $\mathcal{E} \times \mathcal{E}$. Let C and D be two contexts and $\{Y_1, \ldots, Y_n\}$ be a set of variables which include all the free variables of C and D. We say that $C\ \mathcal{R}\ D$ if $C[E_1, \ldots, E_n]\ \mathcal{R}\ D[E_1, \ldots, E_n]$ for all set of processes $\{E_1, \ldots, E_n\}$.*

In the case of weak bisimulation, applying the above definition we have that two contexts are weakly bisimilar if all the processes obtained by instantiating their variables are pair-wise bisimilar. For instance, using our notation, the contexts $C[X] \equiv a.X + \tau.Y$ and $D[X] \equiv a.\tau.X + \tau.Y$ are weakly bisimilar since for all $E, F \in \mathcal{E}$ it holds $a.E + \tau.F \approx_B a.\tau.E + \tau.F$. Notice that not all the free variables of C and D were explicit in the notation $C[X]$ and $D[X]$. However, Definition 3 requires the instantiation of all their free variables.

Weak bisimulation is not a congruence, i.e., if two contexts $C[X]$ and $D[X]$ are weakly bisimilar, and two processes E and F are weakly bisimilar, then $C[E]$ and $D[F]$ are not necessarily weakly bisimilar. For instance, $\mathbf{0} \approx_B \tau.\mathbf{0}$, but $\mathbf{0} + a.\mathbf{0}$ is not weakly bisimilar to $\tau.\mathbf{0} + a.\mathbf{0}$. However, weak bisimulation is a congruence over the *guarded SPA* language, whose terms are defined as:

$$T ::= \mathbf{0} \mid Z \mid a.T \mid a.T + a.T \mid T|T \mid T \setminus v \mid T[f] \mid recZ.T.$$

In order to prove that weak bisimulation is a congruence over the guarded SPA we introduce the following technical lemma to deal with the recursive operator.

Lemma 1. *Let $C[Z, X_1, \dots, X_n]$ and $D[Z, X_1, \dots, X_n]$ be two contexts of the guarded SPA such that $C[Z, X_1, \dots, X_n] \approx_B D[Z, X_1, \dots, X_n]$, then*

$$recZ.C[Z, X_1, \dots, X_n] \approx_B recZ.D[Z, X_1, \dots, X_n].$$

Proof. Without loss of generality we can assume that $C[Z]$ and $D[Z]$ have at most the free variable Z. The general case follows from Definition 3.

From the hypothesis we have that $C[Z] \approx_B D[Z]$. Let us define the relation $\mathcal{S}$ on terms of the guarded SPA as follows:

$$\mathcal{S} = \{(G[recZ.C[Z]], G[recZ.D[Z]]) \mid G[X] \text{ context of the guarded } SPA, \text{ which contains at most one variable}\}.$$

It will be enough to show that $\mathcal{S}$ is a weak bisimulation up to $\approx_B$. From this it follows $recZ.C[Z] \approx_B recZ.D[Z]$, by taking $G \equiv X$. In particular, we prove that

$$\text{If } G[recZ.C[Z]] \xrightarrow{a} P \text{ then there exist } Q, Q' \text{ terms of the guarded SPA s.t. } G[recZ.D[Z]] \overset{\widehat{a}}{\Longrightarrow} Q \approx_B Q', \text{ with } (P, Q') \in \mathcal{S}.$$

The converse follows by the symmetry of $\mathcal{S}$.

We prove the claim by induction on the depth d of the inference used to obtain $G[recZ.C[Z]] \xrightarrow{a} P$.

Base: $d = 0$.
If $G[recZ.C[Z]] \xrightarrow{a} P$ with an inference of depth 0, then the rule "Prefix" has been applied, and $G[Z] \equiv a.G'[Z]$, so $P \equiv G'[recZ.C[Z]]$, and G' is a context in the guarded SPA. Hence, also $G[recZ.D[Z]] \equiv a.G'[recZ.D[Z]] \xrightarrow{a} G'[recZ.D[Z]]$ and we have that $(G'[recZ.C[Z]], G'[recZ.D[Z]]) \in \mathcal{S}$, as required.

Induction. We proceed by cases on the structure of the context G:

- $G \in \mathcal{E}$, is a SPA process. We have $G[recZ.C[Z]] \equiv G[recZ.D[Z]] \equiv G$, hence we immediately obtain the thesis.
- $G \equiv X$. Then $recZ.C[Z] \xrightarrow{a} P$ has been deduced by applying the "Recursion" rule at the last step. So $C[recZ.C[Z]] \xrightarrow{a} P$ with a shorter inference. Hence, by induction there exist Q, Q' terms of the guarded SPA, such that $C[recZ.D[Z]] \overset{\widehat{a}}{\Longrightarrow} Q \approx_B Q'$ with $(P, Q') \in \mathcal{S}$. But also $C[Z] \approx_B D[Z]$ and thus $D[recZ.D[Z]] \overset{\widehat{a}}{\Longrightarrow} Q'' \approx_B Q$. Since $D[recZ.D[Z]] \approx_B recZ.D[Z]$, we have that $recZ.D[Z] \overset{\widehat{a}}{\Longrightarrow} Q'''$ with $Q''' \approx_B Q'' \approx_B Q \approx_B Q'$.

– $G \equiv a_1.G_1 + a_2.G_2$. Then $a_1.G_1[recZ.C[Z]] + a_2.G_2[recZ.C[Z]] \xrightarrow{a} P$ by applying the "Sum" in the last step. So, there exists $i = 1, 2$ such that $a_i.G_i[recZ.C[Z]] \xrightarrow{a} P$. Hence, it must be $P \equiv G_i[recZ.C[Z]]$, with G_i context on the guarded SPA. By applying the same rules, $G[recZ.D[Z]] \stackrel{\widehat{a}}{\Longrightarrow} Q \equiv G_i[recZ.D[Z]]$, and $(P, Q) \in \mathcal{S}$.
– $G \equiv G_1 \setminus v$. Then $G_1[recZ.C[Z]] \setminus v \xrightarrow{a} P$ by applying the rule "Restriction" in the last step. So, $P \equiv P' \setminus v$, $a \notin v$ and $G_1[recZ.C[Z]] \xrightarrow{a} P'$ by a shorter inference. By induction on G_1, there exist Q, Q' in the guarded SPA such that $G_1[recZ.D[Z]] \stackrel{\widehat{a}}{\Longrightarrow} Q \approx_B Q'$ with $(E', Q') \in \mathcal{S}$. Hence, we conclude $G_1[recZ.D[Z]] \setminus v \stackrel{\widehat{a}}{\Longrightarrow} Q \setminus v$, with $Q \setminus v \approx_B Q' \setminus v$ and $(P, Q' \setminus v) \in \mathcal{S}$ by construction of $\mathcal{S}$. In fact, $(P', Q') \in \mathcal{S}$ implies that there exists a context $H[Z]$, with only a free variable Z, such that $P' \equiv H[recZ.C[Z]]$ and $Q' \equiv H[recZ.D[Z]]$. Hence, $P \equiv P' \setminus v \equiv H[recX.C[X]] \setminus v$ and $Q' \setminus v \equiv H[recX.D[X]] \setminus v$.
– $G \equiv G_1[f]$. Then $G_1[recZ.C[Z]][f] \xrightarrow{a} P$ by applying the rule "Relabelling" in the last step. So, $P \equiv P'[f]$, $a = f(a')$, and $G_1[recZ.C[Z]] \xrightarrow{a'} P'$ by a shorter inference. By induction there exist the terms Q, Q' in the guarded SPA, such that it holds $G_1[recZ.D[Z]] \stackrel{\widehat{a'}}{\Longrightarrow} Q \approx_B Q'$ with $(P', Q') \in \mathcal{S}$ Hence, we conclude $G_1[recZ.D[Z]][f] \stackrel{\widehat{f(a')}}{\Longrightarrow} Q[f]$, with $Q[f] \approx_B Q'[f]$ and $(P, Q'[f]) \in \mathcal{S}$ by construction.
– $G \equiv recY.G_1[X, Y]$. Then $recY.G_1[recZ.C[Z], Y] \xrightarrow{a} P$ by applying the rule "Recursion" in the last step. So, $G_1[recZ.C[Z], recY.G_1[recZ.C[Z]]] \xrightarrow{a} P$ with a shorter inference. Hence, by induction there exist the terms Q, Q in the guarded SPA such that $G_1[recZ.D[Z], recY.G_1[recZ.D[Z]]] \stackrel{\widehat{a}}{\Longrightarrow} Q \approx_B Q'$ with $(P, Q') \in \mathcal{S}$. Since

$$G_1[recZ.D[Z], recY.G_1[recZ.D[Z]]] \approx_B recY.G_1[recZ.D[Z], Y]$$

we conclude that $G_1[recZ.D[Z], recY.G_1[recZ.D[Z]]] \stackrel{\widehat{a}}{\Longrightarrow} Q'' \approx_B Q \approx_B Q'$.
□

