RWTH Aachen

Department of Computer Science
Technical Report

A Probabilistic Justification of the Combining Calculus under the Uniform Scheduler Assumption

Tina Kraußer and Heiko Mantel and Henning Sudbrock

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2007-09

RWTH Aachen · Department of Computer Science · May 2007

A Probabilistic Justification of the Combining Calculus under the Uniform Scheduler Assumption

Tina Kraußer and Heiko Mantel and Henning Sudbrock*

Security Engineering Group
RWTH Aachen, Germany
Email: {krausser, mantel, sudbrock}@cs.rwth-aachen.de

Abstract. The *combining calculus* [MSK07] provides a framework for analyzing the information flow of multi-threaded programs. The calculus incorporates so called plug-in rules for integrating several previously existing analysis techniques. By applying a plug-in rule to a subprogram, one decides to analyze this subprogram with the given analysis technique, and not with the rules of the combining calculus. The novelty of the combining calculus was that one can analyze the information flow security of a given program by using multiple analysis techniques in combination. It was demonstrated that this flexibility leads to a more precise analysis, allowing one to successfully verify the security of some programs that cannot be verified with any of the existing analysis techniques in isolation.
In [MSK07], the soundness of the combining calculus is proved for a possibilistic characterization of information flow security. This characterization assumes a purely nondeterministic scheduling of concurrent threads. In this report, we demonstrate that the combining calculus is also sound for a probabilistic characterization of security that assumes a uniform scheduler. This result further increases the confidence in the combining calculus as a reliable and flexible tool for formally analyzing the information flow security of multi-threaded programs.

1 Introduction

Before giving a program access to secret data, one wants to know whether there is any danger that the program might leak secrets to untrusted sinks or, in other words, whether the program has secure information flow. The two main research problems in information flow security are, firstly, finding formal characterizations that faithfully capture the security requirements and, secondly, developing analysis techniques based on these characterizations. Information flow security has been a focal research topic for more than 30 years. Nevertheless, the problem to secure the flow of information in programs is far from being solved [SM03].

In [VSI96], Volpano, Smith and Irvine proposed a type-based information flow analysis for imperative programs. This article is widely acknowledged as a milestone in language-based security because it not only presented an analysis technique that can be efficiently automated, but also a proof that this analysis technique is sound with respect to a formal definition of information flow security.

A soundness result ensures that all programs that pass the analysis are, indeed, secure. However, this does not imply that all programs that are secure also pass the analysis. In fact, many type-based analysis techniques are imprecise,

* This work was funded in part by the Information Society Technologies program of the European Commission, Future and Emerging Technologies under the IST-2005-015905 MOBIUS project and by the German Research Association (DFG) in the Computer Science Action Program. This paper reflects only the author's views and the Community is not liable for any use that may be made of the information contained therein.

which means that some secure programs are rejected by these techniques. Although it would be desirable to have analysis techniques that are precise, one is often willing to sacrifice precision for making the analysis more efficient.

Our main motivation for the combining calculus [MSK07] was the need for more precise techniques to analyze the information flow security of multi-threaded programs. The plug-in rules of the combining calculus allow one to exploit previously existing analysis techniques and to analyze different subprograms by applying different analysis techniques. The combination of combining calculus rules with existing analysis techniques, indeed, improves the precision of the analysis, as demonstrated in [MSK07]. The soundness result for the combining calculus assumes a possibilistic characterization of information flow security. While this result creates some confidence in the calculus, it is not yet fully satisfactory, as already pointed out in [MSK07]. The possibilistic characterization assumes that threats are selected in a particular way, namely by a purely nondeterministic scheduler. Although a particular scheduler is also assumed in other soundness results (see, e.g., [SV98,VS98]), it would be even more desirable to have a soundness result that is scheduler independent (like, e.g., in [SS00]).

The novel contributions of this report are a probabilistic security characterization and a proof that the combining calculus is sound with respect to this characterization for a uniform scheduler. The soundness result further increases the confidence in the combining calculus. The result also constitutes another step towards a scheduler-independent soundness result for the combining calculus.

Overview. In Section 2, a simple multi-threaded programming language is introduced. The probabilistic characterization of information flow security is proposed in Section 3. In Section 4, compositionality results are derived and the general rules of the combining calculus are proved sound with respect to the probabilistic security characterization by exploiting these compositionality results. The various plug-in rules are proved to be sound in Sections 5 and 6. In Section 7, we elaborate the relationship between our probabilistic security characterization and the possibilistic characterization from [MSK07] in more detail. The report concludes with a discussion of related work and an outlook in Section 8.

2 Multi-Threaded While-Language

We use the multi-threaded while language from [SS00], extended by a primitive for barrier synchronization. The set of commands Com is defined by:

$$\begin{aligned} C ::= \mathsf{skip} \mid Id{:=}Exp \mid C_1; C_2 \mid \mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \mathsf{fi} \mid \\ \mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od} \mid \mathsf{fork}(C, \langle C_1 \mid .. \mid C_n \rangle) \mid \mathsf{sync} \end{aligned}$$

The execution of programs is modeled by a sequence of transitions between configurations $\langle\!| V, s |\!\rangle$ where $V = \langle C_1 \mid .. \mid C_n \rangle$ is a thread pool and s is a memory state. We denote the set of memory states by S. The deterministic semantics is given in Figure 1, where $\langle\!| V, s |\!\rangle \twoheadrightarrow \langle\!| V', s' |\!\rangle$ means that $\langle\!| V, s |\!\rangle$ is deterministically evaluating to $\langle\!| V', s' |\!\rangle$ in a single execution step.

The evaluation of a thread pool bases on the underlying scheduler. During the evaluation of a multi-threaded program we track the decisions of the scheduler. Single probabilistic transitions are denoted by $\langle\!| V, s |\!\rangle \xrightarrow{p,i} \langle\!| V', s' |\!\rangle$ where p is

$$\frac{}{\langle\!|\mathsf{skip}, s|\!\rangle \rightharpoonup \langle\!|\langle\rangle, s|\!\rangle} \qquad \frac{\langle\!|Exp, s|\!\rangle \downarrow n}{\langle\!|Id{:=}Exp, s|\!\rangle \rightharpoonup \langle\!|\langle\rangle, [Id = n]s|\!\rangle}$$

$$\frac{\langle\!|C_1, s|\!\rangle \rightharpoonup \langle\!|\langle\rangle, t|\!\rangle}{\langle\!|C_1; C_2, s|\!\rangle \rightharpoonup \langle\!|C_2, t|\!\rangle} \qquad \frac{\langle\!|C_1, s|\!\rangle \rightharpoonup \langle\!|\langle C_1'\rangle V, t|\!\rangle}{\langle\!|C_1; C_2, s|\!\rangle \rightharpoonup \langle\!|\langle C_1'; C_2\rangle V, t|\!\rangle} \qquad \frac{}{\langle\!|\mathsf{fork}(C, V), s|\!\rangle \rightharpoonup \langle\!|\langle C\rangle V, s|\!\rangle}$$

$$\frac{\langle\!|B, s|\!\rangle \downarrow \mathsf{True}}{\langle\!|\mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \mathsf{fi}, s|\!\rangle \rightharpoonup \langle\!|C_1, s|\!\rangle} \qquad \frac{\langle\!|B, s|\!\rangle \downarrow \mathsf{False}}{\langle\!|\mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \mathsf{fi}, s|\!\rangle \rightharpoonup \langle\!|C_2, s|\!\rangle}$$

$$\frac{\langle\!|B, s|\!\rangle \downarrow \mathsf{True}}{\langle\!|\mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od}, s|\!\rangle \rightharpoonup \langle\!|C; \mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od}, s|\!\rangle} \qquad \frac{\langle\!|B, s|\!\rangle \downarrow \mathsf{False}}{\langle\!|\mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od}, s|\!\rangle \rightharpoonup \langle\!|\langle\rangle, s|\!\rangle}$$

Fig. 1. Small-step deterministic semantics

the probability for this transition and i is the number of the thread that was chosen by the scheduler. For the execution of the sync statement we need that all threads do the step together. We mark this by the special symbol $\star$ for the thread choice. Moreover, threads that are blocked due to a sync statement increase the probability for non-blocked threads to be chosen next. This fact is taken into account by the usage of the function *blkd*:

Definition 1. *Let $V = \langle C_1 \mid C_2 \mid .. \mid C_n\rangle$ be a thread pool. The function $blkd(V)$ returns the number of currently blocked threads in $V = \langle C_1 \mid C_2 \mid .. \mid C_n\rangle$. Formally we have:*

$$blkd(V) = \begin{cases} 0 & \textit{if } V = \langle\rangle, \\ blkd(\langle C_2 \mid .. \mid C_n\rangle) + 1 & \textit{if } C_1 = \mathsf{sync}, \\ blkd(\langle C_2 \mid .. \mid C_n\rangle) + 1 & \textit{if } C_1 = \mathsf{sync}; C_1', \\ blkd(\langle C_2 \mid .. \mid C_n\rangle) & \textit{otherwise.} \end{cases}$$

A single decision made by a scheduler along a transition is in the following called *decision step*:

Definition 2. *A **decision step** is a triple $(\langle\!|V, s|\!\rangle, p, i)$ where p is a probability, and $i \in \mathbb{N} \cup \{\circ, \star\}$.*

Figure 2 shows the semantics of thread choices under the uniform scheduler where each transition is labeled with the decision that leads from the left to the right configuration. Notice that the condition $blkd(\langle C_1 \mid \dots \mid C_n\rangle) < n$ in the first rule is already implied by the fact that C_i can make a deterministic transition, we list it only for clarity. We write $\langle V, s\rangle \rightarrow \langle W, t\rangle$ if there exist p, i such that $\langle V, s\rangle \xrightarrow{p,i} \langle W, t\rangle$.

Figure 3 illustrates the transition rules that reason about a sequence of scheduler choices by using judgments of the form $\langle\!|V, s|\!\rangle \overset{p}{\twoheadrightarrow}_l \langle\!|V', s'|\!\rangle$ where the sequence of scheduler decisions is formulated as the *decision path* l and p is the *probability of* l. The infix operator $++$ denotes concatenation of lists of decision steps.

Definition 3. *Let l be a list of decision steps, and $\langle\!|V, s|\!\rangle$ as well as $\langle\!|V', s'|\!\rangle$ configurations. We call l a **decision path from** $\langle\!|V, s|\!\rangle$ **to** $\langle\!|V', s'|\!\rangle$ if the following holds:*

$$\exists p. \langle\!|V, s|\!\rangle \overset{p}{\twoheadrightarrow}_l \langle\!|V', s'|\!\rangle.$$

$$\frac{\langle\!| C_i, s |\!\rangle \twoheadrightarrow \langle\!| V_i', s_i' |\!\rangle \qquad blkd(\langle C_1 \mid \ldots \mid C_n\rangle) < n \qquad p = 1/(n - blkd(\langle C_1 \mid .. \mid C_n\rangle))}{\langle\!| \langle C_1 \mid .. \mid C_i \mid .. \mid C_n\rangle, s |\!\rangle \xrightarrow{p,i} \langle\!| \langle C_1 \mid .. \mid V_i' \mid .. \mid C_n\rangle, s_i' |\!\rangle}$$

$$\frac{\forall i.((C_i = \mathsf{sync} \wedge C_i' = \langle\rangle) \vee (C_i = \mathsf{sync}; C_i'))}{\langle\!| \langle C_1 \mid .. \mid C_n\rangle, s |\!\rangle \xrightarrow{1,\star} \langle\!| \langle C_1' \mid .. \mid C_n'\rangle, s |\!\rangle}$$

Fig. 2. Semantics of thread choices

$$\frac{\begin{array}{c}\langle\!| V, s |\!\rangle \xrightarrow{p,i} \langle\!| V', s' |\!\rangle \\ l = [(\langle\!| V, s |\!\rangle,\ 1,\ \circ), (\langle\!| V', s' |\!\rangle,\ p,\ i)]\end{array}}{\langle\!| V, s |\!\rangle \overset{p}{\twoheadrightarrow}_l \langle\!| V', s' |\!\rangle}$$

$$\frac{\langle\!| V, s |\!\rangle \overset{p_1}{\twoheadrightarrow}_l \langle\!| V', s' |\!\rangle \quad \langle\!| V', s' |\!\rangle \xrightarrow{p_2,i} \langle\!| V'', s'' |\!\rangle \quad l' = l + +[(\langle\!| V'', s'' |\!\rangle, p_2, i)]}{\langle\!| V, s |\!\rangle \overset{p_1 * p_2}{\twoheadrightarrow}_{l'} \langle\!| V'', s'' |\!\rangle}$$

Fig. 3. Probabilistic semantics

We call l a ***decision path*** *if there exist V, s such that $l = [(\langle\!| V, s |\!\rangle, 1, \circ)]$, or if there exist V, V', s, s' such that l is a decision path from $\langle\!| V, s |\!\rangle$ to $\langle\!| V', s' |\!\rangle$. We denote the set of all decision paths by DP.*

Let $l \in DP$ be a decision path with

$$l = [(\langle\!| V_1, s_1 |\!\rangle, p_1, i_1), (\langle\!| V_2, s_2 |\!\rangle, p_2, i_2), \ldots, (\langle\!| V_n, s_n |\!\rangle, p_n, i_n)].$$

We define the following functions on decision paths:

length of a decision path *The number n of decision steps in l is called length of l (denoted by $length(l)$).*

tail of a decision path *The list $[(\langle\!| V_2, s_2 |\!\rangle, p_2, i_2), \ldots, (\langle\!| V_n, s_n |\!\rangle, p_n, i_n)]$ is the tail of l (denoted by l^-).*

concatenation *Let*

$$h = [(\langle\!| V_{n+1}, s_{n+1} |\!\rangle, p_{n+1}, i_{n+1}), \ldots, (\langle\!| V_{n+k}, s_{n+k} |\!\rangle, p_{n+k}, i_{n+k})]$$

be a list of decision steps. The concatenation of l and h (denoted by $l + +h$) is defined by:

$$\begin{aligned} l + +h = [\ & (\langle\!| V_1, s_1 |\!\rangle, p_1, i_1), \ldots, (\langle\!| V_n, s_n |\!\rangle, p_n, i_n), \\ & (\langle\!| V_{n+1}, s_{n+1} |\!\rangle, p_{n+1}, i_{n+1}), \ldots, (\langle\!| V_{n+k}, s_{n+k} |\!\rangle, p_{n+k}, i_{n+k})]. \end{aligned}$$

prefixing *A decision path l' with*

$$\exists h \in DP.\ length(h) > 1 \wedge l = l' + +h^-$$

is called proper prefix of l (denoted by $l' \sqsubset l$). Moreover, l'' is a prefix of l if l'' is either a proper prefix of l or $l'' = l$.

We call a decision path $l = [(\langle\!| V_1, s_1 |\!\rangle, p_1, i_1), \ldots, (\langle\!| V_n, s_n |\!\rangle, p_n, i_n)]$ ***non final*** *if V_n is not the empty program.*

Definition 4. *Let l be a decision path and p the probability such that*

$$l = [(\langle\!| V, s |\!\rangle, p, \circ)] \wedge p = 1 \quad or \quad \langle\!| V, s |\!\rangle \xrightarrow{p}\!\!\!\!\rightarrow_l \langle\!| V', s' |\!\rangle.$$

The probability p is the ***probability of*** *l (denoted by $prob_l$).*

Lemma 1. *For all decision paths l the following holds:*

$$0 < prob_l \leq 1$$

Proof. Since a decision path of length one cannot be generated by the transition rules we have from Definition 3 that $l = [(\langle\!| V, s |\!\rangle, 1, \circ)]$. From this we have

$$\forall l.length(l) = 1 \Longrightarrow 0 < prob_l = p = 1. \tag{1}$$

The first rule of Figure 2 requires that $p = 1/(n - blkd(\langle C_1 \mid .. \mid C_n \rangle))$. Since $blkd(\langle C_1 \mid .. \mid C_n \rangle)$ has to be smaller than n we know that the denominator is not equal to 0. Hence,

$$0 < p = 1/(n - blkd(\langle C_1 \mid .. \mid C_n \rangle)) \leq 1$$

holds. The second rule requires $p = 1$.

The first rule of Figure 3 produces paths of the length 2 and does not change the probability. Hence from the knowledge that rules 1 and 2 require probabilities p with $0 < p \leq 1$ we have by rule 1 of Figure 3 that

$$\forall l'.length(l') = 2 \Longrightarrow (0 < prob_{l'} \leq 1). \tag{2}$$

The second rule of Figure 3 multiplies the probability values p_1 for a shorter path l and p_2 for a single step. Assume $0 < p_1 = prob_l \leq 1$ holds for all decision paths l with length(l)=n. Since p_1 as well as p_2 are always greater than 0 and less or equal than 1 we know that

$$0 < p_1 * p_2 = prob_{l'} \leq 1$$

holds and, hence,

$$\forall l'.(\exists l, V', s', p', i'.l' = l + + \left[(\langle\!| V', s' |\!\rangle, p', i')\right]) \Longrightarrow (0 < prob_{l'} \leq 1). \tag{3}$$

From (1), (2) and (3) we can inductively show that the lemma holds. □

The sum of probabilities for all possibilities for a single evaluation step of a program is 1 unless the program is the empty program. Hence, we know that the evaluation along a given path l has the same probability as the sum over all probabilities of paths that incorporate one further decision step than l.

Lemma 2. *Let l be a non final decision path from the configuration $\langle\!| V, s |\!\rangle$ to the configuration $\langle\!| V', s' |\!\rangle$. Moreover, let Υ denote the set of all decision paths h with*

$$\exists i, V_i'', s_i'', p_i, p. \quad (h = l + +[(\langle\!| V_i'', s_i'' |\!\rangle, p_i, i)] \quad \wedge \quad \langle\!| V, s |\!\rangle \xrightarrow{p}\!\!\!\!\rightarrow_h \langle\!| V_i'', s_i'' |\!\rangle).$$

Under those conditions the following holds:

$$prob_l = \sum_{h \in \Upsilon} prob_h.$$

Proof. Assume length(l)=1. Since we cannot generate a path of length 1 by the evaluation of the transition rules we know by Definition 4 that $prob_l = 1$. Moreover, we know that for all h generated from l length(h) $= 2$ holds. Paths of length 2 can either be generated by the combination of the first rule of Figure 2 and the first rule of Figure 3 or by a combination of the second rule of Figure 2 and the first rule of Figure 3. Assume $V' = \langle C'_1 \mid .. \mid C'_n \rangle$ and $blkd(V') < n$. By the combination of rule one of Figure 2 and rule one of Figure 3 and from the fact that l is non final we know that we can generate $(n - blkd(V'))$ different paths h' and for each of those paths the following transition is the only valid:

$$\langle\!| V, s |\!\rangle \xrightarrow{1/(n-blkd(V'))}\!\!\!\!\!\!\rightarrow_{h'} \langle\!| V''_i, s''_i |\!\rangle.$$

Hence, we have for each of the $(n - blkd(V'))$ paths h with length(h) $= 2$ that the following holds:

$$prob_h = 1/(n - blkd(V'))$$

The sum over $(n - blkd(V'))$ times $1/(n - blkd(V'))$ is 1. On the other hand if $blkd(V')$ equals n then we have by the second rule of Figure 2 and the first rule of Figure 3 and again from the fact that l is non final that $p = prob_h = 1$.

Now assume length(l)>1. The only rule that allows us to generate paths h with length(h)>2 is the second rule of Figure 3. Let n be the number of threads in V'. We can use the same argumentation as before to show that

$$\sum_{i=1}^{n} \{\ p \mid \langle\!| V', s' |\!\rangle \xrightarrow{p,i} \langle\!| V'', s'' |\!\rangle \} = 1$$

and, hence,

$$\begin{aligned} \sum_{h \in \Upsilon} prob_h &= prob_l * \sum_{i=1}^{n} \{\ p \mid \langle\!| V', s' |\!\rangle \xrightarrow{p,i} \langle\!| V'', s'' |\!\rangle \} \\ &= prob_l * 1 \\ &= prob_l \end{aligned}$$

holds. □

Sets of decision paths that give us information about the distribution of probabilities are summarized as *probability trees.*

Definition 5. *The* ***set of probability trees*** *for a configuration* $\langle\!| V, s |\!\rangle$ *(denoted by* $Tr_{\langle\!| V,s |\!\rangle}$*) is a set of sets of decision paths that is inductively defined by:*

$$\{[(\langle\!| V, s |\!\rangle, 1, \circ)]\} \in Tr_{\langle\!| V,s |\!\rangle}$$

and

$$\forall tr \in Tr_{\langle\!| V,s |\!\rangle}, l \in DP.\ l \text{ is a nonfinal decision path} \wedge l \in tr \Longrightarrow$$

$$\Big((tr \setminus l) \cup \{h \mid \quad \exists i, V''_i, s''_i, p_i, p. \\ (h = l ++ [(\langle\!| V''_i, s''_i |\!\rangle, p_i, i)] \wedge \langle\!| V, s |\!\rangle \xrightarrow{p}\!\!\!\!\rightarrow_h \langle\!| V''_i, s''_i |\!\rangle)\}) \in Tr_{\langle\!| V,s |\!\rangle}\Big).$$

In the following we will show that probability trees form a probability space in a natural way.

Definition 6. *Let Ω be a set and $\mathfrak{A}$ be a σ-algebra over Ω (i.e., a non-empty subset of $\mathbb{P}(\Omega)$ that is closed under complements and countable unions).*[1] *Let $P : \mathfrak{A} \to \mathbb{R}$ be a* probability measure*, i.e. a function satisfying*

1. $P(A) \geq 0$ *for all* $A \in \mathfrak{A}$*,*
2. $P(\Omega) = 1$*,*
3. *and* $P\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} P(A_i)$ *for pairwise disjoint sets* $A_i \in \mathfrak{A}$*.*

Then the triple $(\Omega, \mathfrak{A}, P)$ is called probability space*. The elements of $\mathfrak{A}$ are called* events*.*

In our case, we assign a probability to each occurrence of a decision path.

Proposition 1. *Let Ω be a probability tree and let P be the function from $\mathbb{P}(\Omega)$ to the real numbers given by*

$$P(A) = \sum_{l \in A} prob_l.$$

Then the triple $(\Omega, \mathbb{P}(\Omega), P)$ is a probability space.

Proof. We prove the first two properties from Definition 6, the third property is trivially satisfied.
$\mathbf{P(\Omega) = 1}$ We do a proof by induction over the definition of probability trees. Assume $\Omega_0 = \{[(\langle\!|V, s|\!\rangle, p, \circ)]\}$. Hence, the number of steps that are needed to generate Ω_0 is 1. Only the first case of Definition 5 argues about probability trees containing a single decision path with length 1. Hence, we know that p=1 holds and that

$$\begin{aligned} P(\Omega_0) &= \sum prob_{[(\langle\!|V,s|\!\rangle,p,\circ)]} \\ &= 1 \end{aligned}$$

holds as well.

By Lemma 2 we know that if we remove a non-final decision path l from a tree and add the set Υ of all paths h to the tree such that

$$\forall h \in \Upsilon . \exists i, V_i'', s_i'', p_i, p. \quad h = l + +[(\langle\!|V_i'', s_i''|\!\rangle, p_i, i)] \ \wedge \langle\!|V, s|\!\rangle \xrightarrow{p}\!\!\!\!\rightarrow_h \langle\!|V_i'', s_i''|\!\rangle$$

then

$$prob_l = \sum_{h \in \Upsilon} prob_h.$$

Hence, for a probability tree Ω' that is generated from a probability tree Ω by using the second construction rule for trees once we have

$$\sum_{l \in \Omega} prob_l = \sum_{l' \in \Omega'} prob_{l'}.$$

[1] Here $\mathbb{P}(X)$ denotes the powerset of X.

Since we have already shown that for the base case

$$\sum_{l_0 \in \Omega_0} prob_{l_0} = 1$$

holds, we obtain by induction that

$$P(\Omega) = \sum_{l \in \Omega} prob_l = 1$$

holds for all probability trees Ω.

$\mathbf{0 \leq P(A) \leq 1}$ Assume $A = \Omega$ then we have already shown that $P(A) = 1$. Otherwise A is a proper subset of Ω. From Lemma 1 we have for all decision paths l

$$0 < prob_l \leq 1. \tag{4}$$

This implies that for A $\subset \Omega$ the following holds:

$$P(A) = \sum_{l' \in A} prob_{l'} < \sum_{l \in \Omega} prob_l = 1.$$

Moreover, we get from (4) that as long as A is non-empty the following holds:

$$0 < \sum_{l' \in A} prob_{l'} = P(A).$$

For $A = \emptyset$ we have $P(A) = 0$. $\square$

Sets of paths that give us information about the probability to run through a given configuration at least once are summarized as *sets of minimal paths.*

Definition 7. *A decision path l is called* ***minimal path*** *from a configuration $\langle\!|V, s|\!\rangle$ to a configuration $\langle\!|V', s'|\!\rangle$ if either*

$$V = V' \wedge s = s' \wedge l = [(\langle\!|V, s|\!\rangle, 1, \circ)]$$

or

$$\begin{aligned} &l \textit{ is a decision path from } \langle\!|V, s|\!\rangle \textit{ to } \langle\!|V', s'|\!\rangle \textit{ and} \\ &\neg\exists l' \in DP, p.\ \langle\!|V, s|\!\rangle \xrightarrow{p}\!\!\!\!\rightarrow_{l'} \langle\!|V', s'|\!\rangle \wedge l' \sqsubset l. \end{aligned}$$

We denote the set of all minimal paths from $\langle\!|V, s|\!\rangle$ to $\langle\!|V', s'|\!\rangle$ by $\min^{\langle\!|V,s|\!\rangle}_{\langle\!|V',s'|\!\rangle}$.

Example 1. Figure 4 illustrates two possible probability trees (the first one represented by only the dotted lines, the second one represented by both the dotted and the solid lines) for the configuration $\langle\!|C, s|\!\rangle$ with

$$\begin{aligned} C &= \ \mathsf{fork}(l := 1, \langle frk \rangle) \quad \textit{and} \\ frk &= \mathsf{fork}(l := 2, \langle l := 1 \rangle). \end{aligned}$$

The labels of the arrows denote the probability for the decision path starting in $\langle\!|C, s|\!\rangle$ and ending at the destination of the arrow. The set of minimal paths from $\langle\!|C, s|\!\rangle$ to $\langle\!|\langle l := 2 \mid l := 1 \rangle, s[l = 1]|\!\rangle$ contains the following elements:

$$path_1 = [(\langle\!| C, s |\!\rangle,\ 1,\ \circ),\ (\langle\!| \langle l := 1 \mid frk \rangle, s |\!\rangle,\ 1,\ 1),\ \ (\langle\!| frk, s[l = 1] |\!\rangle,\ 1/2,\ 1),$$
$$(\langle\!| \langle l := 2 \mid l := 1 \rangle, s[l = 1] |\!\rangle,\ 1,\ 1)] \quad \text{and}$$

$$path_2 = [(\langle\!| C, s |\!\rangle,\ 1,\ \circ),\ (\langle\!| \langle l := 1 \mid frk \rangle, s |\!\rangle,\ 1,\ 1),$$
$$(\langle\!| l := 1 \mid l := 2 \mid l := 1, s |\!\rangle,\ 1/2,\ 2),$$
$$(\langle\!| \langle l := 2 \mid l := 1 \rangle, s[l = 1] |\!\rangle,\ 1/3,\ 1)]$$

$\Diamond$

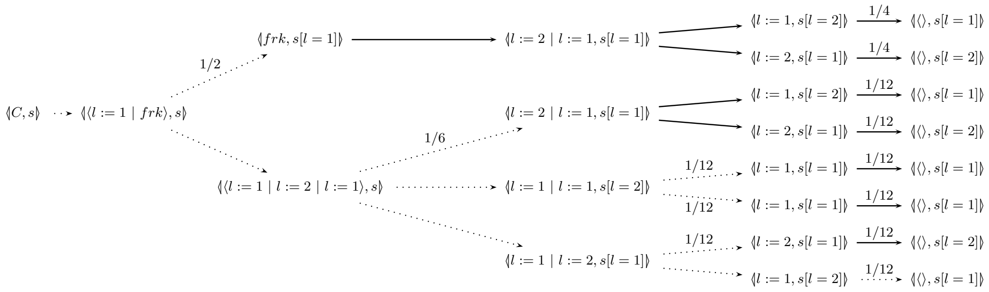


Fig. 4. Two possible probability trees for $\langle\!| C, s |\!\rangle$.

The probability to run through a given configuration at least once can now be computed with the help of the set of minimal paths to that configuration:

Definition 8. *Let h be a minimal path from the configuration $\langle\!| V, s |\!\rangle$ to the configuration $\langle\!| V', s' |\!\rangle$ and let p be the probability of h. For each path l with $h \sqsubseteq l$ the probability p is called the* ***reaching probability*** *for $\langle\!| V', s' |\!\rangle$* ***along the path*** *$l$ (denoted by $prob_{l, \langle\!| V', s' |\!\rangle}$).*

Definition 9. *The* ***reaching probability*** *to run from a configuration $\langle\!| V, s |\!\rangle$ to a configuration $\langle\!| V', s' |\!\rangle$ (denoted by $prob_{\langle\!| V, s |\!\rangle, \langle\!| V', s' |\!\rangle}$) is defined by:*

$$prob_{\langle\!| V,s |\!\rangle, \langle\!| V',s' |\!\rangle} = \sum_{l \in min^{\langle\!| V,s |\!\rangle}_{\langle\!| V',s' |\!\rangle}} prob_{l, \langle\!| V',s' |\!\rangle}$$

Example 2. The reaching probability from the configuration $\langle\!| C, s |\!\rangle$ from Example 1 to the configuration $\langle\!| \langle l := 2 \mid l := 1 \rangle, s[l = 1] |\!\rangle$ is the following:

$$\begin{aligned} prob_{\langle\!| C,s |\!\rangle, \langle\!| \langle l:=2 \mid l:=1 \rangle, s[l=1] |\!\rangle} &= prob_{path_1, \langle\!| \langle l:=2 \mid l:=1 \rangle, s[l=1] |\!\rangle} + \\ & prob_{path_2, \langle\!| \langle l:=2 \mid l:=1 \rangle, s[l=1] |\!\rangle} \\ &= 1/2 + 1/6. \end{aligned}$$

$\Diamond$

Proposition 2. *The reflexive transitive closure of $\rightarrow$ (denoted by $\rightarrow^*$) can now be expressed as follows:*

$$\langle\!| V, s |\!\rangle \rightarrow^* \langle\!| V', s' |\!\rangle \iff prob_{\langle\!| V,s |\!\rangle, \langle\!| V',s' |\!\rangle} > 0$$

3 Security Condition

As in [MSK07], we aim for a security definition that characterizes the possible influences of an initial state on the set of memory states after termination. According to the termination behavior we can divide the set of programs into three different types. First of all, we have programs that always evaluate to the empty program and hence, always terminate. Next, we have programs that never result in the empty program and, hence, always diverge. Finally, we have programs that have the possibility to terminate but do not do it for sure. Our security condition should provide an appropriate statement for all three types of programs.