Lemma 2. *Let $C[X_1, \ldots, X_n], E_1, F_1, \ldots, E_n, F_n$ be contexts of the guarded SPA. If $E_i \approx_B F_i$, for $i = 1, \ldots, n$, then*

$$C[E_1, \ldots, E_n] \approx_B C[F_1, \ldots, F_n].$$

Proof. If $E \approx_B F$, then it is immediate to prove that $a.E \approx_B a.F$. If $E_i \approx_B F_i$ for $i = 1, 2$, then $a_1.E_1 + a_2.E_2 \approx_B a_1.F_1 + a_2.F_2$. If $E_i \approx_B F_i$ for $i = 1, 2$, then $E_1 | E_2 \approx_B F_1 | F_2$. If $E \approx_B F$, then $E \setminus v \approx_B F \setminus v$ and $E[f] \approx_B F[f]$. If $E[Z] \approx_B F[Z]$, then $recZ.E[Z] \approx_B recZ.F[Z]$ by Lemma 1. Hence, we proved the thesis for the contexts $a.X, a.X + b.Y, X|Y, X \setminus v, X[f], recZ.X$. If $C[\bar{X}]$ is a more complex context, then we proceed by induction on the structure of C. $C[\bar{X}] \equiv C'[C_1[\bar{X}], \ldots, C_k[\bar{X}]]$, with $C', C_1, \ldots, C_k$ satisfying the inductive hypothesis. By inductive hypothesis we have that $C_i[\bar{E}] \approx_B C_i[\bar{F}]$, hence by inductive hypothesis we have that $C'[C_1[\bar{E}], \ldots, C_k[\bar{E}]] \approx_B C'[C_1[\bar{E}], \ldots, C_k[\bar{E}]]$, i.e., the thesis. $\square$

Theorem 1. *Let $C[X_1, \ldots, X_n], D[X_1, \ldots, X_n], E_1, F_1, \ldots, E_n, F_n$ be guarded SPA contexts. If $E_i \approx_B F_i$, for $i = 1, \ldots, n$, and $C \approx_B D$, then $C[E_1, \ldots, E_n] \approx_B D[F_1, \ldots, F_n]$.*

Proof.

$$\begin{aligned} C[E_1, \ldots, E_n] &\approx_B C[F_1, \ldots, F_n] && \text{by Lemma 2} \\ &\approx_B D[F_1, \ldots, F_n] && \text{since } C \approx_B D. \end{aligned}$$

$\square$

Strong bisimulation is a congruence with respect to all the contexts, i.e., if $C \sim_B D$ and $E_i \sim_B F_i$, then $C[E_1, \ldots, E_n] \sim_B D[F_1, \ldots, F_n]$.

In general a property $\mathcal{P}$ is nothing but a class of processes, i.e., the class of processes which satisfy $\mathcal{P}$. We extend this concept to contexts as follows.

Definition 4 ($\mathcal{P}$-contexts). *Let $\mathcal{P}$ be a class of processes and $C[X_1, \ldots, X_n]$ be a context whose free variables are in $\{X_1, \ldots, X_n\}$. $C[X_1, \ldots, X_n]$ is said to be a $\mathcal{P}$-*context *if for all $E_1, \ldots, E_n \in \mathcal{P}$ it holds that $C[E_1, \ldots, E_n] \in \mathcal{P}$.*

3 Action Refinement

It is standard practice in software development to obtain the final program starting from an abstract, possibly not executable, specification by successive refinements steps. Abstract operations are replaced by more detailed programs which can be further refined, until a level is reached where no more abstractions occur.

In the context of process algebra, this stepwise development amounts to interpreting actions on a higher level of abstraction by more complicated processes on a lower level. This is obtained by introducing a mechanism to translate actions into processes. There are several ways to do this. We adopt the syntactic approach and define the refinement step as a syntactic process transformation. Given a process E in which there is an occurrence of an abstract action r the idea is to refine E by replacing r with a process F. This requires to introduce a suitable operation which realizes the necessary links from the parts of E which precede an occurrence of r and the parts of E which follow that occurrence. In other words we have to hook F to E, whenever an action r occurs.

To define this transformation we need some definitions and some syntactical operations. We first introduce the concepts of *free*, *bound* and *refinable* actions.

Definition 5 (Free, Bound and Refinable actions). *Let T be a SPA term. The set of* free *actions of T, denoted by $f(T)$ in inductively defined as follows:*

$\mathit{free}(\mathbf{0}) = \emptyset$;	$\mathit{free}(Z) = \emptyset$ *where Z is a variable*;
$\mathit{free}(a.T) = \{a\} \cup \mathit{free}(T)$;	$\mathit{free}(T_1 + T_2) = \mathit{free}(T_1) \cup \mathit{free}(T_2)$;
$\mathit{free}(T_1 \vert T_2) = \mathit{free}(T_1) \cup \mathit{free}(T_2)$;	$\mathit{free}(T \setminus v) = \mathit{free}(T) \setminus v$;
$\mathit{free}(T[f]) = \mathit{free}(T) \setminus \{a \mid f(a) \neq a\}$;	$\mathit{free}(recZ.T) = \mathit{free}(T)$.

An action occurring in T is said to be bound *if it is not free. We denote by $b(T)$ the set of bound actions of T. An action r is said to be* refinable *in T if it does not occur bound in T and $\bar{r}$ does not occur in T.*

Then we define the set of the parts of E which syntactically follow the outermost occurrences of an action r, and the context $E\{r\}$ which represents the part of E before the outermost occurrences of r.

Definition 6 ($E@r$ and $E\{r\}$). *Let E be a SPA term and r be an action occurring in E. The set of terms $E@r$ is inductively defined as follows:*

$$\begin{array}{ll}
\mathbf{0}@r = \emptyset; & Z@r = \emptyset; \\
(r.T)@r = \{T\}; & (a.T)@r = T@r, \ \text{if } a \neq r; \\
(T_1 + T_2)@r = T_1@r \cup T_2@r; & (T_1|T_2)@r = T_1@r \cup T_2@r; \\
(T \setminus v)@r = T@r; & (T[f])@r = T@r; \\
(recZ.T)@r = T@r. &
\end{array}$$

Let $E@r = \{T_1, \ldots, T_n\}$ and $X_{T_1}, \ldots, X_{T_n}$ be variables which do not occur in E. The context $E\{r\}$ is inductively defined as follows:

$$\begin{array}{ll}
\mathbf{0}\{r\} = \mathbf{0}; & Z\{r\} = Z; \\
(r.T)\{r\} = \tau.X_T; & (a.T)\{r\} = a.(T\{r\}), \ \text{if } a \neq r; \\
(T_1 + T_2)\{r\} = T_1\{r\} + T_2\{r\}; & (T_1|T_2)\{r\} = T_1\{r\}|T_2\{r\}; \\
(T \setminus v)\{r\} = (T\{r\}) \setminus v; & (T[f])\{r\} = (T\{r\})[f]; \\
(recZ.T)\{r\} = recZ.(T\{r\}). &
\end{array}$$

Example 1.