Definition 10. *A program V is **deterministic terminating in the state** s if*

$$\exists n \in \mathbb{N}.\ \forall V', s', p, l.\ \left(\langle\!| V, s |\!\rangle \overset{p}{\twoheadrightarrow}_l \langle\!| V', s' |\!\rangle \Longrightarrow \mathit{length}(l) \leq n \right)$$

*holds.The program V is **deterministic terminating** if it is deterministic terminating under all states.*[2]

All deterministic terminating programs, started in an arbitrary state, terminate with probability 1:

Lemma 3. *Let C be a deterministic terminating program and s a state, then*

$$\sum_{s'} \mathit{prob}_{\langle\!| C,s |\!\rangle, \langle\!| \langle\rangle, s' |\!\rangle} = 1.$$

Proof. From Definition 10 we have that

$$\forall s. \exists n \in \mathbb{N}.\ \forall V', s', p, l.\ \langle\!| C, s |\!\rangle \overset{p}{\twoheadrightarrow}_l \langle\!| V', s' |\!\rangle \Longrightarrow \mathit{length}(l) \leq n.$$

Hence, all paths are bounded above by n and there exists a fully expanded probability tree for each configuration $\langle\!| C, s |\!\rangle$ (denoted by TR_s). Given s we can generate TR_s by starting with $\{[(\langle\!| C, s |\!\rangle, 1, \circ)]\}$ and repeatedly using the second rule of Definition 5 to substitute paths that do not have the empty program as right most program. Since all paths are shorter than n and we have no possibility to fork an infinite set of threads this procedure terminates. Moreover since we assume only syntactically correct programs we know from the semantics that the generated tree has the empty program as right most program in each of its decision paths. From Proposition 1 we know

$$\sum_{l \in TR_s} \mathit{prob}_l = 1.$$

Hence, the probability to reach the empty program is 1. □

[2] With this definition, the length of evaluation sequences of deterministic terminating programs is bounded from above by some $n \in \mathbb{N}$. It thus might appear to exclude programs that have arbitraryly long execution sequences, but nevertheless always terminate. But such programs are not possible in our programming language, since in each step only finitely many threads can be forked.

Definition 11. *A program V is **deterministic non-terminating** in a state s if the following holds:*

$$\forall s'.prob_{\langle\!| V,s |\!\rangle, \langle\!| \langle\rangle, s' |\!\rangle} = 0$$

The program V is (generally) deterministic non-terminating if it is deterministic non-terminating in all states.

Definition 12. *A program V is **possibly terminating** if it is neither deterministic terminating nor deterministic non-terminating.*

Example 3. Let C_1=(h:=False) and C_2=(h:=True;while h do skip od). Then C_1 is deterministic terminating, C_2 is deterministic non-terminating and $\langle C_1 \mid C_2 \rangle$ is possibly terminating. $\diamondsuit$

To formalize our instance of the indistinguishability property we use a two-level security lattice with a distinction between high (secure) variables and low (public) variables where no information is allowed to flow from high to low. We will refer to states that differ only in the values of the high variables as low-equal states (denoted by $s =_L t$) and summarize the set of all states that are low equal to a state s as low-equivalence class of s (denoted by $[s]_{=_L}$). Two expressions are low equivalent (denoted by $Exp_1 \equiv_L Exp_2$) if they evaluate to the same value for each pair of low-equal states.

We assume an attacker who can see the initial low memory and the final low memory of terminated programs. Moreover, we give him the possibility to re-run the program as often as he wishes. We classify a program as secure if it fulfills the following requirements:

- the termination probability is independent of initial values of high variables
- the probability to terminate in a given set of low-equal states is independent of the initial value of high variables

The probabilities for the sets of low equal final states of a secure program should be distributed equally and, hence, indistinguishable for an attacker if we start the execution in two low equal states.

Definition 13. *Two thread pools V and W are **low indistinguishable** (denoted by $V \sim_L^p W$) iff the following holds:*

$$\forall s,t,r \in S.\ s =_L t \Longrightarrow \sum_{s' \in [r]_{=_L}} prob_{\langle\!| V,s |\!\rangle, \langle\!| \langle\rangle, s' |\!\rangle} = \sum_{t' \in [r]_{=_L}} prob_{\langle\!| W,t |\!\rangle, \langle\!| \langle\rangle, t' |\!\rangle}$$

Definition 14. *A program V is **probBL secure** if*

$$V \sim_L^p V.$$

This definition is applicable for each type of program identified at the beginning of this section. Moreover, for deterministic non-terminating programs the probability to terminate is 0 independent of the starting state. Hence, we can easily show the following Lemma:

Lemma 4. *All deterministic non-terminating programs are probBL secure.*

Proof. Due to the definition of probBL security and low indistinguishability. $\square$

4 Combining Calculus

In the previous section we have introduced a definition of program security. The next step is to support the check of programs against probBL security. Instead of developing a new proof system from scratch we reuse the calculus from [MSK07].

4.1 Compositionality and Basic Calculus

Sequential and parallel composition do not preserve probBL security as the following examples show:

Example 4. Let $C_1 = (\mathsf{if}\ h\ \mathsf{then}\ l := 0\ \mathsf{else}\ l := 1\ \mathsf{fi}; l := 2)$ and $C_2 = (\mathsf{fork}(\mathsf{skip}, \langle C_1 \rangle))$. Both programs are probBL secure. Nevertheless a parallel respectively sequential composition with the probBL secure program $l' := l$ result in programs that are not probBL secure. ◇

The program $\mathsf{fork}(C_1, \langle l' := l \rangle)$ in Example 4 demonstrates that races are a major problem during parallel composition. To avoid races we demand *variable independence* of concurrent threads during parallel composition:

Definition 15 ([MSK07]). *Two thread pools V and W are* ***variable independent*** *(denoted by $V \gtrless W$) if the sets of variables occurring in V respectively W are disjoint.*

Example 4 demonstrates that races are not only a problem for security when composing programs in parallel, but also in a sequential composition. These races originate from the termination of the first (main) thread before the termination of spawned threads. To avoid those races we use a property that guarantees that the main thread is the last one that terminates:

Definition 16 ([MSK07]). *V is* ***main-surviving*** *in s (denoted by $\langle\!| V, s |\!\rangle \succ$) iff*

$$\forall D_1 \dots D_n, t.\ \langle\!| V, s |\!\rangle \rightarrow^* \langle\!| \langle D_1 | \dots | D_n \rangle, t |\!\rangle \Longrightarrow (n = 1 \lor (\neg \exists t'.\ \langle\!| D_1, t |\!\rangle \twoheadrightarrow \langle\!| \langle\rangle, t' |\!\rangle)).$$

A non main-surviving program can be transformed into a main-surviving program by adding sync-statements into the main thread.

The repeated execution of a probBL secure program can either be a secure composition if the number of iterations depends only on low variables or the number of iterations cannot be reproduced by the observation of the low memory after termination. To formulate the second case we introduce the following property, which is inspired by [BC01]:

Definition 17. *A program V is* ***free of low affectings*** *(denoted by $H(V)$) if the following holds:*

$$\forall s, s', V'.\ \langle\!| V, s |\!\rangle \rightarrow^* \langle\!| V', s' |\!\rangle \Longrightarrow s =_L s'$$

The following two lemmas will help us to prove our compositionality result:

Lemma 5. *Let l=$[(\langle\!| V, s |\!\rangle, 1, \circ), (\langle\!| V_1, s_1 |\!\rangle, p_1, i_1), \dots, (\langle\!| V_n, s_n |\!\rangle, p_n, i_n)]$ respectively l'=$[(\langle\!| V', s' |\!\rangle, 1, \circ), (\langle\!| V_1', s_1' |\!\rangle, p_1', i_1'), \dots, (\langle\!| V_k', s_k' |\!\rangle, p_k', i_k')]$ be decision paths such that*

$$V_n = V' \land s_n = s'.$$

Then

$$h = [\,(\langle\!|V, s|\!\rangle, 1, \circ), (\langle\!|V_1, s_1|\!\rangle, p_1, i_1), \ldots, (\langle\!|V_n, s_n|\!\rangle, p_n, i_n),\\ (\langle\!|V'_1, s'_1|\!\rangle, p'_1, i'_1), \ldots, (\langle\!|V'_k, s'_k|\!\rangle, p'_k, i'_k)]$$

is a decision path from $\langle\!|V, s|\!\rangle$ to $\langle\!|V'_k, s'_k|\!\rangle$. Moreover, we have $prob_h = prob_l * prob_{l'}$.

Proof. Due to the use of the uniform scheduler the transition probabilities are independent of the history. Hence, the decision-steps of l' are the same independent if the execution starts in $\langle\!|V', s'|\!\rangle$ or if a sequence of transitions leads to $\langle\!|V', s'|\!\rangle$ respectively $\langle\!|V_n, s_n|\!\rangle$. From the rules of Figure 3 we have also that $prob_h = prob_l * prob_{l'}$. □

Lemma 6.

$$\forall C, V', V'', s, s', s''.\ ((\langle\!|\langle C\rangle, s|\!\rangle \neq \langle\!|V'', s''|\!\rangle \wedge \langle\!|C, s|\!\rangle \twoheadrightarrow \langle\!|V', s'|\!\rangle) \Longrightarrow\\ prob_{\langle\!|C,s|\!\rangle, \langle\!|V'',s''|\!\rangle} = prob_{\langle\!|V',s'|\!\rangle, \langle\!|V'',s''|\!\rangle})$$

Proof. Let $l = [(\langle\!|\langle C\rangle, s|\!\rangle, 1, \circ), (\langle\!|V', s'|\!\rangle, p, i)]$ be a decision path. Since $\langle C\rangle$ is single-threaded we know that there exists exactly one such path with $p = 1, i = 1$. From Lemma 5 we know that for each path l' from $\langle\!|V', s'|\!\rangle$ to $\langle\!|V'', s''|\!\rangle$ there exists a path h with

$$\begin{aligned} h =&\ \ l + +l'^{-} \quad \mathit{and} \\ prob_h =&\ prob_l * prob_{l'} \\ =&\ \ 1 * prob_{l'}. \end{aligned}$$

From this and the definition of minimal paths we have

$$\sum_{l' \in min^{\langle\!|V',s'|\!\rangle}_{\langle\!|V'',s''|\!\rangle}} prob_{l', \langle\!|V'',s''|\!\rangle} \\ = \sum_{\substack{l' \in min^{\langle\!|V',s'|\!\rangle}_{\langle\!|V'',s''|\!\rangle}, \\ h = l++l'^{-}}} prob_{h, \langle\!|V'',s''|\!\rangle}.$$

Since we excluded the case where $\langle\!|C, s|\!\rangle = \langle\!|V'', s''|\!\rangle$ we know that for each minimal path l' the corresponding path h is a minimal path from $\langle\!|C, s|\!\rangle$ to $\langle\!|V'', s''|\!\rangle$. Moreover, since there is only one possible transition for $\langle\!|\langle C\rangle, s|\!\rangle$, there are no other minimal paths to $\langle\!|V'', s''|\!\rangle$. Hence, we have

$$\sum_{l \in min^{\langle\!|V',s'|\!\rangle}_{\langle\!|V'',s''|\!\rangle}} prob_{l, \langle\!|V'',s''|\!\rangle} = \sum_{h \in min^{\langle\!|C,s|\!\rangle}_{\langle\!|V'',s''|\!\rangle}} prob_{h, \langle\!|V'',s''|\!\rangle}.$$

Lemma 6 follows by Definition 9. □

For a composition of the form $\mathsf{if}\ B\ \mathsf{then}\ C\ \mathsf{else}\ D\ \mathsf{fi}$ with $B \not\equiv_L B$ we introduce the notion of *branch low-equivalence.*

Definition 18. *Let Exp_s denote the value n with $\langle\!| Exp, s |\!\rangle \downarrow n$. Moreover, let C_1 and C_2 be probBL secure programs. A program $C =$ if B then C_1 else C_2 fi is* ***branch low-equivalent*** *(denoted by $if^B_{C_1 \sim^p_L C_2}$) if the following holds:*

$$\forall s, t, r. \left((s =_L t \wedge B_s \neq B_t) \Longrightarrow \sum_{s' \in [r]_{=_L}} prob_{\langle\!| C_1, s |\!\rangle, \langle\!| \langle\rangle, s' |\!\rangle} = \sum_{t' \in [r]_{=_L}} prob_{\langle\!| C_2, t |\!\rangle, \langle\!| \langle\rangle, t' |\!\rangle} \right)$$

Remark 1. Branch low-equivalence allows to analyze two different cases: Firstly, the case that the guard is constant on low-equivalence classes, i.e., the choice between C_1 and C_2 does not depend on the high part of the state. And secondly, the case that the probabilities to terminate in a given low equivalence class in one of the two branches C_1 and C_2 do not depend on the high part of the state. In both cases, branch low-equivalence is satisfied.

But branch low-equivalence admits even more programs to be checked. Consider the program $C =$ if $(h^l = 1)$ then C_1 else C_2 fi with $C_1 =$ if $l = 0$ then $l := 1$ else skip fi and $C_2 =$ if $l = 0$ then $l := 2$ else skip fi. The guard is only constant on the low equivalence class of $l = 0$, while the probabilities to terminate in a given low equivalence class of the two branches differ only on the low equivalence class of $l = 0$. But still $if^{(h^l=1)}_{C_1 \sim^p_L C_2}$ holds. $\Diamond$

Theorem 1. *Let C, D, V, W be probBL secure programs. The following table describes under which condition (left side) the compositions on the right side preserve probBL security:*

Condition	*probBL Secure Composition*
$\forall s \in S. \langle\!| C, s |\!\rangle \succ$	$C;D$
$if^B_{C \sim^p_L D}$	if B then C else D fi
$(B \equiv_L B \wedge \forall s \in S. \langle\!| C, s |\!\rangle \succ)$	while B do C od
$C \gtrless W$	$\mathsf{fork}(C, W)$
$V \gtrless W$	$\langle V W \rangle$

Proof. Let E denote the particular composition for each sub-theorem. Then we have to show that $E \sim^p_L E$ holds, if the respective condition is satisfied.

C;D Since D is probBL secure we have the following:

$$\forall s', t', r', r''.\ \ s', t' \in [r']_{=_L} \Longrightarrow \sum_{s'' \in [r'']_{=_L}} prob_{\langle\!| D, s' |\!\rangle, \langle\!| \langle\rangle, s'' |\!\rangle} = \sum_{t'' \in [r'']_{=_L}} prob_{\langle\!| D, t' |\!\rangle, \langle\!| \langle\rangle, t'' |\!\rangle} \quad (5)$$

Moreover, from the rules in Figure 1 we know that D is only executed if the main thread of C has terminated before. Since C is main-surviving we know that the main thread of C only terminates if all other threads have terminated before. Hence, we know that D is executed after the full termination of C and, hence, has no influence on the scheduling of C. Together with the probBL security of C this results in:

$$\begin{aligned}\forall s,t,r'.\;\; s =_L t \Longrightarrow & \sum_{s' \in [r']_{=_L}} prob_{\langle\!| C;D,s |\!\rangle, \langle\!| D,s' |\!\rangle} \\ & = \sum_{t' \in [r']_{=_L}} prob_{\langle\!| C;D,t |\!\rangle, \langle\!| D,t' |\!\rangle}\end{aligned} \tag{6}$$

Since s' and t' are in the same low-equivalence class $[r']_{=_L}$ we can combine (5) and (6) to

$$\begin{aligned}&\forall s,t,r',r''.\;\; s =_L t \Longrightarrow \\ &\quad \sum_{s' \in [r']_{=_L}} \left(prob_{\langle\!| C;D,s |\!\rangle, \langle\!| D,s' |\!\rangle} * \Big(\sum_{s'' \in [r'']_{=_L}} prob_{\langle\!| D,s' |\!\rangle, \langle\!| \langle\rangle, s'' |\!\rangle} \Big) \right) \\ &\quad = \sum_{t' \in [r']_{=_L}} \left(prob_{\langle\!| C;D,t |\!\rangle, \langle\!| D,t' |\!\rangle} * \Big(\sum_{t'' \in [r'']_{=_L}} prob_{\langle\!| D,t' |\!\rangle, \langle\!| \langle\rangle, t'' |\!\rangle} \Big) \right).\end{aligned} \tag{7}$$

This can be done since both sides use only states s' and t' from the same low-equivalence class. Hence, by (5) the second coefficient is a constant. Since we assume a uniform scheduler the execution of C has no influence on the scheduling of D and the Equation (7) states that the probabilities to run from a configuration $\langle\!| C;D,s |\!\rangle$ respectively $\langle\!| C;D,t |\!\rangle$ through a configuration $\langle\!| D,s' |\!\rangle$ respectively $\langle\!| D,t' |\!\rangle$ with $s',t' \in [r']_{=_L}$ and finally terminate in the low equivalence class $[r'']_{=_L}$ are equal for two low equal starting states s and t. Since the inner sums of Equation (7) are constants for given s' and t' this can be transformed into the following equation:

$$\begin{aligned}&\forall s,t,r',r''.\;\; s =_L t \Longrightarrow \\ &\quad \sum_{s' \in [r']_{=_L}} \sum_{s'' \in [r'']_{=_L}} \left(prob_{\langle\!| C;D,s |\!\rangle, \langle\!| D,s' |\!\rangle} * prob_{\langle\!| D,s' |\!\rangle, \langle\!| \langle\rangle, s'' |\!\rangle} \right) \\ &\quad = \sum_{t' \in [r']_{=_L}} \sum_{t'' \in [r'']_{=_L}} \left(prob_{\langle\!| C;D,t |\!\rangle, \langle\!| D,t' |\!\rangle} * prob_{\langle\!| D,t' |\!\rangle, \langle\!| \langle\rangle, t'' |\!\rangle} \right)\end{aligned}$$

Since we have a quantification over all states r' we can sum up over all possible low-equivalence classes (LEC) and get the following equation:

$$\begin{aligned}&\forall s,t,r''.\;\; s =_L t \Longrightarrow \\ &\quad \sum_{[r']_{=_L} \in LEC} \left(\sum_{s' \in [r']_{=_L}} \sum_{s'' \in [r'']_{=_L}} \left(prob_{\langle\!| C;D,s |\!\rangle, \langle\!| D,s' |\!\rangle} * prob_{\langle\!| D,s' |\!\rangle, \langle\!| \langle\rangle, s'' |\!\rangle} \right) \right) \\ &\quad = \sum_{[r']_{=_L} \in LEC} \left(\sum_{t' \in [r']_{=_L}} \sum_{t'' \in [r'']_{=_L}} \left(prob_{\langle\!| C;D,t |\!\rangle, \langle\!| D,t' |\!\rangle} * prob_{\langle\!| D,t' |\!\rangle, \langle\!| \langle\rangle, t'' |\!\rangle} \right) \right)\end{aligned}$$

The outer most two sums can be summarized as follows:

$$\sum_{[r']_{=_L} \in LEC} \left(\sum_{s' \in [r']_{=_L}} p(s') \right) = \sum_{s' \in S} p(s')$$

Hence, we have

$$\begin{aligned}\forall s,t,r''.\quad & s =_L t \Longrightarrow \\ & \sum_{s'\in S}\sum_{s''\in[r'']_{=_L}} \Big(prob_{\langle\!|C;D,s|\!\rangle,\langle\!|D,s'|\!\rangle} * prob_{\langle\!|D,s'|\!\rangle,\langle\!|\langle\rangle,s''|\!\rangle}\Big) \\ &= \sum_{t'\in S}\sum_{t''\in[r'']_{=_L}} \Big(prob_{\langle\!|C;D,t|\!\rangle,\langle\!|D,t'|\!\rangle} * prob_{\langle\!|D,t'|\!\rangle,\langle\!|\langle\rangle,t''|\!\rangle}\Big).\end{aligned} \tag{8}$$

From the main-surviving of C and the rules of Figure 1 we know that the execution of D cannot start until C is completely executed. Hence, each decision path l from $\langle\!|C;D,s|\!\rangle$ to any configuration $\langle\!|\langle\rangle,s''|\!\rangle$ has the form:

$$\exists s',p',i'.l = \big[(\langle\!|C;D,s|\!\rangle,1,\circ),\ldots,(\langle\!|D,s'|\!\rangle,p',i'),\ldots(\langle\!|\langle\rangle,s|\!\rangle,p'',i'')\big]$$

Hence, for each such path l the probability can be computed by

$$prob_l = prob_{l_1} * prob_{l_2},$$

where $l_1 = [(\langle\!|C;D,s|\!\rangle,1,\circ),\ldots,(\langle\!|D,s'|\!\rangle,p',i')]$ is an initial segment of l, l_2' is a terminal segment of l such that $l = l_1 + +l_2'$, and $l_2 = [(\langle\!|D,s'|\!\rangle,1,\circ)] + +l_2'$. Hence, we can summarize (8) to

$$\begin{aligned}\forall s,t,r''.\quad s =_L t \Longrightarrow & \sum_{s''\in[r'']_{=_L}} (prob_{\langle\!|C;D,s|\!\rangle,\langle\!|\langle\rangle,s''|\!\rangle}) \\ =& \sum_{t''\in[r'']_{=_L}} (prob_{\langle\!|C;D,t|\!\rangle,\langle\!|\langle\rangle,t''|\!\rangle}).\end{aligned}$$

if B then C else D ($if^{B}_{C\sim^p_L D}$)

Let E abbreviate the program $\mathsf{if}\ B\ \mathsf{then}\ C\ \mathsf{else}\ D\ \mathsf{fi}$. For $B_s = B_t = \mathsf{True}$ the configuration $\langle\!|E,s|\!\rangle$ deterministically results in the new configuration $\langle\!|C,s|\!\rangle$. Moreover, from the conditions we have that C is probBL secure. Due to Definitions 14 and 13 that implies:

$$\begin{aligned}\forall s,t,r'\in S.\ s =_L t \Longrightarrow & \sum_{s'\in[r']_{=_L}} prob_{\langle\!|C,s|\!\rangle,\langle\!|\langle\rangle,s'|\!\rangle} \\ =& \sum_{t'\in[r']_{=_L}} prob_{\langle\!|C,t|\!\rangle,\langle\!|\langle\rangle,t'|\!\rangle}\end{aligned}$$

This and Lemma 6 give us

$$\begin{aligned}\forall s,t,r'\in S.\ B_s = B_t = \mathsf{True} \wedge\ s =_L t \Longrightarrow & \sum_{s'\in[r']_{=_L}} prob_{\langle\!|E,s|\!\rangle,\langle\!|\langle\rangle,s'|\!\rangle} \\ =& \sum_{t'\in[r']_{=_L}} prob_{\langle\!|E,t|\!\rangle,\langle\!|\langle\rangle,t'|\!\rangle}.\end{aligned}$$

By Definition 14 this implies for $B_s = B_t = \mathsf{True}$ that E is probBL secure. Following the same argumentation over the probBL security of D we can show that E is also probBL secure for $B_s = B_t = \mathsf{False}$.
Now, assume $B_s \neq B_t$. For the case $B_s = \mathsf{True}$ and $B_t = \mathsf{False}$ we have to show that

$$\forall r'\in S. \sum_{s'\in[r']_{=_L}} prob_{\langle\!|C,s|\!\rangle,\langle\!|\langle\rangle,s'|\!\rangle} = \sum_{t'\in[r']_{=_L}} prob_{\langle\!|D,t|\!\rangle,\langle\!|\langle\rangle,t'|\!\rangle},$$

and for the case $B_s = \mathsf{False}$ and $B_t = \mathsf{True}$ that

$$\forall r' \in S. \sum_{s' \in [r']_{=_L}} prob_{\langle\!|D,s|\!\rangle,\langle\!|\langle\rangle,s'|\!\rangle} = \sum_{t' \in [r']_{=_L}} prob_{\langle\!|C,t|\!\rangle,\langle\!|\langle\rangle,t'|\!\rangle}$$

holds.
We have both due to $if^B_{C \sim^p_L D}$ and by using Definition 18. Using Lemma 6 we can now show

$$\begin{aligned}\forall s,t,r' \in S.\ B_s \neq B_t \wedge\ s =_L t \Longrightarrow & \sum_{s' \in [r']_{=_L}} prob_{\langle\!|E,s|\!\rangle,\langle\!|\langle\rangle,s'|\!\rangle} \\ = & \sum_{t' \in [r']_{=_L}} prob_{\langle\!|E,t|\!\rangle,\langle\!|\langle\rangle,t'|\!\rangle}.\end{aligned}$$

while B do C od ($B \equiv_L B \wedge \forall s \in S.\langle\!|C,s|\!\rangle\!\succ$) Let $E = \mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od}$. Let the function $\#while : DP \to \mathbb{N}$ be defined as follows:

$$\#while(l) = \begin{cases} 0 & \text{if } length(l) = 0, \\ \#while(l^-) + 1 & \text{if } l = [(\langle\!|E,s|\!\rangle, p, i)] + +h \text{ for some } s,p,i,h, \\ \#while(l^-) & \text{otherwise.} \end{cases}$$

Hence, $\#while(l)$ is the number of configurations in a path l containing the program E. If l is a decision path from $\langle\!|E,s|\!\rangle$ for some state s we know from the rules from Figure 1 and the fact that C is main-surviving that such configurations only occur as initial configuration and every time C is completely executed. Hence, the number $\#while(l) - 1$ is the number of executions of C along the decision path l. We denote the set of decision paths l that end up with a configuration of the form $\langle\!|E,s|\!\rangle$ for some state s and for that $\#while(l) = n$ holds for a given n with P^n_E.
First we show that starting in two low equal states s and t and evaluating the body of the loop $(n-1)$ times will result with the same probability in a given low-equivalence class. Hence, we start by proving the following statement by induction over n:

$$\forall n \in \mathbb{N}_1, r', s, t.s =_L t \Longrightarrow \tag{9}$$

$$\sum_{\substack{l_s \in P^n_E, s' \in [r']_{=_L} \\ \exists i,q.l_s = [(\langle\!|E,s|\!\rangle,1,\circ)],\ldots,[(\langle\!|E,s'|\!\rangle,q,i)]}} prob_{l_s} = \sum_{\substack{l_t \in P^n_E, t' \in [r']_{=_L} \\ \exists i,q.l_t = [(\langle\!|E,t|\!\rangle,1,\circ)],\ldots,[(\langle\!|E,t'|\!\rangle,q,i)]}} prob_{l_t}$$

For $l \in P^1_E$ we have only one possible decision path with $l_s = [\langle\!|E,s|\!\rangle, 1, \circ]$. For any state t with $s =_L t$ we have the decision path $l_t = [\langle\!|E,t|\!\rangle, 1, \circ]$. Hence, for $n = 1$ the statement holds since $prob_{l_s} = prob_{l_t}$.
Assume now that (9) holds for $n \in \mathbb{N}_1$. From $B \equiv_L B$ we know that $B_{s'} = B_{t'}$ for $s', t' \in [r']_{=_L}$. With the semantics for while loops we therefore obtain for

all states r', s, t with $s =_L t$

$$\left(\sum_{\substack{l_s \in P_E^n, s' \in [r']_{=_L} \\ \exists i,q. l_s = ([(\langle\!| E,s |\!\rangle,1,\circ)], \\ \ldots,[(\langle\!| E,s' |\!\rangle,q,i)])}} prob_{l_s ++[(\langle\!| C;E,s' |\!\rangle,1,1)]} = \sum_{\substack{l_t \in P_E^n, t' \in [r']_{=_L} \\ \exists i,q. l_t = ([(\langle\!| E,t |\!\rangle,1,\circ)], \\ \ldots,[(\langle\!| E,t' |\!\rangle,q,i)])}} prob_{l_t ++[(\langle\!| C;E,t' |\!\rangle,1,1)]}\right) \wedge$$

$$\left(\sum_{\substack{l_s \in P_E^n, s' \in [r']_{=_L} \\ \exists i,q. l_s = ([(\langle\!| E,s |\!\rangle,1,\circ)], \\ \ldots,[(\langle\!| E,s' |\!\rangle,q,i)])}} prob_{l_s ++[(\langle\!| \langle\rangle,s' |\!\rangle,1,1)]} = \sum_{\substack{l_t \in P_E^n, t' \in [r']_{=_L} \\ \exists i,q. l_t = ([(\langle\!| E,t |\!\rangle,1,\circ)], \\ \ldots,[(\langle\!| E,t' |\!\rangle,q,i)])}} prob_{l_t ++[(\langle\!| \langle\rangle,t' |\!\rangle,1,1)]}.\right).$$

From the main-surviving of C and the semantic rule for sequential composition we know that C has to be completely executed before E is touched again and hence

$$\begin{aligned} \forall s, s', s'', i_s, k_s,\ q_s, i, p, l_s, l_{\langle\rangle}. \\ (\ l_s = [(\langle\!| E, s |\!\rangle, 1, \circ)] + +k_s + +[(\langle\!| E, s' |\!\rangle, q_s, i_s)] \wedge \\ l_{\langle\rangle} = l_s + +[(\langle\!| C; E, s' |\!\rangle, 1, 1), \ldots, (\langle\!| \langle\rangle, s'' |\!\rangle, p, i)]) \implies \\ \#while(l_{\langle\rangle}) \geq (\#while(l_s) + 1). \end{aligned}$$

Since C is main-surviving and C is probBL secure we have the following:

$$\forall r', r'', s' \in [r']_{=_L}, t' \in [r']_{=_L} . \tag{10}$$

$$\sum_{\substack{s'' \in [r'']_{=_L}, l_{s'} \in P_E^2 \\ l_{s'} = [(\langle\!| C;E,s' |\!\rangle,1,\circ),\ldots,(\langle\!| E,s'' |\!\rangle,1,1)]}} prob_{l_{s'}} = \sum_{\substack{t'' \in [r'']_{=_L}, l_{t'} \in P_E^2 \\ l_{t'} = [(\langle\!| C;E,t' |\!\rangle,1,\circ),\ldots,(\langle\!| E,t'' |\!\rangle,1,1)]}} prob_{l_{t'}}$$

The probability to execute C $(n-1)$ times resulting in a configuration $\langle\!| E, s' |\!\rangle$ with $s' \in [r']_{=_L}$ followed by another execution of C finally resulting in a configuration $\langle\!| E, s'' |\!\rangle$ with $s'' \in [r'']_{=_L}$ can now be computed by

$$\sum_{\substack{l_s \in P_E^n, s' \in [r']_{=_L} \\ \exists i,q. l_s = [(\langle\!| E,s |\!\rangle,1,\circ)], \\ \ldots,[(\langle\!| E,s' |\!\rangle,q,i)]}} \left[prob_{l_s ++[(\langle\!| C;E,s' |\!\rangle,1,1)]} * \left(\sum_{\substack{l_{s'} \in P_E^2, s'' \in [r'']_{=_L} \\ l_{s'} = [(\langle\!| C;E,s' |\!\rangle,1,\circ), \\ \ldots,(\langle\!| E,s'' |\!\rangle,1,1)]}} prob_{l_{s'}} \right) \right].$$