- Let $E \equiv r.\mathbf{0}|a.\mathbf{0}$. We have that $E@r$ is $\{\mathbf{0}\}$ and $E\{r\}$ is $\tau.X_{\mathbf{0}}|a.\mathbf{0}$.
- Let $E \equiv (a.r.\mathbf{0} + b.r.c.r.a.\mathbf{0}) \mid r.\mathbf{0}$. The set $E@r$ contains two processes and is equal to $\{\mathbf{0}, c.r.a.\mathbf{0}\}$. Note that the term $c.r.a.\mathbf{0}$ in $E@r$ contains an occurrence of r. The context $E\{r\}$ is $(a.\tau.X_{\mathbf{0}} + b.\tau.X_{c.r.a.\mathbf{0}}) \mid \tau.X_{\mathbf{0}}$. The set of the free variables of $E\{r\}$ is exactly $\{X_T \mid T \in E@r\}$.
- Let $E \equiv recZ.(a.Z+r.Z)$. We have that $E@r$ is $\{Z\}$ and $E\{r\}$ is $recZ.(a.Z+\tau.X_Z)$. In this case $E@r$ has only one element which is not a process.

Definition 7 (F^Y). *Let F be a term and Y be a variable not occurring in F. F^Y is the context obtained by replacing each occurrence of $\mathbf{0}$ in F with Y.*

The refinement of an action r in a term E with a term F is obtained by successive context composition as follows.

Definition 8 (Refinement of r in E with F). *Let E be a term, $r \in Act$ an action refinable in E, and F be a term which can refine r in E, that is, such that $b(E) \cap free(F) = \emptyset$ and r and $\bar{r}$ do not occur in F. Let Y be a variable which does not occur neither in E nor in $E\{r\}$. Let $E@r = \{T_1, \ldots, T_n\}$. The* partial refinement $ParRef(r, E, F)$ of r in E with F *is defined as*

$$ParRef(r, E, F) = E\{r\}[F^Y[T_1], \ldots, F^Y[T_n]].$$

The refinement $Ref(r, E, F)$ of r in E with F *is*

- *$ParRef^0(r, E, F) = E$, if r does not occur in E;*
- *$ParRef^1(r, E, F) = ParRef(r, E, F)$, if r occurs once in E;*
- *$ParRef^{n+1}(r, E, F) = ParRef(r, ParRef^n(r, E, F), F)$, if r occurs $n + 1$ times in E.*

Intuitively $E@r$ are the parts of E which syntactically follow the occurrences of the action r, while $E\{r\}$ is the part of E which precedes the r's. The holes X_T's in $E\{r\}$ serve to hook the refinement F. Similarly the free variable Y of F^Y serves to hook the elements of $E@r$ after the execution of F. The partial refinement $ParRef(r, E, F)$ replaces in E as many occurrences as possible of r with F. When one occurrence of r is followed by another occurrence of r (e.g., $r.a.r.\mathbf{0}$) the partial refinement replaces only the first occurrence. Hence in order to replace all the occurrences in the worst case it is necessary to compute the partial refinement n times, where n is the number of occurrences of r in E. This is equivalent to say that our definition introduces a partial order between the occurrences of r, and it replaces the r's following this partial order. We would obtain the same result by arbitrarily choosing at each step one occurrence of r replacing it with F, and going on until there are no more occurrences of r.

Notice that even if E is a process $E@r$ can be a set of terms with free variables (see Example 1), while the X_T's are always the only free variables occurring in $E\{r\}$. Hence, if E is a process, then $Ref(r, E, F)$ is a process.

Example 2. We consider again the three processes of Example 1.

- Let $E \equiv r.\mathbf{0}|a.\mathbf{0}$ and $F \equiv b_1.b_2.\mathbf{0}$. The refinement $Ref(r, E, F)$ is equal to $ParRef(r, E, F)$ and it is the process $(\tau.b_1.b_2.\mathbf{0})|a.\mathbf{0}$. It is worth noticing that in the refined process the action a can be performed *before* the expansion of action r is finished. In fact, our refinement is not atomic.
- Let $E \equiv (a.r.\mathbf{0} + b.r.c.r.a.\mathbf{0}) \mid r.0$ and $F \equiv e.f.\mathbf{0}$. The partial refinement $ParRef(r, E, F)$ is the the process $E' \equiv a.\tau.e.f.\mathbf{0} + b.\tau.e.f.c.r.a.\mathbf{0}) \mid \tau.e.f.\mathbf{0}$. Since the context $E'\{r\}$ is $(a.\tau.e.f.\mathbf{0} + b.\tau.e.f.c.\tau.X_{a.\mathbf{0}}) \mid \tau.e.f.\mathbf{0}$, $Ref(r, E, F)$ is $ParRef(r, E', F) = a.\tau.e.f.\mathbf{0} + b.\tau.e.f.c.\tau.e.f.a.\mathbf{0}) \mid \tau.e.f.\mathbf{0}$.
- Let $E \equiv recZ.(a.Z + r.Z)$ and $F \equiv b.c.\mathbf{0}$. The refinement $Ref(r, E, F)$ is the process $recZ.(a.Z + \tau.b.c.Z)$.

Notice that the refinement $Ref(r, E, F)$ is defined only if r is refinable in E and F can refine r in E. From now on when we write $Ref(r, E, F)$ we always tacitly assume that r, E, and F are such that the refinement is defined. Similarly, when we write $E@r$ or $E\{r\}$ we assume that r is refinable in E.

The refinement introduced in Definition 8 is based on syntactic substitutions of subterms in SPA language. It is worth noticing that it is not the straightest syntactic substitution: the action r is substituted by $\tau.F$ and not by F. Thus, for instance, $Ref(r, r.\mathbf{0} + a.b.\mathbf{0}, c.\mathbf{0} + d.\mathbf{0})$ is $\tau.(c.\mathbf{0} + d.\mathbf{0}) + a.b.\mathbf{0}$ instead of $c.\mathbf{0} + d.\mathbf{0} + a.b.\mathbf{0}$. Our choice is motivated by the idea that the implementation of an abstract action r by means of a more complex process F requires first *calling* F and then executing it. This choice has also the nice consequence of preserving guarded terms. In fact, it is easy to prove that if E is a guarded term then also

$E\{r\}$ is guarded. This property would not hold if X_T instead of $\tau.X_T$ would be used to abstract the action r.

As many other syntactic refinements (see [18]) our refinement is not atomic and it does not preserve all semantic properties. In particular, it preserves neither weak nor strong bisimulation, as shown by the following example.

Example 3.

- Let $E_1 \equiv r.\mathbf{0}|a.\mathbf{0}$ and $E_2 \equiv r.a.\mathbf{0} + a.r.\mathbf{0}$. E_1 and E_2 are strongly bisimilar. Let $F \equiv b.c.\mathbf{0}$. We have that $\mathit{Ref}(r, E_1, F)$ and $\mathit{Ref}(r, E_2, F)$ are not even weakly bisimilar. In particular, the first process can perform the sequence of actions $b.a.c.\mathbf{0}$, while the second cannot.
- Let $E \equiv a.r.b.c.\mathbf{0}$, $F_1 \equiv d.\mathbf{0}|e.\mathbf{0}$ and $F_2 \equiv d.e.\mathbf{0} + e.d.\mathbf{0}$. We have that F_1 and F_2 are strongly bisimilar, but $\mathit{Ref}(r, E, F_1)$ is not strongly bisimilar to $\mathit{Ref}(r, E, F_2)$.

However, exploiting Theorem 5 (see next section) we get the following result.

Theorem 2. *Let E, F_1, F_2 be terms. If $F_1^Y \sim_B F_2^Y$, then $\mathit{Ref}(r, E, F_1) \sim_B \mathit{Ref}(r, E, F_2)$. If E, F_1, F_2 are terms of the guarded SPA and $F_1^Y \approx_B F_2^Y$, then $\mathit{Ref}(r, E, F_1) \approx_B \mathit{Ref}(r, E, F_2)$.*

Proof. By induction on the structure of E.
If either $E \equiv Z$ or $E \equiv \mathbf{0}$, we immediately get the thesis.
If $E \equiv r.E_1$, then

$\mathit{Ref}(r, r.E_1, F_1) \equiv$	by Theorem 5
$\tau.F_1^Y[\mathit{Ref}(r, E_1, F_1)] \sim_B$	since $F_1^Y \sim_B F_2^Y$
$\tau.F_2^Y[\mathit{Ref}(r, E_1, F_1)] \sim_B$	by induction
$\tau.F_2^Y[\mathit{Ref}(r, E_1, F_2)] \equiv$	by Theorem 5
$\mathit{Ref}(r, r.E_1, F_2)$	

If $E \equiv E_1 + E_2$, then

$\mathit{Ref}(r, E_1 + E_2, F_1) \equiv$	by Theorem 5
$\mathit{Ref}(r, E_1, F_1) + \mathit{Ref}(r, E_2, F_1) \sim_B$	by induction
$\mathit{Ref}(r, E_1, F_2) + \mathit{Ref}(r, E_2, F_2) \equiv$	by Theorem 5
$\mathit{Ref}(r, E_1 + E_2, F_2)$	

The other cases are similar.

The second part of the thesis follows similarly exploiting also Theorem 1. □

Moreover even if from $E_1 \sim_B E_2$ we do not get that $\mathit{ParRef}(r, E_1, F) \sim_B \mathit{ParRef}(r, E_2, F)$ (see Example 3) we can preserve the equivalence provided that it holds also between the contexts $E_1\{r\}$ and $E_2\{r\}$.

Theorem 3. *Let E_1, E_2, F be terms. If $E_1\{r\} \sim_B E_2\{r\}$, then it holds that $\mathit{ParRef}(r, E_1, F) \sim_B \mathit{ParRef}(r, E_2, F)$. If E_1, E_2 are terms of the guarded SPA and $E_1\{r\} \approx_B E_2\{r\}$, then $\mathit{ParRef}(r, E_1, F) \approx_B \mathit{ParRef}(r, E_2, F)$.*

Proof. The first part of the thesis follows from the fact that $\sim_B$ is a congruence. The second part of the thesis follows from Theorem 1. □

It is worth noticing that we cannot change the statement by considering *Ref* instead of *ParRef*. It can be $\mathit{Ref}(r, E_1, F)\{r\} \not\sim_B \mathit{Ref}(r, E_2, F)\{r\}$ even if $\mathit{ParRef}(r, E_1, F) \sim_B \mathit{ParRef}(r, E_2, F)$. For instance taking the processes $E_1 \equiv r.(r.\mathbf{0}|a.\mathbf{0})$, $E_2 \equiv r.(a.r.\mathbf{0}+r.a.\mathbf{0})$ and $F \equiv b.c.\mathbf{0}$ we have that $\mathit{ParRef}(r, E_1, F) \sim_B \mathit{ParRef}(r, E_2, F)$, but the contexts $\mathit{ParRef}(r, E_1, F)\{r\}$ and $\mathit{ParRef}(r, E_2, F)\{r\}$ are not bisimilar. Thus, we cannot iterate the above reasoning. We could avoid this problem by adding the constraint that no element in $E@r$ contains r.

By applying both the above results we immediately get the corollary below.

Corollary 1. *Let E_1, E_2, F_1, and F_2 be terms. If $E_1\{r\} \sim_B E_2\{r\}$ and $F_1^Y \sim_B F_2^Y$, then $\mathit{ParRef}(r, E_1, F_1) \sim_B \mathit{ParRef}(r, E_2, F_2)$. If E_1, E_2, F_1, F_2 are terms of the guarded SPA such that $E_1\{r\} \approx_B E_2\{r\}$ and $F_1^Y \approx_B F_2^Y$, then it holds $\mathit{ParRef}(r, E_1, F_1) \approx_B \mathit{ParRef}(r, E_2, F_2)$.*

4 Action Refinement and Compositionality

At any fixed abstraction level during the top-down development of a program, it is unrealistic to think that there is just one action to be refined at that level. Usually, different abstract actions coexist, all of them have to be refined, and we do not want to worry about the specific ordering in which the refinements occur. This is guaranteed only if the refinement operation enjoys compositional properties. Here we show some of the compositional properties of our refinement.

First we show that our refinement is local to the components in which the action to be refined occurs. This is a consequence of the following theorem.

Theorem 4. *Let $E_1, \ldots, E_n$ and F be terms. Let $C[Z_1, \ldots, Z_n]$ be a context with no occurrences of r and $\bar{r}$. It holds*

$$\mathit{Ref}(r, C[E_1, \ldots, E_n], F) \;\equiv\; C[\mathit{Ref}(r, E_1, F), \ldots, \mathit{Ref}(r, E_n, F)].$$

Proof. We consider the case $n = 1$, the other cases are similar. The theorem, in this case, follows from the following claim by choosing m as the number of occurrences of r in E.

Claim. For every $m \geq 0$, it holds $C[\mathit{ParRef}^m(r, E, F)] \equiv \mathit{ParRef}^m(r, C[E], F)$.

Proof. The proof follows by induction on m. The base $m = 0$ is trivial. The case $m > 1$ of the inductive step is a consequence of functional composition. To handle the case $m = 1$ we proceed by induction on the structure of C.
If $C \equiv Z$, then we immediately have the thesis.
If $C \equiv C_1 + C_2$, then

$$\begin{aligned} C[\mathit{ParRef}(r, E, F)] &\equiv C_1[\mathit{ParRef}(r, E, F)] + C_2[\mathit{ParRef}(r, E, F)] \\ &\equiv \mathit{ParRef}(r, C_1[E], F) + \mathit{ParRef}(r, C_2[E], F), \end{aligned}$$

this last applying the definition of refinement is syntactically equivalent to $\mathit{ParRef}(r, C[E], F)$, i.e., we have the thesis. All the other cases are similar. □

Hence, if we have a term G which is of the form $E_1|E_2|\dots|E_n$ and the action r occurs only in E_i it is sufficient to apply the refinement to E_i to obtain $\mathit{Ref}(r, G, F) \equiv E_1|E_2|\dots|\mathit{Ref}(r, E_i, F)|\dots|E_n$.

Example 4. Let us consider the process $G \equiv \mathit{recV}.(a.V + \mathit{recW}.(a.W + r.W))$. We can decompose it into $C[Z] \equiv \mathit{recV}(a.V + Z)$ and $E \equiv \mathit{recW}.(a.W + r.W)$ and apply the refinement to E. For instance, if $F \equiv b.c.\mathbf{0}$ we get that $\mathit{Ref}(r, E, F) \equiv \mathit{recW}.(a.W + \tau.b.c.W)$. Hence, $\mathit{Ref}(r, G, F) \equiv \mathit{recV}.(a.V + \mathit{recW}.(a.W + \tau.b.c.W))$.

Instead of applying directly the definition of refinement it is possible to compute the refinement by induction on the structure of the process E to be refined, as shown by the following theorem.

Theorem 5. *Let E and F be terms.*

$$\begin{array}{ll}
\mathit{Ref}(r, \mathbf{0}, F) & \equiv \mathbf{0}; \\
\mathit{Ref}(r, Z, F) & \equiv Z; \\
\mathit{Ref}(r, r.E, F) & \equiv \tau.F^Y[\mathit{Ref}(r, E, F)]; \\
\mathit{Ref}(r, a.E, F) & \equiv a.\mathit{Ref}(r, E, F), \;\; \textit{if } a \neq r; \\
\mathit{Ref}(r, E_1 + E_2, F) & \equiv \mathit{Ref}(r, E_1, F) + \mathit{Ref}(r, E_2, F); \\
\mathit{Ref}(r, E_1|E_2, F) & \equiv \mathit{Ref}(r, E_1, F)|\mathit{Ref}(r, E_2, F); \\
\mathit{Ref}(r, E[f], F) & \equiv \mathit{Ref}(r, E, F)[f]; \\
\mathit{Ref}(r, E \setminus v, F) & \equiv \mathit{Ref}(r, E, F) \setminus v; \\
\mathit{Ref}(r, \mathit{recZ}.E_1, F) & \equiv \mathit{recZ}.\mathit{Ref}(r, E_1, F).
\end{array}$$

Proof. First, we prove the following Claim.