Using the identities from above this is equal to

$$\sum_{\substack{l_t \in P_E^n, t' \in [r']_{=_L} \\ \exists i,q. l_t = [(\langle\!| E,t |\!\rangle,1,\circ)], \\ \ldots,[(\langle\!| E,t' |\!\rangle,q,i)]}} \left[prob_{l_t ++[(\langle\!| C;E,t' |\!\rangle,1,1)]} * \left(\sum_{\substack{l_{t'} \in P_E^2, t'' \in [r'']_{=_L} \\ l_{t'} = [(\langle\!| C;E,t' |\!\rangle,1,\circ), \\ \ldots,(\langle\!| E,t'' |\!\rangle,1,1)]}} prob_{l_{t'}} \right) \right]$$

for all $s =_L t$. Using distributivity we transform the equation to:

$$\begin{aligned}
&\sum_{\substack{l_s\in P_E^n, s'\in[r']_{=_L}\\ \exists i,q. l_s=[(\langle\!|E,s|\!\rangle,1,\circ)],\\ \dots,[(\langle\!|E,s'|\!\rangle,q,i)]}} \sum_{\substack{l_{s'}\in P_E^2, s''\in[r'']_{=_L}\\ l_{s'}=[(\langle\!|C;E,s'|\!\rangle,1,\circ),\\ \dots,(\langle\!|E,s''|\!\rangle,1,1)]}} prob_{l_s++[(\langle\!|C;E,s'|\!\rangle,1,1)]} * prob_{l_{s'}}\\
=&\sum_{\substack{l_t\in P_E^n, t'\in[r']_{=_L}\\ \exists i,q. l_t=[(\langle\!|E,t|\!\rangle,1,\circ)],\\ \dots,[(\langle\!|E,t'|\!\rangle,q,i)]}} \sum_{\substack{l_{t'}\in P_E^2, t''\in[r'']_{=_L}\\ l_{t'}=[(\langle\!|C;E,t'|\!\rangle,1,\circ),\\ \dots,(\langle\!|E,t''|\!\rangle,1,1)]}} prob_{l_t++[(\langle\!|C;E,t'|\!\rangle,1,1)]} * prob_{l_{t'}}
\end{aligned}$$

Summing up over all low-equivalence classes $[r']_{=_L}$ we obtain

$$\begin{aligned}
&\sum_{\substack{l_s\in P_E^n, s'\in S\\ \exists i,q. l_s=[(\langle\!|E,s|\!\rangle,1,\circ)],\\ \dots,[(\langle\!|E,s'|\!\rangle,q,i)]}} \sum_{\substack{l_{s'}\in P_E^2, s''\in[r'']_{=_L}\\ l_{s'}=[(\langle\!|C;E,s'|\!\rangle,1,\circ),\\ \dots,(\langle\!|E,s''|\!\rangle,1,1)]}} prob_{l_s++[(\langle\!|C;E,s'|\!\rangle,1,1)]} * prob_{l_{s'}}\\
=&\sum_{\substack{l_t\in P_E^n, t'\in S\\ \exists i,q. l_t=[(\langle\!|E,t|\!\rangle,1,\circ)],\\ \dots,[(\langle\!|E,t'|\!\rangle,q,i)]}} \sum_{\substack{l_{t'}\in P_E^2, t''\in[r'']_{=_L}\\ l_{t'}=[(\langle\!|C;E,t'|\!\rangle,1,\circ),\\ \dots,(\langle\!|E,t''|\!\rangle,1,1)]}} prob_{l_t++[(\langle\!|C;E,t'|\!\rangle,1,1)]} * prob_{l_{t'}}.
\end{aligned}$$

In summary, we have for all s, t, r', r'' with $s =_L t$ that the following holds:

$$\sum_{\substack{s''\in[r'']_{=_L}, l_{s''}\in P_E^{(n+1)}\\ l_{s''}=[(\langle\!|E,s|\!\rangle,1,\circ),\\ \dots,(\langle\!|E,s''|\!\rangle,1,1)]}} prob_{l_{s''}} = \sum_{\substack{t''\in[r'']_{=_L}, l_{t''}\in P_E^{(n+1)}\\ l_{t''}=[(\langle\!|E,t|\!\rangle,1,\circ),\\ \dots,(\langle\!|E,t''|\!\rangle,1,1)]}} prob_{l_{t''}}$$

As we have already shown above if the equation holds for n then the probabilities to terminate in the next step in low equal states is equal for two runs starting in low equal states. Since we have now shown that the equation holds for all n we know that starting in two low equal states we have the same probability to execute C n times and then terminate in low equal states. Since the number of passes is not of interest for us we can sum up over all n and obtain for all s, t, r'' with $s =_L t$ that

$$\sum_{n\in\mathbb{N}_1} \sum_{\substack{s''\in[r'']_{=_L}\\ l=[(\langle\!|E,s|\!\rangle,1,\circ),\dots,(\langle\!|\langle\rangle,s''|\!\rangle,1,1)]\\ l\in P_E^n}} prob_l = \sum_{n\in\mathbb{N}_1} \sum_{\substack{t''\in[r'']_{=_L}\\ l=[(\langle\!|E,t|\!\rangle,1,\circ),\dots,(\langle\!|\langle\rangle,t''|\!\rangle,1,1)]\\ l\in P_E^n}} prob_l,$$

which can be summarized to

$$\sum_{\substack{s''\in[r'']_{=_L}\\ l=[(\langle\!|E,s|\!\rangle,1,\circ),\dots,(\langle\!|\langle\rangle,s''|\!\rangle,1,1)]}} prob_l = \sum_{\substack{t''\in[r'']_{=_L}\\ l=[(\langle\!|E,t|\!\rangle,1,\circ),\dots,(\langle\!|\langle\rangle,t''|\!\rangle,1,1)]}} prob_l.$$

Since we sum up over all paths to the empty program this is by Definition 9 the same as

$$\forall s, t, r''. s =_L t \Longrightarrow \sum_{s''\in[r'']_{=_L}} prob_{\langle\!|E,s|\!\rangle,\langle\!|\langle\rangle,s''|\!\rangle} = \sum_{t''\in[r'']_{=_L}} prob_{\langle\!|E,t|\!\rangle,\langle\!|\langle\rangle,t''|\!\rangle}.$$

$\langle$**VW**$\rangle$ Let $\Omega(A)$ denote the set of variables occurring syntactically in A. Since, V$\gtrless$W we have $\Omega(V) \cap \Omega(W) = \emptyset$. Moreover, let s_v and s_w be the parts of the memory s that can influence and can be written by V and W respectively, and let s' be the part of s that is neither read nor written by either V or W. Let $s_V \circ s_W \circ s'$ denote the combined memory of s_V and s_W. Hence, $s = s_V \circ s_W \circ s'$. We now color each thread occurring in the probability tree of $\langle\!|VW, s|\!\rangle$ with either red or green. Initially all threads occurring in V are colored red and all threads occurring in W are colored green. Every thread that is the result of the execution of a red (green) thread is also colored red (green). I.e., for each transition

$$\langle\!|\langle C_1..C_i..C_n\rangle, s|\!\rangle \xrightarrow{p,i} \langle\!|\langle C_1..U_i..C_n\rangle, s'|\!\rangle$$

all threads in U_i are colored with the same color as C_i.

Let n_V be the number of red threads in the current thread pool and n_W be the number of green threads. The probability for each non-blocked red (green) thread to be chosen as next thread is $1/(n_V - blkd(V) + n_W - blkd(W))$. Hence the relation between the probabilities of two non-blocked red (green) threads is 1:1. The probability for blocked threads is always 0 if not all threads have reached a sync-statement. It is the same relation as if $\langle\!|V, s_V|\!\rangle$ ($\langle\!|W, s_W|\!\rangle$) would be executed separately. If during the separated execution of V the choice of the i-th thread would cause the following transition

$$\langle\!|\langle C_1..C_i..C_n\rangle, s_V|\!\rangle \xrightarrow{1/(n_V - blkd(V)),i} \langle\!|\langle C_1..V_i..C_n\rangle, s'_V|\!\rangle$$

then due to the variable independence the choice of the i-th red thread in $\langle\!|VW, s_V \circ s_W \circ s'|\!\rangle$ would produce the following transition:

$$\langle\!|\langle C_1..C_i..C_n\rangle W, s_V \circ s_W \circ s'|\!\rangle \xrightarrow{p^*,i} \langle\!|\langle C_1..V_i..C_n\rangle W, s'_V \circ s_W \circ s'|\!\rangle,$$

where $p^* = 1/(n_V - blkd(V) + n_W - blkd(W))$. From $\Omega(V) \cap \Omega(W) = \emptyset$ we know that each green (red) thread is neither influenced by variables occurring in $\Omega(W)$ ($\Omega(V)$) nor will it change the value of any of these variables. Hence, the execution of a green (red) thread would only introduce an idle step for the execution of the red (green) thread pool.

Since this holds for each step in the execution the probabilities for each sequence of the execution of red (green) threads are the same independently if the red (green) threads are separated or in parallel with the green (red) threads. Let p_V (p_W) be the probability to run from a given state s (t) into a configuration with the empty program and a state from the low-equivalence class $[r_V]_{=_L}$ ($[r_W]_{=_L}$). Since both events are independent of each other we have the probability $p_V * p_W$ to run from the state $s_V \circ s_W \circ s'$ into a configuration with the empty program and a final state from $[r_V \circ r_W \circ r']_{=_L}$.

Hence, we have:

$$\forall s_V, s_W, s', r_V, r_W, r'. \; (p_V = \sum_{s'_V \in [r_V]_{=_L}} prob_{\langle\!|V, s_V|\!\rangle, \langle\!|\langle\rangle, s'_V|\!\rangle} \wedge$$
$$p_W = \sum_{s'_W \in [r_W]_{=_L}} prob_{\langle\!|W, s_W|\!\rangle, \langle\!|\langle\rangle, s'_W|\!\rangle} \Longrightarrow$$
$$p_V * p_W = \sum_{(s'_V \circ s'_W \circ s') \in [r_V \circ r_W \circ r']_{=_L}} prob_{\langle\!|VW, s_V \circ s_W \circ s'|\!\rangle, \langle\!|\langle\rangle, s'_V \circ s'_W \circ s'|\!\rangle})$$

Since V and W are probBL secure this implies

$$\forall s_V, s_W, s', t_V, t_W, t', r_V, r_W, r'. (s_V \circ s_W \circ s') =_L (t_V \circ t_W \circ t') \Longrightarrow$$
$$\sum_{(s'_V \circ s'_W \circ s') \in [r_V \circ r_W \circ r']_{=_L}} prob_{\langle\!|VW, s_V \circ s_W \circ s'|\!\rangle, \langle\!|\langle\rangle, s'_V \circ s'_W \circ s'|\!\rangle}) =$$
$$\sum_{(t'_V \circ t'_W \circ t') \in [r_V \circ r_W \circ r']_{=_L}} prob_{\langle\!|VW, t_V \circ t_W \circ t'|\!\rangle, \langle\!|\langle\rangle, t'_V \circ t'_W \circ t'|\!\rangle})$$

and hence, the probBL security of $\langle VW \rangle$.

fork(C,W) Due to the rules in Figure 1, the result of the former subproof, and the fact that $V = \langle C \rangle$ and W are probBL secure we have (where $\{\!|\dots|\!\}$ denotes a multiset):

$$\forall s, t, r \in S. \; s =_L t \Longrightarrow$$
$$\sum_{s' \in [r]_{=_L}} \{\!|\, p \mid \langle\!|\mathsf{fork}(C, W), s|\!\rangle \twoheadrightarrow \langle\!|VW, s|\!\rangle \wedge p = prob_{\langle\!|VW,s|\!\rangle} \langle\!|\langle\rangle, s'|\!\rangle |\!\} =$$
$$\sum_{t' \in [r]_{=_L}} \{\!|\, p \mid \langle\!|\mathsf{fork}(C, W), t|\!\rangle \twoheadrightarrow \langle\!|VW, t|\!\rangle \wedge p = prob_{\langle\!|VW,t|\!\rangle} \langle\!|\langle\rangle, t'|\!\rangle |\!\}$$

From Lemma 6 and by the Definitions 14, 13 this implies probBL security for $\mathsf{fork}(C, W)$.

□

Basic calculus rules of the combining calculus. The judgement $V \vdash probBL$ intuitively means that V is probBL secure.

$$[\text{SEQ}] \; \frac{C \vdash probBL \quad D \vdash probBL \quad \forall s \in S. \langle\!|C, s|\!\rangle \succ}{C; D \vdash probBL}$$

$$[\text{SNC}] \; \frac{C \vdash probBL}{C; \mathsf{sync} \vdash probBL} \qquad [\text{IF}] \; \frac{C \vdash probBL \quad D \vdash probBL \quad if^{B}_{C \sim^{p}_{L} D}}{if \; B \; then \; C \; else \; D \; fi \vdash probBL}$$

$$[\text{WHL}_{low}] \; \frac{C \vdash probBL \quad \forall s \in S. \; \langle\!|C, s|\!\rangle \succ \quad B \equiv_L B}{\mathsf{while} \; B \; \mathsf{do} \; C \; \mathsf{od} \vdash probBL}$$

$$[\text{PAR}] \; \frac{V \vdash probBL \quad W \vdash probBL \quad V \gtrless W}{VW \vdash probBL} \qquad [\text{FRK}] \; \frac{\langle C \rangle W \vdash probBL}{\mathsf{fork}(C, W) \vdash probBL}$$

Remark 2. In comparison to [MSK07], the rules for sequential composition, while loops, parallel composition,forking, and sync-statements remain unchanged. The rule for the if-construct was enhanced by relaxing the original assumption that the guard must be constant on low-equivalence classes: the new rule allows arbitrary guards, as long as the branches are not distinguishable by an attacker. E.g., the program $\mathsf{if}\ h_1\ \mathsf{then}\ h_2 = 0\ \mathsf{else}\ h_2 = 1\ \mathsf{fi}$ cannot be checked with the basic rules from [MSK07], but it can be checked with the rule [IF] (and a simple plugin rule for the assignments in the branches). That is, a modified rule results in a slightly more precise calculus.[3]

Hence the basic calculus rules from [MSK07] are sensible rules for both a possibilistic and a probabilistic baseline security property. ◇

5 Plugins for Semantic Security Definitions

In previous works the major practical effect of compositionality results was the possibility to split the proof of large programs into subproblems. The approach of [MSK07] goes one step further. Subprograms can be checked against different security definitions as long as each of these security definitions fulfills the following condition:

- The subprograms that are classified as secure by the considered security definition must also be secure according to the baseline definition of security.

For such security definitions we can formulate plugin rules to integrate them into the basic calculus.

Sections 5.1 and 5.2 illustrate the use of the plugin approach based on the integration of strong security [SS00] and low-deterministic security [ZM03]. A soundness result for the basic calculus and the plugin rules is provided in Section 5.3.

5.1 Strong-Security Plugin

Definition 19 ([SS00]). *The **strong low-bisimulation** (denoted by $\cong_L$) is the union of all symmetric relations R on command vectors $V = \langle C_1 \dots C_n \rangle$ and $W = \langle D_1 \dots D_n \rangle$ of equal size, such that*

$$(\forall s, s', t, V', i \in \{1 \dots n\}. \, (VRW \wedge s =_L t \wedge \langle\!| C_i, s |\!\rangle \twoheadrightarrow \langle\!| V', s' |\!\rangle) \Longrightarrow \\ (\exists W', t'. \langle\!| D_i, t |\!\rangle \twoheadrightarrow \langle\!| W', t' |\!\rangle \wedge s' =_L t' \wedge V'RW')).$$

Definition 20 ([SS00]). *A program V is **strongly secure** iff $V \cong_L V$ holds.*

As mentioned before we have to show that the set of strongly secure programs is a subset of probBL secure programs. However, this is not true in general. Consider, e.g., the program $C = \mathsf{sync}; l := h$. It is intuitively insecure, since the value of h is leaked to the public variable l after processing sync. Nevertheless it is strongly secure, since there is no program V such that $C \twoheadrightarrow V$ (processing the sync-statement results in a $\xrightarrow{1,\star}$-transition and not a $\twoheadrightarrow$ transition).

[3] This improvement is independent from the move from a possibilistic to a probabilistic characterization, which is the main contribution of this report.

Lemma 7. *Let V and W be strongly low-bisimilar programs that do not contain* sync*-statements. Let furthermore*

$$l_V = [(\langle\!| V, s |\!\rangle, 1, \circ), (\langle\!| V_1, s_1 |\!\rangle, p_1, i_1), \ldots, (\langle\!| V_m, s_m |\!\rangle, p_m, i_m)]$$

be a decision path. Then for $s =_L t$ there exist $W_1, \ldots, W_m, t_1, \ldots, t_m$ such that $[(\langle\!| W_1, t_1 |\!\rangle, q_1, j_1), \ldots, (\langle\!| W_m, t_m |\!\rangle, q_m, j_m)]$ is a decision path for suitable q_i, j_i, and such that $\forall j \in \{1..m\}.\ (V_j \approxeq_L W_j \wedge s_j =_L t_j)$. Moreover, the programs $W_1, \ldots, W_m$ and the states $t_1, \ldots, t_m$ are unique with this property.

Proof. Let $l_V = [(\langle\!| V, s |\!\rangle, 1, \circ)]$. We can choose $l_W = [(\langle\!| W, t |\!\rangle, 1, \circ)]$. Since there exists no second decision step in l_V and l_W the statement is true for $length(l_V) = 1$ (uniqueness is trivially fulfilled). Now, let $length(l_V)$=2 and $V = \langle C_1 | \ldots | C_n \rangle$ as well as $W = \langle D_1 | \ldots | D_n \rangle$. Since neither V nor W contain any sync-statements we have $blkd(V) = blkd(W) = 0 < n$. From Definition 19 we have:

$$\begin{aligned}\forall s, t, s'_i, V'_i, i \in \{1..n\}.((s =_L t \wedge\ \langle\!| C_i, s |\!\rangle \twoheadrightarrow \langle\!| V'_i, s'_i |\!\rangle) \Longrightarrow \\ (\exists W'_i, t'_i.\ \langle\!| D_i, t |\!\rangle \twoheadrightarrow \langle\!| W'_i, t'_i |\!\rangle \wedge s'_i =_L t'_i \wedge V'_i \approxeq_L W'_i))\end{aligned}$$

Hence, we have by the first rule of Figure 3 that the following holds:

$$\begin{aligned}&\forall s, t, s'_i, V'_i\ , i \in \{1..n\}.((s =_L t \wedge \\ &\qquad \langle\!| \langle C_1 | \ldots | C_i | \ldots | C_n \rangle, s |\!\rangle \xrightarrow{1/n, i} \langle\!| \langle C_1 | \ldots | V'_i | \ldots | C_n \rangle, s'_i |\!\rangle) \Longrightarrow \\ &(\exists W'_i, t'_i.\ \langle\!| \langle D_1 | \ldots | D_i | \ldots | D_n \rangle, t |\!\rangle \xrightarrow{1/n, i} \langle\!| \langle D_1 | \ldots | W'_i | \ldots | D_n \rangle, t'_i |\!\rangle \\ &\qquad \wedge s'_i =_L t'_i \wedge V'_i \approxeq_L W'_i))\end{aligned}$$

Moreover, since all threads except for C_i remain unchanged we know from $V \approxeq_L W$ and $V'_i \approxeq_L W'_i$ that $V' \approxeq_L W'$ holds for $V' = \langle C_1 | \ldots | V'_i | \ldots | C_n \rangle$ and $W' = \langle D_1 | \ldots | W'_i | \ldots | D_n \rangle$. If we now generate paths of length 2 by using the third rule of Figure 3 we obtain

$$\begin{aligned}&\forall s, t, s', V'\ , i \in \{1..n\}.((s =_L t \wedge \\ &\qquad \langle\!| V, s |\!\rangle \xrightarrow{1/n}\!\!\!\!\rightarrow_{[(\langle\!| V,s |\!\rangle,1,\circ),(\langle\!| V',s' |\!\rangle,1/n,i)]} \langle\!| V', s' |\!\rangle) \Longrightarrow \\ &(\exists W', t'.\ \langle\!| W, t |\!\rangle \xrightarrow{1/n}\!\!\!\!\rightarrow_{[(\langle\!| W,t |\!\rangle,1,\circ),(\langle\!| W',t' |\!\rangle,1/n,i)]} \langle\!| W', t' |\!\rangle \wedge s' =_L t' \wedge V' \approxeq_L W')).\end{aligned} \tag{11}$$

By the deterministic operational semantics there is only one choice for W'_i, and hence the constructed path is unique with the desired property. Since paths of length 2 can only be produced by a combination of the first and the third rule of Figure 3 or by a combination of the second and the third rule of Figure 3 we have shown that

$$\begin{aligned}&\forall s, t, l_V.\ (s =_L t \wedge\ l_V = [(\langle\!| V, s |\!\rangle, 1, \circ), (\langle\!| V_1, s_1 |\!\rangle, 1/n, i_1)]) \Longrightarrow \\ &\quad (\exists l_W.\quad (\quad l_W = [(\langle\!| W, t |\!\rangle, 1, \circ), (\langle\!| W_1, t_1 |\!\rangle, 1/n, i_1)] \wedge V_1 \approxeq_L W_1 \wedge s_1 =_L t_1)).\end{aligned}$$

Assume that for all l_V with $length(l_V) = m$ and $m \geq 2$ we can find an appropriate path l_W such that Lemma 7 holds (induction hypothesis). The only possibility to generate paths l'_V with $length(l'_V) = m + 1$ is provided by the last rule of Figure 3. From the condition we have that for each path l_V such that

$$\langle\!| V, s |\!\rangle \xrightarrow{p_1}\!\!\!\!\rightarrow_{l_V} \langle\!| V_m, s_m |\!\rangle$$

there exists a path l_W of length m such that

$$\langle\!| W, t |\!\rangle \xrightarrow{p_1}\!\!\!\!\rightarrow_{l_W} \langle\!| W_m, t_m |\!\rangle \wedge V_m \cong_L W_m \wedge s_m =_L t_m.$$

Moreover, we already have shown that from $V_m \cong_L W_m$ and $s_m =_L t_m$ for each path

$$l''_V = [(\langle\!| V_m, s_m |\!\rangle, 1, \circ), (\langle\!| V', s' |\!\rangle, p', i')]$$

exists a path

$$l''_W = [(\langle\!| W_m, t_m |\!\rangle, 1, \circ), (\langle\!| W', t' |\!\rangle, p', i')]$$

such that $V' \cong_L W'$ and $s' =_L t'$. Together this results in

$$\begin{aligned}
\forall s, t, l_V.\ (\ & s =_L t \wedge \\
& \quad l_V = [(\langle\!| V, s |\!\rangle, 1, \circ), \ldots, (\langle\!| V_j, s_j |\!\rangle, p_j, i_j), \ldots, \\
& \qquad\qquad (\langle\!| V_m, s_m |\!\rangle, p_m, i_m), (\langle\!| V', s' |\!\rangle, p', i')]) \Longrightarrow \\
& (\exists l_W.\ (\ l_W = [(\langle\!| W, t |\!\rangle, 1, \circ), \ldots, (\langle\!| W_j, t_j |\!\rangle, p_j, i_j), \ldots, \\
& \qquad\qquad (\langle\!| W_m, t_m |\!\rangle, p_m, i_m), (\langle\!| W', t' |\!\rangle, p', i')] \wedge \\
& \quad \forall j \in \{1..m\}.\ (V_j \cong_L W_j \wedge s_j =_L t_j) \wedge V' \cong_L W' \wedge s' =_L t')).
\end{aligned}$$

Again, one has no choice while constructing l_W, and hence l_W is unique with the desired property. Hence, Lemma 7 holds for all l'_V with $length(l'_V) = m + 1$. □

Theorem 2. *Let V be a strongly secure program that does not contain any* sync*-statements. Then V is probBL secure.*

Proof. We show $V \cong_L W \Longrightarrow V \sim^p_L W$. From this Theorem 2 follows instantly by Definition 14 and Definition 20.

Assume $V \cong_L W$. The only program without sync-statements that is strongly low-bisimilar to the empty program is the empty program itself. Hence, we have by Lemma 7 that the following holds:

$$\begin{aligned}
\forall s, t, l_V.\ (\ & s =_L t \wedge \\
& \quad l_V = [(\langle\!| V, s |\!\rangle, 1, \circ), \ldots, (\langle\!| \langle\rangle, s_n |\!\rangle, p_n, i_n)]) \Longrightarrow \\
& (\exists l_w.\ (\ l_W = [(\langle\!| W, t |\!\rangle, 1, \circ), \ldots, (\langle\!| \langle\rangle, t_n |\!\rangle, p_n, i_n)] \wedge \\
& \quad \forall j \in \{1..n\}.\ (V_j \cong_L W_j \wedge s_j =_L t_j)))
\end{aligned}$$

Moreover, we know that all paths to the empty program are minimal paths to the empty program and from the semantics of our language we know that for a path l to the empty program the following holds:

$$\neg \exists l'. l \sqsubset l'.$$

Hence, we know from the equation above and Definition 4 that the following holds:

$$\begin{aligned}
& \forall s, t, s', l_V.\ l_V \in min^{\langle\!| V, s |\!\rangle}_{\langle\!| \langle\rangle, s' |\!\rangle} \wedge s =_L t \Longrightarrow \\
& \quad (\exists t', l_W.\ l_W \in min^{\langle\!| W, t |\!\rangle}_{\langle\!| \langle\rangle, t' |\!\rangle} \wedge s' =_L t' \wedge prob_{l_V} = prob_{l_W})
\end{aligned}$$

Moreover, from the symmetry of $V \approxeq_L W$ we also have:

$$\forall s, t, t', l_W.\ l_W \in min^{\langle\!| W,t |\!\rangle}_{\langle\!| \langle\rangle, t' |\!\rangle} \wedge t =_L s \Longrightarrow$$
$$(\exists s', l_V.\ l_V \in min^{\langle\!| V,s |\!\rangle}_{\langle\!| \langle\rangle, s' |\!\rangle} \wedge t' =_L s' \wedge prob_{l_W} = prob_{l_V})$$

From the uniqueness property of Lemma 7 we know that there is only one path l_W matching a path l_V (and vice versa). Hence there is a bijection between the paths from $\langle\!| V, s |\!\rangle$ to the empty program and the paths from $\langle\!| W, t |\!\rangle$ to the empty program.

Hence, if we sum up over all probabilities $prob_{l_V}$ and $prob_{l_W}$ such that $l_V \in min^{\langle\!| V,s |\!\rangle}_{\langle\!| \langle\rangle, s' |\!\rangle}$ and $l_W \in min^{\langle\!| W,t |\!\rangle}_{\langle\!| \langle\rangle, t' |\!\rangle}$ and over the same set $[r']_{=_L}$ of low-equal final states we have:

$$\forall s, t, r'. s =_L t \Longrightarrow \sum_{s' \in [r']_{=_L}} \sum_{l_V \in min^{\langle\!| V,s |\!\rangle}_{\langle\!| \langle\rangle, s' |\!\rangle}} prob_{l_V} = \sum_{t' \in [r']_{=_L}} \sum_{l_W \in min^{\langle\!| W,t |\!\rangle}_{\langle\!| \langle\rangle, t' |\!\rangle}} prob_{l_W}$$

Due to Definition 9 and Definition 13 this is equal to

$$V \sim^p_L W.$$

□

That strongly secure programs form a proper subset of probBL secure programs even in the case of sync-free programs can be seen by the following example:

Example 5. A program violates strong security if it contains a loop that is reachable and whose guard depends on the values of high variables. Nevertheless, such programs may be probBL secure. Consider the following two programs:

$$\begin{aligned} P_1 &= \textsf{while } h > 0 \textsf{ do } h := h - 1 \textsf{ od};\ l := 1 \\ P_2 &= h := 1;\ \textsf{while } h = 1 \textsf{ do skip od}; l := 1 \end{aligned}$$

For all starting states we have the probability 1 respectively 0 to terminate the computation in $\langle\!| \langle\rangle, s[l = 1] |\!\rangle$. Hence, P_1 and P_2 are probBL secure.

◊

Theorem 2 provides a justification for the plugin rule from [MSK07] that allows one to use analysis techniques for strong security as a plugin for the combining calculus:

$$[\mathrm{P}_{SLS}]\ \frac{V \approxeq_L V \quad V \text{is } \textsf{sync-free}}{V \vdash probBL}$$

The integration of strong security provides the possibility to allow races between threads without loosing compositionality in general.

Example 6. Let $C_1 = (l := 0)$ and $C_2 = (l := 1)$. Since both C_1 as well as C_2 use the variable l we cannot use the rule $[PAR]$ to deduce probBL security of the composition of the two probBL secure programs. Nevertheless, using the compositionality result from [SS00] we can show that $\langle C_1 | C_2 \rangle$ is strongly secure. With the help of the plugin rule we can derive that the parallel composition of C_1 and C_2 is also probBL secure.

◊

5.2 Low-Deterministic Security Plugin

We use the notion of low-deterministic security formulated in [MSK07]:

Definition 21. *A program V is **low-deterministic** secure iff*

$$\forall s,t,s',t'.\, \big(\big(s =_L t \wedge \langle\!| V,s |\!\rangle \rightarrow^* \langle\!| \langle\rangle, s' |\!\rangle \wedge \langle\!| V,t |\!\rangle \rightarrow^* \langle\!| \langle\rangle, t' |\!\rangle \big) \Longrightarrow s' =_L t' \big)\,.$$

We cannot show the general case

$$\forall V.\ (V\ \textit{low deterministic secure} \implies V\ \textit{probBL secure}),$$

as for some possibly non-terminating programs this implication does not hold.

Example 7. The program

$$\textsf{while}\ h\ \textsf{do skip od};\, l := 0$$

is intuitively insecure: if it terminates the observer can be sure that the initial value of h was $\textsf{False}$. It is also rejected by Definition 14: Two low-equal starting states $s = [l = 1, h = \textsf{True}]$ and $t = [l = 1, h = \textsf{False}]$ would result in the probability 0 and the probability 1 respectively for a final state in the low-equivalence class where $l = 0$. Nevertheless, it is low-deterministic secure.