Claim. Let $n \geq 0$. Then,

1. $\mathit{ParRef}^n(r, a.E, F) \equiv a.\mathit{ParRef}^n(r, E, F)$, if $a \neq r$;
2. $\mathit{ParRef}^n(r, E_1 + E_2, F) \equiv \mathit{ParRef}^n(r, E_1, F) + \mathit{ParRef}^n(r, E_2, F)$;
3. $\mathit{ParRef}^n(r, E_1 \mid e_2, F) \equiv \mathit{ParRef}^n(r, E_1, F) \mid \mathit{ParRef}^n(r, E_2, F)$;
4. $\mathit{ParRef}^n(r, E[f], F) \equiv \mathit{ParRef}^n(r, E, F)[f]$;
5. $\mathit{ParRef}^n(r, E \setminus v, F) \equiv \mathit{ParRef}^n(r, E, F) \setminus v$;
6. $\mathit{ParRef}^n(r, \mathit{recZ}.E_1, F) \equiv \mathit{recZ}.\mathit{ParRef}^n(r, E_1, F)$;
7. $\mathit{ParRef}^{n+1}(r, r.E, F) \equiv \tau.F^Y[\mathit{ParRef}^n(r, E, F)]$.

Proof. Properties from 1 to 6 follow from the Claim inside the proof of Theorem 4. In order to prove the last one we proceed by induction on $n \geq 1$.

Base $n = 1$.

$$\begin{array}{lll}
\mathit{ParRef}(r, r.E, F) & \equiv \tau.F^Y[E] & \text{since } r.E\{r\} = \tau.X_E \\
 & \equiv \tau.F^Y[\mathit{Ref}(r, E, F)] & \text{since } r \text{ does not occur in } E.
\end{array}$$

Inductive step, let $n \geq 2$.

$$\begin{array}{lll}
\mathit{ParRef}^n(r, r.E, F) & \equiv \mathit{ParRef}(r, \mathit{ParRef}^{n-1}(r, r.E, F), F) & \\
 & \equiv \mathit{ParRef}^{n-1}(r, \mathit{ParRef}(r, r.E, F), F) & \text{by funct. composition} \\
 & \equiv \mathit{ParRef}^{n-1}(r, \tau.F^Y[E], F) & \text{since } r.E\{r\} = \tau.X_E \\
 & \equiv \tau.F^Y[\mathit{ParRef}^{n-1}(r, E, F)] & \text{by point 1.}
\end{array}$$

□

Our theorem follows by choosing n as the number of occurrences of r in E. $\square$

If we need to refine two actions in a process E, then the order in which we apply the refinements does not matter.

Theorem 6. *Let E be a term. Let F_1 and F_2 be two terms with no occurrences of r_1, r_2, $\bar{r}_1$, and $\bar{r}_2$.*

$$Ref(r_2, Ref(r_1, E, F_1), F_2) \equiv Ref(r_1, Ref(r_2, E, F_2), F_1).$$

Proof. We proceed by induction on the structure of E.
Base. The cases $\mathbf{0}$ and X are trivial.
Inductive step. The more interesting case is $r_1.E$
$Ref(r_2, Ref(r_1, r_1.E, F_1), F_2)$
$\equiv Ref(r_2, \tau.F_1^Y[Ref(r_1, E, F_1)], F_2)$ by Theorem 5
$\equiv \tau.Ref(r_2, F_1^Y[Ref(r_1, E, F_1)], F_2)$ by Theorem 5
$\equiv \tau.F_1^Y[Ref(r_2, Ref(r_1, E, F_1), F_2)$ by Theorem 4, since r_2 is not in F_1
$\equiv \tau.F_1^Y[Ref(r_1, Ref(r_2, E, F_2), F_1)$ by induction
$\equiv Ref(r_1, r_1.Ref(r_2, E, F_2), F_1)$ by Theorem 5
$\equiv Ref(r_1, Ref(r_2, r_1.E, F_2), F_1)$ by Theorem 5

The case $r_2.E$ is symmetric. All the other cases have to be treated in the same way by applying Theorem 5. Let us only see the case $E_1 + E_2$
$Ref(r_2, Ref(r_1, E_1 + E_2, F_1), F_2)$
$\equiv Ref(r_2, Ref(r_1, E_1, F_1) + Ref(r_1, E_2, F_1), F_2)$
$\equiv Ref(r_2, Ref(r_1, E_1, F_1), F_2) + Ref(r_2, Ref(r_1, E_2, F_1), F_2)$
$\equiv Ref(r_1, Ref(r_2, E_1, F_2), F_1) + Ref(r_1, Ref(r_2, E_2, F_2), F_1)$ by induction
$\equiv Ref(r_1, Ref(r_2, E_1 + E_2, F_2), F_1)$
$\square$

Example 5. Let $E \equiv r_1.a.\mathbf{0} + r_2.b.r_2.\mathbf{0}$, $F_1 \equiv b.\mathbf{0}$ and $F_2 \equiv c.\mathbf{0}$. We have that $Ref(r_2, Ref(r_1, E, F_1), F_2) \equiv ParRef^2(r_2, \tau.b.a.\mathbf{0} + r_2.b.r_2.\mathbf{0}, F_2) \equiv$ $ParRef(r_2, ParRef(r_2, \tau.b.a.\mathbf{0} + r_2.b.r_2.\mathbf{0}, F_2), F_2) \equiv \tau.b.a.\mathbf{0} + \tau.c.b.\tau.c.\mathbf{0} \equiv$ $Ref(r_1, r_1.a.\mathbf{0} + \tau.c.b.\tau.c.\mathbf{0}, F_1) \equiv Ref(r_1, ParRef^2(r_2, E, F_2), F_1) \equiv$ $Ref(r_1, Ref(r_2, E, F_2), F_1)$.

Moreover, we can refine r_1 in E using F_1 and r_2 in F_1 using F_2 independently from the order in which the refinements are applied.

First we extend Theorem 4.

Lemma 3. *Let $C[Z_1, \ldots, Z_n]$ be a context. Let $E_1, \ldots, E_n, F$ be terms.*

$$Ref(r, C[E_1, \ldots, E_n], F) \equiv Ref(r, C, F)[Ref(r, E_1, F), \ldots, Ref(r, E_n, F)].$$

Proof. We prove the thesis by induction on the structure of C in the case $n = 1$. The general case is similar.
If C has no occurrences of Z, then we immediately have the thesis.
If $C \equiv Z$, then we have the thesis.

If $C \equiv a.D[Z]$ with $a \neq r$, then

$$\begin{array}{ll}
Ref(r, a.D[E], F) \equiv & \text{by Theorem 5} \\
a.Ref(r, D[E], F) \equiv & \text{by induction} \\
a.Ref(r, D, F)[Ref(r, E, F)] \equiv & \text{by Theorem 5} \\
Ref(r, a.D, F)[Ref(r, E, F)] &
\end{array}$$

If $C \equiv r.D[Z]$, then

$$\begin{array}{ll}
Ref(r, r.D[E], F) \equiv & \text{by Theorem 5} \\
\tau.F^Y[Ref(r, D[E], F)] \equiv & \text{by induction} \\
\tau.F^Y[Ref(r, D, F)[Ref(r, E, F)]] \equiv & \text{by Theorem 5} \\
Ref(r, r.D, F)[Ref(r, E, F)] &
\end{array}$$

If $C \equiv H[Z] + K[Z]$, then

$$\begin{array}{ll}
Ref(r, H[E] + K[E], F) \equiv & \text{by Theorem 5} \\
Ref(r, H[E], F) + Ref(r, K[E], F) \equiv & \text{by induction} \\
Ref(r, H, F)[Ref(r, E, F)] + Ref(r, K, F)[Ref(r, E, F)] \equiv & \text{by Theorem 5} \\
Ref(r, H + K, F)[Ref(r, E, F)] &
\end{array}$$

All the other cases are similar. □

We need also the following technical lemma.

Lemma 4. *Let E, F_1, F_2 be terms such that F_2 has no occurrences of r_1 and $\bar{r}_1$.*

$$Ref(r_2, ParRef(r_1, r_1.E, F_1), F_2) \equiv \\ ParRef(r_1, Ref(r_2, r_1.E, F_2), Ref(r_2, F_1, F_2)).$$

Proof. We first prove the following claim.