◊

For deterministic terminating and deterministic non-terminating programs the implication of probBL security for low-deterministic secure programs holds. Before we can give a formal proof of this proposition we show the following lemma:

Lemma 8. *If a deterministic-terminating program V is low-deterministic secure then the following holds:*

$$\forall s.\exists r'. \sum_{s' \in [r']_{=_L}} prob_{\langle\!| V,s |\!\rangle, \langle\!| \langle\rangle, s' |\!\rangle} = 1 \wedge \\ \sum_{t' \notin [r']_{=_L}} prob_{\langle\!| V,s |\!\rangle, \langle\!| \langle\rangle, t' |\!\rangle} = 0$$

Proof. From Lemma 3 we have

$$\forall s. \sum_{s' \in S} prob_{\langle\!| V,s |\!\rangle, \langle\!| \langle\rangle, s' |\!\rangle} = 1.$$

Hence,

$$\forall s,r'. \left(\sum_{s' \in [r']_{=_L}} prob_{\langle\!| V,s |\!\rangle, \langle\!| \langle\rangle, s' |\!\rangle} \right) + \left(\sum_{t' \notin [r']_{=_L}} prob_{\langle\!| V,s |\!\rangle, \langle\!| \langle\rangle, t' |\!\rangle} \right) = 1 \qquad (12)$$

and

$$\forall s.\exists r'. prob_{\langle\!| V,s |\!\rangle, \langle\!| \langle\rangle, r' |\!\rangle} > 0.$$

Because V is low-deterministic secure we have:

$$\forall s.\exists r'.\forall s'. \langle\!| V,s |\!\rangle \rightarrow^* \langle\!| \langle\rangle, s' |\!\rangle \Longrightarrow s' \in [r']_{=_L}$$

which implies

$$\forall s.\exists r'.\forall s'.prob_{\langle\!|V,s|\!\rangle,\langle\!|\langle\rangle,s'|\!\rangle} > 0 \Longrightarrow s' \in [r']_{=_L}$$

and, hence,

$$\forall s.\exists r'.\forall t' \notin [r']_{=_L} \; . \; prob_{\langle\!|V,s|\!\rangle,\langle\!|\langle\rangle,t'|\!\rangle} = 0.$$

This results in

$$\forall s.\exists r'. \sum_{t' \notin [r']_{=_L}} prob_{\langle\!|V,s|\!\rangle,\langle\!|\langle\rangle,t'|\!\rangle} = 0.$$

From this we get by (12):

$$\sum_{s' \in [r']_{=_L}} prob_{\langle\!|V,s|\!\rangle,\langle\!|\langle\rangle,s'|\!\rangle} = 1.$$

□

Theorem 3. *For all deterministic terminating programs V the following holds:*

$$V \text{ is low deterministic secure} \implies V \text{ is probBL secure.}$$

Proof. Let s and t be low-equal states. Assume s' and t' such that $\langle\!|V, s|\!\rangle \to^* \langle\!|\langle\rangle, s'|\!\rangle$ and $\langle\!|V, t|\!\rangle \to^* \langle\!|\langle\rangle, t'|\!\rangle$. This implies

$$prob_{\langle\!|V,s|\!\rangle,\langle\!|\langle\rangle,s'|\!\rangle} > 0 \wedge prob_{\langle\!|V,t|\!\rangle,\langle\!|\langle\rangle,t'|\!\rangle} > 0$$

and by Definition 21 $s' =_L t'$. From this we know for a low-equivalence class $[r']_{=_L}$ with $s' \in [r']_{=_L}$ that $t' \in [r']_{=_L}$ holds. Moreover, since Lemma 8 reasons about all states we have the following:

$$\begin{aligned} \sum_{s' \in [r']_{=_L}} prob_{\langle\!|V,s|\!\rangle,\langle\!|\langle\rangle,s'|\!\rangle} &= \\ \sum_{t' \in [r']_{=_L}} prob_{\langle\!|V,t|\!\rangle,\langle\!|\langle\rangle,t'|\!\rangle} &= 1 \\ \text{and} \qquad & \\ \forall [q']_{=_L} \neq [r']_{=_L} \; . \; \sum_{s'' \in [q']_{=_L}} prob_{\langle\!|V,s|\!\rangle,\langle\!|\langle\rangle,s''|\!\rangle} &= \\ \sum_{t'' \in [q']_{=_L}} prob_{\langle\!|V,t|\!\rangle,\langle\!|\langle\rangle,t''|\!\rangle} &= 0. \end{aligned}$$

By Definition 14 this leads to the probBL security of V. □

The following example shows that for deterministic terminating programs the set of low-deterministic secure programs is a proper subset of the probBL secure programs.

Example 8. The program $C = \mathsf{fork}(l := 1, \langle \mathsf{fork}(l := 2, \langle l := 1 \rangle) \rangle)$ from Example 1 has two possible final low-equivalence classes $[s[l = 1]]_{=_L}$ and $[s[l = 2]]_{=_L}$ (here, $s[l = x]$ denotes the state which is equal to s, but for the value of l, which equals x). Since there is no branching or looping in C the probabilities to reach these final low-equivalence classes are equal for all starting states. Hence, C is probBL secure. Nevertheless, since we have two possible final low-equivalence classes C is not low-deterministic secure.

$\Diamond$

Theorem 3 provides a justification for the plugin rule from [MSK07] that allows one to use low-deterministic security. A low-deterministic secure program is probBL secure if it is either deterministic terminating or deterministic non-terminating.

The judgment $V \models lds$ means that V is low-deterministic secure.

$$[\mathrm{P}_{LDS}]\ \frac{V \models lds, \quad \neg(V poss.term.)}{V \vdash probBL}$$

5.3 Soundness

The combining calculus is sound with respect to the following proposition:

Theorem 4. *Let V be a program. If $V \vdash probBL$ is derivable in the combining calculus then V is probBL secure.*

Proof. Except for [SNC] the soundness of the basic calculus follows instantly from Theorem 1. The rule [SNC] is sound due to the fact that sync neither changes the memory state nor the termination probability of the program C.

The plugin rules are sound due to Theorems 2 and 3. $\square$

6 Plugins for Type-based Analysis techniques

Plugin-rules for semantic characterizations of security offer the possibility to re-use analysis techniques that guarantee the security of a given program according to the given security definition. The combining calculus also provides plugin rules for integrating syntactic analysis techniques correctly. In order to be integrated, a plugin type system must fulfill the following condition:

- The set of typable programs is a subset of probBL secure programs.

For such type systems we can formulate plugin rules to integrate them into the basic calculus.

Section 6.1 illustrates the use of the plugin approach based on the integration of the transforming type system of [SS00]. Section 6.2 provides an integration of the type system of [BC01].

6.1 Sabelfeld and Sand's Security Type System

This subsection introduces a plugin for the type system introduced in [SS00] in the same way as it was done in [MSK07].

The strong security condition introduced by Sabelfeld and Sands [SS00] is a very restrictive security condition but it has the advantage that it is the "largest compositional indistinguishability-based security property that implies scheduler-independent security" [Sab03]. The type system of [SS00] provides a possibility to make the restrictive security definition easier to fulfill. Instead of just checking programs against strong security the judgments of the transforming type system transform the original program V into a strongly secure program V' that has type Sl (denoted by $V \hookrightarrow V' : Sl$). The type Sl contains information that is needed for the transformation process. Sabelfeld and Sands [SS00] provide the following theorem:

Theorem 5. *[SS00]*

$$V \hookrightarrow V' : Sl \Longrightarrow V' \approxeq_L V'$$

Since we have already shown that strongly secure programs that do not contain sync-statements are probBL secure we have by this theorem and Definition 20 that the transformed program is probBL secure. Let $V \hookrightarrow V' \vdash probBL$ denote that V can be transformed into a probBL secure program V'. We can justify the following plugin rule of our calculus:

$$[\mathrm{T}_{SS}]\ \frac{V \hookrightarrow V' : Sl \quad V \text{is } \mathsf{sync}\text{-free}}{V \hookrightarrow V' \vdash probBL}$$

Due to the non-transforming nature of the basic calculus it does not support the possibility to integrate a security property of a program different from the program that should be checked. Hence, the integration of the transforming type system into the combining calculus needs the introduction of further rules:

$$[\mathrm{MIX}_1]\ \frac{C \vdash probBL}{C \hookrightarrow C \vdash probBL} \qquad [\mathrm{SNC'}]\ \frac{C \hookrightarrow C' \vdash probBL}{C; \mathsf{sync} \hookrightarrow C'; \mathsf{sync} \vdash probBL}$$

$$[\mathrm{SEQ'}]\ \frac{C \hookrightarrow C' \vdash probBL \quad D \hookrightarrow D' \vdash probBL \quad \forall s \in S. \langle\!| C', s |\!\rangle \succ}{C; D \hookrightarrow C'; D' \vdash probBL}$$

$$[\mathrm{IF'}]\ \frac{C \hookrightarrow C' \vdash probBL \qquad D \hookrightarrow D' \vdash probBL \qquad if^{B}_{C' \sim^{p}_{L} D'}}{\mathsf{if}\ B\ \mathsf{then}\ C\ \mathsf{else}\ D\ \mathsf{fi} \hookrightarrow \mathsf{if}\ B\ \mathsf{then}\ C'\ \mathsf{else}\ D'\ \mathsf{fi} \vdash probBL}$$

$$[\mathrm{WHL'}]\ \frac{C \hookrightarrow C' \vdash probBL \quad \forall s \in S.\ \langle\!| C', s |\!\rangle \succ \quad B \equiv_L B}{\mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od} \hookrightarrow \mathsf{while}\ B\ \mathsf{do}\ C'\ \mathsf{od} \vdash probBL}$$

$$[\mathrm{PAR'}]\ \frac{V \hookrightarrow V' \vdash probBL \quad W \hookrightarrow W' \vdash probBL \quad V' \gtreqless W'}{VW \hookrightarrow V'W' \vdash probBL}$$

$$[\mathrm{FRK'}]\ \frac{\langle C \rangle W \hookrightarrow \langle C' \rangle W' \vdash probBL}{\mathsf{fork}(C, W) \hookrightarrow \mathsf{fork}(C', W') \vdash probBL}$$

Based on Theorem 4 we can show the following theorem:

Theorem 6.

$$V \hookrightarrow V' \vdash probBL \Longrightarrow V' \textit{ is } probBL \textit{ secure}$$

Proof. For rule [MIX_1] we know from the condition that C is probBL secure. Hence, each transformation to C leads to a probBL secure program. From [SNC] we know that for a probBL secure program C' the sequential execution of $C'; \mathsf{sync}$ is also probBL secure. Hence, if a transformation to C' leads to a probBL secure program then a transformation to $C'; \mathsf{sync}$ leads to a probBL secure program, too. With the same argumentation we can show that Theorem 6 also holds for the rest of the rules. □

Hence, we can add the following rule to the calculus without loosing soundness:

$$[\mathrm{MIX}_2]\ \frac{V \hookrightarrow V' \vdash probBL}{V' \vdash probBL}$$

This allows us to support the programmer during the programming process by transforming some probBL insecure programs to probBL secure programs. Moreover, if a transformation of a program leads to the same program then the integration of the transforming type system provides a first possibility for an automated security check.

Example 9. For a program like $\mathsf{fork}(\mathsf{l}_1{:=}1\ \langle \mathsf{l}_1{:=}\mathsf{l}_2\rangle)$ we can now show probBL security due to the typability according to the type system of Sabelfeld and Sands [SS00]. More precisely we show that $\mathsf{fork}(\mathsf{l}_1{:=}1\ \langle \mathsf{l}_1{:=}\mathsf{l}_2\rangle)$ can be transformed into a probBL secure program that is the same as the original program:

$$\dfrac{\dfrac{\dfrac{\cdots}{\mathsf{fork}(\mathsf{l}_1{:=}1\ \langle \mathsf{l}_1{:=}\mathsf{l}_2\rangle) \hookrightarrow \mathsf{fork}(\mathsf{l}_1{:=}1\ \langle \mathsf{l}_1{:=}\mathsf{l}_2\rangle) : \mathsf{fork}(\mathsf{l}_1{:=}1\ \langle \mathsf{l}_1{:=}\mathsf{l}_2\rangle)}\ [\mathrm{SS00}]}{\mathsf{fork}(\mathsf{l}_1{:=}1\ \langle \mathsf{l}_1{:=}\mathsf{l}_2\rangle) \hookrightarrow \mathsf{fork}(\mathsf{l}_1{:=}1\ \langle \mathsf{l}_1{:=}\mathsf{l}_2\rangle) \vdash probBL}\ \mathrm{T}_{SS}}{\mathsf{fork}(\mathsf{l}_1{:=}1\ \langle \mathsf{l}_1{:=}\mathsf{l}_2\rangle) \vdash probBL}\ \mathrm{MIX}_2$$

◊

6.2 Boudol and Castellani's Security Type System

The type system of [SS00] restricts guards of while loops to be of type low. Since probBL security does not prohibit high guarded loops in general we want to integrate a type system that allows us to be more precise at the formal check of high guarded loops. Boudol and Castellani [BC01] provide such a type system. Their approach bases on the concept that low assignments must not follow sequentially after a high-guarded loop or a high conditional. Formally this is achieved by a security typing $\Gamma \vdash C : (\tau, \sigma)\ cmd$, where Γ is a context given by an assignment of variables to security levels in a security lattice. Here we consider only the two element security lattice consisting of *high* and *low*, but we will stick to Γ to preserve the notation used by Boudol and Castellani. Intuitively, a program C can be typed with $(\tau, \sigma)\ cmd$ in the context Γ, if τ is a lower bound of the security levels of variables occurring on the left hand side of an assignment in C and σ is an upper bound of the security levels of guards and conditionals in C. Figure 5 recalls the type system of [BC01] adapted to our language. The typing rules WHL_{BC} and IF_{BC} increase σ if the security label of the guard is higher than σ was before. The rule SEQ_{BC} allows the sequential composition $C_1; C_2$ only if assignments in C_2 change only variables with a higher security level than the highest guard in C_1. Hence no high loops or conditionals can influence a low assignment.

As for the low-deterministic security in Section 5.2 we cannot show the general case:

$$\forall V.(\Gamma \vdash V : (\tau, \sigma)cmd \Longrightarrow V\ is\ probBL\ secure)$$

Again, possibly terminating programs as $P = \mathsf{while}\ h\ \mathsf{do\ skip\ od}$ are the reason. Nevertheless, for deterministic terminating programs the implication holds.

Lemma 9. *Let V and V' be probBL secure programs such that*

$$V = \langle C_1 | .. | C_{i-1} | C_i | C_{i+1} | .. | C_n \rangle$$
$$V' = \langle C_1 | .. | C_{i-1} | C_i' | C_{i+1} | .. | C_n \rangle$$

$$\text{SKIP}_{BC}\ \frac{}{\Gamma \vdash \mathsf{skip} : (\tau,\sigma)cmd} \qquad \text{ASSIGN}_{BC}\ \frac{\Gamma \vdash e : \tau \quad \Gamma(x) = \tau}{\Gamma \vdash x := e : (\tau,\sigma)cmd}$$

$$\text{SEQ}_{BC}\ \frac{\Gamma \vdash C : (\tau,\sigma)cmd \quad \Gamma \vdash D : (\tau',\sigma')cmd \quad \sigma \leq \tau'}{\Gamma \vdash C; D : (\tau \sqcap \tau', \sigma \sqcup \sigma')cmd}$$

$$\text{IF}_{BC}\ \frac{\Gamma \vdash e : \theta \quad \Gamma \vdash C_i : (\tau,\sigma)cmd \quad \theta \leq \tau}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \mathsf{fi} : (\tau, \sigma \sqcup \theta)cmd}$$

$$\text{WHL}_{BC}\ \frac{\Gamma \vdash e : \theta \quad \Gamma \vdash C : (\tau,\sigma)cmd \quad \theta \sqcup \sigma \leq \tau}{\Gamma \vdash \mathsf{while}\ e\ \mathsf{do}\ C\ \mathsf{od} : (\tau, \sigma \sqcup \theta)cmd}$$

$$\text{PARA}_{BC}\ \frac{\forall i : 1..k : \Gamma \vdash C_i : (\tau,\sigma)cmd}{\Gamma \vdash \langle C_1|..|C_K\rangle : (\tau,\sigma)cmd} \qquad \text{FRK}_{BC}\ \frac{\Gamma \vdash \langle CV\rangle : (\tau,\sigma)cmd}{\Gamma \vdash \mathsf{fork}(C,V) : (\tau,\sigma)cmd}$$

$$\text{SUB}_{BC}\ \frac{\Gamma \vdash C : (\tau,\sigma)cmd \quad \tau' \leq \tau \quad \sigma \leq \sigma'}{\Gamma \vdash C : (\tau',\sigma')cmd}$$

Fig. 5. Adapted Typing Rules [BC01]

with

$$C_i = C_i'; D \wedge H(D)$$

and D deterministic terminating. Then we have

$$\begin{aligned}\forall s,t,r.s =_L t \Longrightarrow & \sum_{s' \in [r]_{=_L}} prob_{\langle\!|V,s|\!\rangle,\langle\!|\langle\rangle,s'|\!\rangle} \\ = & \sum_{t' \in [r]_{=_L}} prob_{\langle\!|V',t|\!\rangle,\langle\!|\langle\rangle,t'|\!\rangle}.\end{aligned}$$

Hence, a high program that is deterministic terminating can be attached sequentially to each thread without changing the probabilities for low-equal final states for the evaluation of the thread pool.