Claim.
$$Ref(r, F^Y, G) \equiv Ref(r, F, G)^Y.$$

Proof. By induction on the structure of F.
The only interesting case is $F \equiv r.H$.

$$\begin{array}{ll}
Ref(r, r.H^Y, G) \equiv & \text{by Theorem 5} \\
\tau.G^Z[Ref(r, H^Y, G)] \equiv & \text{by induction} \\
\tau.G^Z[Ref(r, H, G)^Y] \equiv & \text{by Def. of } F^Y \\
(\tau.G^Z[Ref(r, H, G)])^Y \equiv & \text{by Theorem 5} \\
Ref(r, r.H, G)^Y &
\end{array}$$

□

We are now ready to prove the lemma.

$$\begin{array}{ll}
Ref(r_2, ParRef(r_1, r_1.E, F_1), F_2) \equiv & \text{by Def. of ParRef} \\
Ref(r_2, \tau.F_1^Y[E], F_2) \equiv & \text{by Lemma 3} \\
Ref(r_2, \tau.F_1^Y, F_2)[Ref(r_2, E, F_2)] \equiv & \text{by Theorem 5} \\
\tau.Ref(r_2, F_1^Y, F_2)[Ref(r_2, E, F_2)] \equiv & \text{by the above claim} \\
\tau.Ref(r_2, F_1, F_2)^Y[Ref(r_2, E, F_2)] \equiv & \text{by Def. of ParRef} \\
ParRef(r_1, r_1.Ref(r_2, E, F_2), Ref(r_2, F_1, F_2)) \equiv & \text{by Theorem 5} \\
ParRef(r_1, Ref(r_2, r_1.E, F_2), Ref(r_2, F_1, F_2)) &
\end{array}$$

□

Theorem 7. *Let E, F_1, F_2 be terms such that r_1 and $\bar{r}_1$ do not occur in F_2.*

$$Ref(r_2, Ref(r_1, E, F_1), F_2) \equiv Ref(r_1, Ref(r_2, E, F_2), Ref(r_2, F_1, F_2)).$$

Proof. We proceed by induction on the structure of the term E.
The cases $\mathbf{0}, Z, G_1 + G_2, G_1|G_2, G[f], G\backslash v, recZ.G, a.G$, with $a \neq r_1$ immediately follows from Theorems 4 and 5.
Let $E \equiv r_1.G$.
$Ref(r_2, Ref(r_1, r_1.G, F_1), F_2) \equiv$
by Theorem 5
$Ref(r_2, \tau.F_1^Y[Ref(r_1, G, F_1)], F_2) \equiv$
by Lemma 3
$Ref(r_2, \tau.F_1^Y, F_2)[Ref(r_2, Ref(r_1, G, F_1), F_2)] \equiv$
by induction
$Ref(r_2, \tau.F_1^Y, F_2)[Ref(r_1, Ref(r_2, G, F_2), Ref(r_2, F_1, F_2))] \equiv$
by Theorem 4
$Ref(r_1, Ref(r_2, \tau.F_1^Y, F_2)[Ref(r_2, G, F_2)], Ref(r_2, F_1, F_2)) \equiv$
by Lemma 3
$Ref(r_1, Ref(r_2, \tau.F_1^Y[G], F_2), Ref(r_2, F_1, F_2)) \equiv$
by Def. of ParRef
$Ref(r_1, Ref(r_2, ParRef(r_1, r_1.G, F_1), F_2), Ref(r_2, F_1, F_2)) \equiv$
by Lemma 4
$Ref(r_1, ParRef(r_1, Ref(r_2, r_1.G, F_2), Ref(r_2, F_1, F_2)), Ref(r_2, F_1, F_2)) \equiv$
by Def. of Ref
$Ref(r_1, Ref(r_2, r_1.G, F_2), Ref(r_2, F_1, F_2))$

□

Example 6. Let $E \equiv r_1.a.\mathbf{0} + a.r_2.\mathbf{0}$, $F_1 \equiv b.r_2\mathbf{0}$ and $F_2 \equiv c.\mathbf{0}$. We have $Ref(r_2, Ref(r_1, E, F_1), F_2) \equiv Ref(r_2, \tau.b.r_2.a.\mathbf{0} + a.r_2.\mathbf{0}, F_2) \equiv \tau.b.\tau.c.a.\mathbf{0} + a.\tau.c.\mathbf{0}$ $\equiv Ref(r_1, r_1.a.\mathbf{0} + a.\tau.c.\mathbf{0}, b.\tau.c.\mathbf{0}) \equiv Ref(r_1, Ref(r_2, E, F_2), Ref(r_2, F_1, F_2))$.

5 Action Refinement and Security Properties

Let $\mathcal{P}$ be a generic property, i.e., a class of processes. It is immediate to prove that the partial refinement applied to $\mathcal{P}$-contexts preserves $\mathcal{P}$.

Theorem 8. *Let $\mathcal{P}$ be a class of processes. Let E and F be processes. If $E\{r\}$ and F^Y are $\mathcal{P}$-contexts and $E@r$ is a set of $\mathcal{P}$ processes, then $ParRef(r, E, F)$ is a $\mathcal{P}$ process.*

Proof. Since F^Y is a $\mathcal{P}$-context, Y is the only free variable in it, and $E@r$ is a set of $\mathcal{P}$ process, we have that $\{F^Y[T] \mid T \in E@r\}$ is a set of $\mathcal{P}$ processes. Hence, since $E\{r\}$ is a $\mathcal{P}$-context with only the X_{T_i}'s as free variables, we get that $ParRef(r, E, F) \equiv E\{r\}[F^Y[T_1], \ldots, F^Y[T_n]]$ is a $\mathcal{P}$-process. □

Notice that the above theorem cannot be applied when $E@r$ is not a set of processes, i.e., when in $E@r$ there is a term with a free variable (see Example

2). This means that we have no general results when r occurs inside a recursive loop. Moreover, Theorem 8 is limited to the partial refinement, since the fact that $ParRef(r, E, F)$ is in $\mathcal{P}$ does not imply that $ParRef(r, E, F)\{r\}$ is a $\mathcal{P}$-context. In order to obtain a more general result we introduce the following definition.

Definition 9 ($\mathcal{P}$-refinable contexts). *Let $\mathcal{P}$ be a class of processes. A class $\mathcal{C}$ of contexts is said to be a class of* $\mathcal{P}$-refinable contexts *if:*

- *$\mathcal{C}$ is a class of $\mathcal{P}$-contexts;*
- *if $C, D \in \mathcal{C}$, then $C[D] \in \mathcal{C}$;*
- *if $C \in \mathcal{C}$, then $C@r \cup \{C\{r\}\} \subseteq \mathcal{C}$ where r is refinable in C.*

Theorem 9. *Let $\mathcal{P}$ be a class of processes and $\mathcal{C}$ be a class of $\mathcal{P}$-refinable contexts. Let E and F be processes. If $E, F \in \mathcal{C}$, then $Ref(r, E, F)$ is a $\mathcal{P}$ process and it is in $\mathcal{C}$.*

Proof. Since $E \in \mathcal{C}$ we have that $E@r \cup \{E\{r\}\} \subseteq \mathcal{C}$. From the fact that $\mathcal{C}$ is closed under composition we get the thesis. □

In order to apply either Theorem 8 or Theorem 9 we need to be able to characterize classes of $\mathcal{P}$-contexts. This problem has been considered in[4, 5] where some security properties have been considered and classes of contexts with nice properties have been identified. Moreover, in order to apply Theorem 9 we need to characterize classes of $\mathcal{P}$-refinable contexts. In the following subsection we analyze one of the security property considered in[4, 5] , namely P_BNDC, and we show how to apply (and generalize) Theorems 8 and 9.

5.1 Preserving *P_BNDC* under Refinement

Information flow security in a multilevel system aims at guaranteeing that no high level (confidential) information is revealed to users running at low security levels [14, 11, 19, 22, 23], even in the presence of any possible malicious process. *Persistent Bisimulation Non Deducibility on Composition* (P_BNDC, for short) [13], is an information flow security property suitable to analyze processes in completely dynamic hostile environments, i.e., environments which can be dynamically reconfigured at run-time. The notion of P_BNDC is based on the idea of Non-Interference [15] and requires that every state which is reachable by the system still satisfies a basic Non-Interference property. In this paper we present P_BNDC by exploiting an unwinding characterization of it (see [2]).

We first introduce the notion of weak bisimulation on low security actions.

Definition 10 (Weak Bisimulation on Low Actions). *A symmetric binary relation $\mathcal{R} \subseteq \mathcal{E} \times \mathcal{E}$ over processes is a* weak bisimulation on low security actions *if $(E, F) \in \mathcal{R}$ implies, for all $a \in L \cup \{\tau\}$, if $E \xrightarrow{a} E'$, then there exists F' such that $F \overset{\hat{a}}{\Longrightarrow} F'$ and $(E', F') \in \mathcal{R}$.*

Two processes $E, F \in \mathcal{E}$ are weakly bisimilar on low security actions*, denoted by $E \approx_B^L F$, if there exists a weak bisimulation on low security actions $\mathcal{R}$ containing the pair (E, F).*

The definition of P_BNDC in terms of unwinding condition requires that all the high security actions can be locally simulated by a sequence of τ actions.

Definition 11 (P_BNDC). *A process E is P_BNDC if for all E' reachable from E (i.e., $E \xrightarrow{a_1} \ldots \xrightarrow{a_n} E'$) and for all $h \in H$ if $E' \xrightarrow{h} E''$, then $E' \overset{\hat{\tau}}{\Longrightarrow} E'''$ with $E'' \setminus H \approx_B E''' \setminus H$.*

Example 7. Let $l \in L$ and $h \in H$. The process $h.l.h.\mathbf{0} + \tau.l.\mathbf{0}$ is P_BNDC. The process $h.l.\mathbf{0}$ is not P_BNDC.

The decidability of P_BNDC has been proved in [13] and an efficient (polynomial) algorithm has been presented in [12]. In [2], a proof system which allows us to incrementally build P_BNDC processes has been obtained by exploiting the unwinding characterization of P_BNDC.

The compositional properties of P_BNDC allows us to syntactically characterize two classes of P_BNDC-contexts.

Definition 12 (The classes $\mathcal{C}'$ and $\mathcal{C}$).

- *$\mathcal{C}'$ is the class of contexts containing: the process $\mathbf{0}$; Z, where Z is a variable; $\sum_{i\in I} l_i.C_i + \sum_{j\in J}(h_j.D_j + \tau.D_j)$, with $l_i \in L \cup \{\tau\}$, $h_j \in H$, $C_i, D_j \in \mathcal{C}'$; $C \setminus v$, $C[f]$, and $recZ.C$, with $C \in \mathcal{C}'$.*
- *$\mathcal{C}$ is the class of contexts containing: all the P_BNDC processes; Z, where Z is a variable; $\sum_{i\in I} l_i.C_i + \sum_{j\in J}(h_j.D_j + \tau.D_j)$, with $l_i \in L \cup \{\tau\}$, $h_j \in H$, $C_i, D_j \in \mathcal{C}$; $C \setminus v$, $C[f]$, and $recZ.C$, with $C \in \mathcal{C}$.*

Lemma 5. *$\mathcal{C}'$ and $\mathcal{C}$ are classes of P_BNDC-contexts.*

Proof. We prove that $\mathcal{C}'$ is a class of P_BNDC-contexts.

The only interesting case is the case of recursion.

We have to prove that for all $C \in \mathcal{C}'$, for all $E_1, \ldots, E_n \in P\_BNDC$, $C[E_1, \ldots, E_n] \in P\_BNDC$. Since if $E \approx_B F$, then $E \in P\_BNDC$ if and only if $F \in P\_BNDC$, we can prove that a process equivalent to $C[E_1, \ldots, E_n]$ is P_BNDC.

Since the parallel composition is not used in $\mathcal{C}'$, we can push the restrictions and the renamings inside until we reach a variable. In fact $(E_1 + E_2) \setminus v \approx_B E_1 \setminus v + E_2 \setminus v$ and $(recZ.E) \setminus v \approx_B recZ.(E \setminus v)$ (see Lemma 7 in [6]).

Moreover, by applying some transformations on the renamings we can permute them with all the restrictions, in order to push the renamings inside. Hence, we can transform C into D, where D is built using the following productions

$$\begin{array}{l} B ::= X \mid B \setminus v \mid B[f] \\ D ::= B \mid \sum_{i\in I} l_i.D_i + \sum_{j\in J}(h_j.D_j + \tau.D_j) \end{array}$$

with $l_i \in L \cup \{\tau\}$ and $h_j \in H$. The context D is strongly bisimilar to C, hence it is sufficient to prove that $D[E_1, \ldots, E_n]$ is P_BNDC.

The process $D[E_1, \ldots, E_n]$ can be seen as $G[F_1, \ldots, F_n]$, where G is built only using variables and sums of the form $\sum_{i\in I} l_i.G_i + \sum_{j\in J}(h_j.G_j + \tau.G_j)$ and F_i is of the form $E_i[f] \setminus v$ for some f and v. Since the E_i's are P_BNDC, the F_i's are P_BNDC. Since G is built using only variables and sums we have that if $G[F_1, \ldots, F_n]$ reaches a process G', then two cases are possible:

- G reaches $\overline{G}$ and $G' \equiv \overline{G}[F_1, \ldots, F_n]$;
- there exists i such that F_i reaches G'.

In the second case, since F_i is P_BNDC we immediately get that if $G' \xrightarrow{h} G''$, then $G' \overset{\hat{\tau}}{\Rightarrow} G'''$, with $G'' \approx_B^L G'''$. In the first case if $\overline{G}[F_1, \ldots, F_n] \xrightarrow{h} G''$, two cases are possible:

- $\overline{G} \xrightarrow{h} \tilde{G}$ and $G'' \equiv \tilde{G}[F_1, \ldots, F_n]$;
- there exists i such that $F_i \xrightarrow{h} G''$.

Again in the second case we get the thesis, since F_i is P_BNDC. In the firs case we get the thesis since by construction $\overline{G} \xrightarrow{\tau} \tilde{G}$, and $G'' \approx_B^L \tilde{G}[F_1, \ldots, F_n]$.

We prove that $\mathcal{C}$ is a class of P_BNDC-contexts.

The thesis immediately follows from the fact that each context in $\mathcal{C}$ is nothing but a context of $\mathcal{C}'$ in which some variables have already been replaced by P_BNDC processes. □

Both $\mathcal{C}'$ and $\mathcal{C}$ are decidable classes of contexts. In particular, the decidability of $\mathcal{C}$ is a consequence of the fact that P_BNDC is decidable (see [13, 12]).

Since the class $\mathcal{C}$ is closed under composition of contexts we obtain the following result. Notice that it is not a consequence of Theorem 8, since $E@r$ can now contain terms with free variables.

Corollary 2. *Let E and F be processes. If $E@r \cup \{E\{r\}, F^Y\} \subseteq \mathcal{C}$, then*

$$ParRef(r, E, F) \in P\_BNDC.$$

Proof. Since $E\{r\}$, $E@r$, and F^Y are in $\mathcal{C}$ we have that $ParRef(r, E, F)$ is in $\mathcal{C}$. Hence by Theorem 5, $ParRef(r, E, F)$ is a P_BNDC-context. Since E is a process $ParRef(r, E, F)$ is a process, i.e., it is a P_BNDC process. □

The following lemma allows us to instantiate Theorem 9 in the case of P_BNDC.

Lemma 6. *$\mathcal{C}'$ is a class of P_BNDC-refinable contexts.*

Proof. By Lemma 5 we have that $\mathcal{C}'$ is a class of P_BNDC-contexts.

It is immediate to prove that $\mathcal{C}'$ is closed under composition.

By definition of $\mathcal{C}'$ we have that if $C \in \mathcal{C}$, then $C@r \subseteq \mathcal{C}'$.

The fact that if $C \in \mathcal{C}$, then $C\{r\} \in \mathcal{C}$ is a consequence of the fact that we replace r with $\tau.Y$, hence all the sums remain guarded. □

Corollary 3. *Let E, F be processes. If $E, F^Y \in \mathcal{C}'$, then $Ref(r, E, F)$ is a P_BNDC process and it is in $\mathcal{C}'$.*

Proof. This is a consequence of Theorem 9 and Lemma 6. □

We conclude this section by reporting an example adapted from [18].

Example 8. Let us consider a distributed data base which can take two values and which can be both queried and updated. In particular, the high level user can query it (qry_1, qry_2), while the low level user can only update it (upd_1, upd_2). Hence $qry_1, qry_2 \in H$ and $upd_1, upd_2 \in L$. We can model the data base with the following SPA process

$$\begin{aligned} E \equiv recZ.(& qry_1.Z + upd_1.Z + \tau.Z + \\ & upd_2.recW.(qry_2.W + upd_2.W + \tau.W + upd_1.Z)). \end{aligned}$$

The process E is in $\mathcal{C}'$. We can now refine the update actions by requiring that each update is requested and confirmed, i.e., we refine upd_1 using $F_1 \equiv req_1.cnf_1.\mathbf{0}$ and upd_2 using $F_2 \equiv req_2.cnf_2.\mathbf{0}$, where $req_1, cnf_1, req_2, cnf_2$ are low security level actions. We obtain the process $Ref(r_2, Ref(r_1, E, F_1), F_2)$ is

$$\begin{aligned} recZ.(& qry_1.Z + req_1.cnf_1.Z + \tau.Z + \\ & req_2.cnf_2.recW.(qry_2.W + req_2.cnf_2.W + \tau.W + req_1.cnf_1.Z)). \end{aligned}$$

Since F_1^Y and F_2^Y are in $\mathcal{C}'$, by Corollary 3, $Ref(r_2, Ref(r_1, E, F_1), F_2)$ is a P_BNDC process.

6 Conclusions and Related Works

In this paper we study the relations between action refinement, compositionality, and information flow security within the Security Process Algebra (SPA).

We formalize the notion of action refinement in terms of context composition. This approach allows us to exploit properties of open terms to individuate conditions under which security properties can be preserved under action refinement. In particular, we consider the security property P_BNDC and show how it can be preserved under action refinement.

Action refinement has been extensively studied in the literature. There are essentially two interpretations of action refinement: *semantic* and *syntactic* (see [16]). In the semantic interpretation an explicit refinement operator, written $E[r \to F]$, is introduced in the semantic domain used to interpret the terms of the algebra. The semantics of $E[r \to F]$ models the fact that r is an action of E to be refined by process F. In the syntactic approach, the same situation is modelled by syntactically replacing r by F in E. The replacement can be *static*, i.e., before execution, or *dynamic*, i.e., r is replaced as soon as it occurs while executing E. In order to correctly formalize the replacement, the process algebra is usually equipped with an operation of sequential composition (rather than the more standard action prefix), as, e.g., in ACP, since otherwise it would not be closed under the necessary syntactic substitution. Our approach to action refinement follows the static, syntactic interpretation. However, the use of context composition to realize the refinement allows us to keep the original SPA language without introducing a sequential composition operator for processes.

Action refinement is also classified as *atomic* or *non-atomic*. Atomic refinement is based on the view that actions are atomic and their refinements should in some sense preserve this atomicity (see, e.g.,[9, 7, 17]). On the other hand, non-atomic refinement takes the view that atomicity is always relative to the current level of abstraction and may, in a sense, be destroyed by the refinement (see, e.g., [1, 10, 24]). In this paper we follow the *non-atomic* approach. Actually, this approach is on the whole more popular then the former.

In the literature the term *refinement* is also used to indicate any transformation of a system that can be justified because the transformed system implements the original one on the *same* abstraction level, by being more nearly executable, for instance more deterministic. The implementation relation is expressed in terms of pre-orders such as trace inclusion or various kinds of simulation. Many papers in this tradition can be found in [8]. The relations between this form of refinement and information flow security have been studied in [3].

References

1. L. Aceto and M. Hennessy. Adding action refinement to a finite process algebra. *Information and Computation*, 115(2):179–247, 1994.
2. A. Bossi, R. Focardi, C. Piazza, and S. Rossi. A Proof System for Information Flow Security. In M. Leuschel, editor, *Logic Based Program Development and Transformation*, volume 2664 of *LNCS*, pages 199–218. Springer-Verlag, 2003.
3. A. Bossi, R. Focardi, C. Piazza, and S. Rossi. Refinement Operators and Information Flow Security. In *Proc. of the International Conference on Software Engineering and Formal Methods (SEFM'03)*. IEEE Comp. Soc. Press, 2003. To appear.
4. A. Bossi, D. Macedonio, C. Piazza, and S. Rossi. Secure Contexts for Confidential Data. In *Proc. of the 16th IEEE Computer Security Foundations Workshop*, pages 14–28. IEEE Computer Society Press, 2003.
5. A. Bossi, D. Macedonio, C. Piazza, and S. Rossi. Information Flow Security and Recursive Systems. In *Italian Conference on Theoretical Computer Science*, LNCS. Springer-Verlag, 2004. To appear.
6. A. Bossi, D. Macedonio, C. Piazza, and S. Rosssi. Secure Contexts for Information Flow Security. Technical Report CS-2002-18, Dipartimento di Informatica, Università Ca' Foscari di Venezia, Italy, 2002.
7. G. Boudol. Atomic actions. *Bulletin of the EATCS*, 38:136–144, 1989.
8. J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors. *Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness, REX Workshop, Mook, The Netherlands, May 29 - June 2, 1989, Proceedings*, volume 430 of *Lecture Notes in Computer Science*. Springer, 1990.
9. J. W. de Bakker and E. P. de Vink. Bisimulation semantics for concurrency with atomicity and action refinement. *Fundamenta Informaticae*, 20(1/2/3):3–34, 1994.
10. P. Degano and R. Gorrieri. A causal operational semantics of action refinement. *Information and Computation*, 122(1):97–119, 1995.
11. R. Focardi and R. Gorrieri. Classification of Security Properties (Part I: Information Flow). In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *LNCS*. Springer-Verlag, 2001.
12. R. Focardi, C. Piazza, and S. Rossi. Proof Methods for Bisimulation based Information Flow Security. In A. Cortesi, editor, *Proc. of Int. Workshop on Verification, Model Checking and Abstract Interpretation*, volume 2294 of *LNCS*, pages 16–31. Springer-Verlag, 2002.

13. R. Focardi and S. Rossi. Information Flow Security in Dynamic Contexts. In *Proc. of the IEEE Computer Security Foundations Workshop (CSFW'02)*, pages 307–319. IEEE Comp. Soc. Press, 2002.
14. S. N. Foley. A Universal Theory of Information Flow. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 116–122. IEEE Comp. Soc. Press, 1987.
15. J. A. Goguen and J. Meseguer. Security Policies and Security Models. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 11–20. IEEE Comp. Soc. Press, 1982.
16. U. Goltz, R. Gorrieri, and A. Rensink. Comparing syntactic and semantic action refinement. *Information and Computation*, 125(2):118–143, 1996.
17. R. Gorrieri, S. Marchetti, and U. Montanari. A^{2}CCS: Atomic actions for CCS. *Theoretical Computer Science*, 72(2-3):203–223, 1990.
18. R. Gorrieri and A. Rensink. Action Refinement. Technical Report UBLCS-99-09, University of Bologna (Italy), 1999.
19. J. McLean. Security Models and Information Flow. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 180–187. IEEE Comp. Soc. Press, 1990.
20. R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.
21. M. Nielsen, U. Engberg, and K. S. Larsen. Fully Abstract Models for a Process Language with Refinement. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Proc. of the Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop*, volume 354 of *LNCS*, pages 523–548. Springer-Verlag, 1989.
22. C. O'Halloran. A Calculus of Information Flow. In *Proc. of the European Symposium on Research in Security and Privacy*, pages 180–187. AFCET, 1990.
23. G. Smith and D. M. Volpano. Secure Information Flow in a Multi-threaded Imperative Language. In *Proc. of ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'98)*, pages 355–364. ACM Press, 1998.
24. R. J. van Glabbeek and U. Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Informatica*, 37(4/5):229–327, 2001.
25. N. Wirth. Program development by stepwise refinement. *Communications of the ACM*, 14(4):221–227, 1971.