Proof. Assume Lemma 9 does not hold. Then the execution of D influences the probabilities for the final low-equivalence classes. This implies that D either changes the low part of the states directly or D influences the probabilities for the execution sequence of other threads that change the low part of the state. By Definition 17 D does not change the low part of the memory directly. Moreover, since we assume the uniform scheduler the probability for each pair of threads C_i and C_j $(i \neq j)$ is 1/2 that C_i is executed before C_j and 1/2 for the other way round. This is independent of the number of threads and the nature of the threads. Moreover, since D is free of low affectings and V as well as V' are probBL secure, assignments in D have no influence on the probabilities for the final low-equivalence classes. Hence, D introduces just stuttering steps but has no influence on the probabilities of the execution sequences of the other threads. □

Definition 22. *A program V is a* ***simple program*** *(denoted by simp(V)) iff V fulfils one of the following conditions:*

– V=$\langle \mathsf{skip} \rangle$

- $V = \langle x := e \rangle \wedge \Gamma \vdash e : \tau \wedge \Gamma(x) \geq \tau$
- $V = \langle C_1; C_2 \rangle \wedge simp(C_1) \wedge simp(C_2)$
- $V = \langle \textsf{if } B \textsf{ then } C_1 \textsf{ else } C_2 \textsf{ fi} \rangle \wedge simp(C_1) \wedge simp(C_2) \wedge \Gamma \vdash B : low$
- $V = \langle \textsf{while } B \textsf{ do } C \textsf{ od} \rangle \wedge simp(C) \wedge \Gamma \vdash B : low$
- $V = \textsf{fork}(C, V) \wedge simp(C) \wedge simp(V)$
- $V = \langle V_1 V_2 \rangle \wedge simp(V_1) \wedge simp(V_2)$

Hence, a simple program is a program without high conditionals, high guarded loops, or assignments from high to low.

Lemma 10. *Let V be a program vector then the following holds:*

$$simp(V) \Longrightarrow V \vdash probBLsecure$$

Proof. The only possibility where high variables might have an influence are assignments to high variables. Hence, two low equal starting states will produce decision trees that differ at most on their high parts of the states. This implies that the sum over all probabilities for each equivalence class is the same for both starting states. Hence, simple programs are probBL secure. □

Theorem 7. *For all deterministic terminating and all deterministic non-terminating programs V the following holds:*

$$\forall V.(\Gamma \vdash V : (\tau, \sigma)cmd \Longrightarrow V \textit{ is probBL secure})$$

Proof. All deterministic non-terminating programs are probBL secure due to Lemma 4. Hence, we have only to show that the theorem also holds for deterministic terminating programs: After a high conditional or a high-guarded loop no low assignment is allowed by the typing rules. Hence, as soon as a high conditional or a high-guarded loop occurs syntactically in a thread we can be sure that the rest of the thread is free of low assignments. Since we know that V is deterministic terminating we have that all threads are deterministic terminating. Hence, by Lemma 9 we can remove those high programs without changing the results for the sums over the probabilities for low-equivalence classes. Moreover, removing all such high programs will result in a simple program. By Lemma 10 simple programs are probBL secure. Hence, the probabilities for the final equivalence classes sum up in the right way. □

Since Boudol and Castellani assume an attacker who can observe the low part of the memory at any time some probBL secure programs are not typable in [BC01] as the following example shows:

Example 10. Let

$$P_1 = \textsf{if } h \textsf{ then } l := 1 \textsf{ else } l := 1 \textsf{ fi } fi \quad \text{and}$$
$$P_2 = \textsf{while } h > 0 \textsf{ do } h := h - 1 \textsf{ od}; \ l := 1.$$

Both programs are probBL secure. Nevertheless, the high conditions lift the rest of the program into a high context where no low assignment is allowed by the typing rules.

◊

The integration of the type system of [BC01] allows us the check of programs containing loops with high guards:

Example 11. Let

$$P = h := 10; \textsf{while } h > 0 \textsf{ do } h := h - 1 \textsf{ od}.$$

Due to the high guard of the loop we cannot check the program with the type system by Sabelfeld and Sands, but we can check it with the type system by Boudol and Castellani.

7 Integration to Possibilistic Low Security [MSK07]

In [MSK07] we have introduced a plugin calculus that supports a possibilistic based security definition. We will now show, that we can use probBL security as a plugin for this calculus.

Definition 23. *[MSK07] A symmetric relation R on command vectors is a possibilistic low indistinguishability iff for all programs V, W with $V\ R\ W$ holds:*

$$\forall s, s', t.((s =_L t \wedge \langle\!| V, s |\!\rangle \rightarrow^* \langle\!| \langle\rangle, s' |\!\rangle) \Longrightarrow \exists t'.(\langle\!| W, t |\!\rangle \rightarrow^* \langle\!| \langle\rangle, t' |\!\rangle \wedge s' =_L t'))$$

The union of all such relations R is denoted by $\sim_L$.

Definition 24. *A program V is possibilistic low secure iff $V \sim_L V$.*

Theorem 8. *$\forall V$. (V is probBL secure $\Longrightarrow$ V is possibilistic low secure)*

Proof. We show $\forall V.(V \sim_L^p V \Longrightarrow V \sim_L V)$. By Definition 14 and Definition 23 Theorem 8 follows instantly.

Assume states s, s', t such that

$$s =_L t \ \wedge prob_{\langle\!| V,s |\!\rangle, \langle\!| \langle\rangle, s' |\!\rangle} > 0.$$

From this we have that the following holds:

$$\sum_{s'' \in [s']_{=_L}} prob_{\langle\!| V,s |\!\rangle, \langle\!| \langle\rangle, s'' |\!\rangle} > 0$$

From this, the low-equality of s and t and Definitions 13 and 14 we have

$$\sum_{t' \in [s']_{=_L}} prob_{\langle\!| V,t |\!\rangle, \langle\!| \langle\rangle, t' |\!\rangle} > 0.$$

Hence, there exists a state t' such that the following holds:

$$prob_{\langle\!| V,t |\!\rangle, \langle\!| \langle\rangle, t' |\!\rangle} > 0 \wedge s' =_L t'$$

Finally, we have

$$\forall s, s', t.((s =_L t \wedge \quad \langle\!| V, s |\!\rangle \rightarrow^* \langle\!| \langle\rangle, s' |\!\rangle) \quad \Longrightarrow \\ (\exists t'.\ \langle\!| V, s |\!\rangle \rightarrow^* \langle\!| \langle\rangle, t' |\!\rangle \wedge s' =_L t')).$$

On the other hand, if we assume a combination of states s, t, s' such that

$$s \neq_L t \ \vee prob_{\langle\!| V,s |\!\rangle, \langle\!| \langle\rangle, s' |\!\rangle} \leq 0$$

then we have $V \sim_L V$ independently of the probBL security of V. □

Again we use an example to show that the set of probBL secure programs is a proper subset of possibilistic low-secure programs.

Example 12. Let C be the following program:

$$\mathsf{if}\ h\ \mathsf{then}\ \mathsf{fork}(l := 1, \langle l := 0 | l := 1\rangle)\ \mathsf{else}\ \mathsf{fork}(l := 0, \langle l := 0\rangle)\ \mathsf{fi}$$

We have two possible final low-equivalence classes $[s[l = 0]]_{=_L}$ and $[s[l = 1]]_{=_L}$. Both can be reached independently of the initial value of h. Hence, C is possibilistic low secure. Nevertheless, the probabilities to reach them differ for the low-equal starting states $s_1 = [h = \mathsf{True}]$ and $s_1 = [h = \mathsf{False}]$:

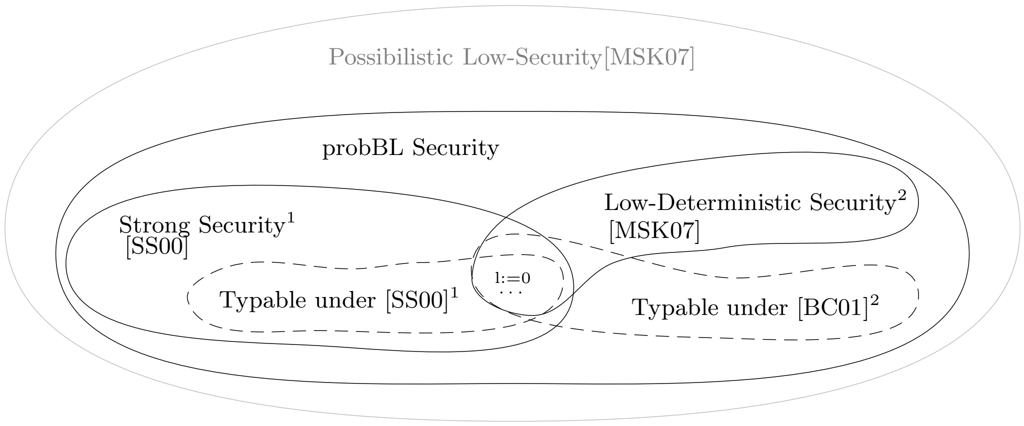
$$\sum_{s' \in [s[l=1]]_{=_L}} prob_{\langle\!|C,s_1|\!\rangle,\langle\!|\langle\rangle,s'|\!\rangle} = \tfrac{2}{3}$$
$$\sum_{s' \in [s[l=1]]_{=_L}} prob_{\langle\!|C,s_2|\!\rangle,\langle\!|\langle\rangle,s'|\!\rangle} = \tfrac{1}{2}$$

◊

We can now add the following novel plugin rule to the combining calculus of [MSK07] where $V \vdash bls$ means that V is possibilistic low secure:

$$[\mathrm{P}_{prob}]\ \frac{V \vdash probBL}{V \vdash bls}$$

In summary, we have given a probabilistic justification for the framework from [MSK07] where one can flexibly chose the security definition under which a program should be checked. The security under a less restrictive security definition can then be derived instantly.



¹ constrained to sync-free programs
² constrained to deterministic terminating and deterministic non-terminating programs

Fig. 6. Hierarchy of Security Definitions

8 Conclusion and Future Work

We have presented a probabilistic justification of the plugin approach of [MSK07].

With this work the basis is given for research in several directions. The development of a scheduler-independent justification of the baseline policy and an appropriate integration would lead to more flexibility in choosing a runtime environment, where one is currently limited to two schedulers: the possibilistic scheduler and the uniform scheduler. On the other hand the integration of further plugin rules might lead to an even higher precision of the analysis.

The setting assumed by such a characterization is appropriate for programs running internally on devices and that have no intermediate output on observable channels. Programs that produce intermediate output such as messages on the screen or network communication would need a slightly different characterization of security.

Another interesting research topic might be the integration of probabilistic non-interference [Smi03] into our framework. Like the type system of [BC01], the approach of [Smi03] allows us to use high-guarded loops as well as flexible high conditionals. This can be done due to the following restriction:

- it is not allowed to use assignments to low variables in or sequentially after a high-guarded loop or a high condition that has different timing-behavior for both branches

Smith uses the protect statement that allows him to treat program blocks as atomic computations. The use of protect relaxes the restrictions for high-conditionals, but also reduces concurrency. Another difference between [Smi03] and our approach is that he uses a termination-insensitive security characterization. Due to this the following program can be typed as secure under [Smi03]:

$$P \;=\; \mathsf{fork}(l := 0, \langle \mathsf{while}\ h\ \mathsf{do\ skip\ od}\ ,\ l := 1\rangle))$$

A uniform scheduler has the probability $\frac{1}{2}$ to set l:=0 and then l:=1 as well as for the other way round. This probability is independent of the initial value of h. Nevertheless, the termination probability depends on the initial value of h. The termination of P implies that h was False. Hence, the integration of [Smi03] appears straightforward only for deterministic terminating and deterministic non-terminating programs. An extension for the analysis of possibly terminating programs would be a more interesting topic for the future.

Finally the use of Markov-Chains instead of probability trees would integrate a well studied mathematical background into the theory. However, we cannot directly use the approach of [Smi03] that bases on the idea to take configurations as states. The proof of the compositionality result is one example where this approach would fail.

References

BC01. G. Boudol and I. Castellani. Noninterference for Concurrent Programs. *Lecture Notes in Computer Science*, 2076:382+, 2001.

MSK07. Heiko Mantel, Henning Sudbrock, and Tina Kraußer. Combining different proof techniques for verifying information flow security. In German Puebla, editor, *16th International Symposium on Logic Based Program Synthesis and Transformation, LOPSTR 2006*, volume 4407 of *LNCS*. Springer, 2007.

Sab03. A. Sabelfeld. Confidentiality for Multithreaded Programs via Bisimulation. In *Proceedings of Andrei Ershov 5th International Conference on Perspectives of System Informatics*, number 2890 in LNCS, pages 260–274, 2003.

SM03. A. Sabelfeld and A. C. Myers. Language-based Information-Flow Security. *IEEE Journal on Selected Areas in Communication*, 21(1):5–19, 2003.

Smi03. G. Smith. Probabilistic Noninterference through Weak Probabilistic Bisimulation. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop*, pages 3–13, Pacific Grove, California, USA, 2003.

SS00. A. Sabelfeld and D. Sands. Probabilistic Noninterference for Multi-threaded Programs. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 200–215, Cambridge, UK, 2000.

SV98. G. Smith and D. Volpano. Secure Information Flow in a Multi-threaded Imperative Language. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages*, pages 355–364, San Diego, California, 1998.

VS98. D. Volpano and G. Smith. Probabilistic Noninterference in a Concurrent Language. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, pages 34–43, Rockport, Massachusetts, 1998.

VSI96. D. Volpano, G. Smith, and C. Irvine. A Sound Type System for Secure Flow Analysis. *Journal of Computer Security*, 4(3):1–21, 1996.

ZM03. S. Zdancewic and A. C. Myers. Observational Determinism for Concurrent Program Security. In *CSFW*, pages 29–, 2003.

Aachener Informatik-Berichte

This list contains all technical reports published during the past five years. A complete list of reports dating back to 1987 is available from http://aib.informatik.rwth-aachen.de/. To obtain copies consult the above URL or send your request to: Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Email: `biblio@informatik.rwth-aachen.de`

2001-01 *	Jahresbericht 2000
2001-02	Benedikt Bollig, Martin Leucker: Deciding LTL over Mazurkiewicz Traces
2001-03	Thierry Cachat: The power of one-letter rational languages
2001-04	Benedikt Bollig, Martin Leucker, Michael Weber: Local Parallel Model Checking for the Alternation Free mu-Calculus
2001-05	Benedikt Bollig, Martin Leucker, Thomas Noll: Regular MSC Languages
2001-06	Achim Blumensath: Prefix-Recognisable Graphs and Monadic Second-Order Logic
2001-07	Martin Grohe, Stefan Wöhrle: An Existential Locality Theorem
2001-08	Mareike Schoop, James Taylor (eds.): Proceedings of the Sixth International Workshop on the Language-Action Perspective on Communication Modelling
2001-09	Thomas Arts, Jürgen Giesl: A collection of examples for termination of term rewriting using dependency pairs
2001-10	Achim Blumensath: Axiomatising Tree-interpretable Structures
2001-11	Klaus Indermark, Thomas Noll (eds.): Kolloquium Programmiersprachen und Grundlagen der Programmierung
2002-01 *	Jahresbericht 2001
2002-02	Jürgen Giesl, Aart Middeldorp: Transformation Techniques for Context-Sensitive Rewrite Systems
2002-03	Benedikt Bollig, Martin Leucker, Thomas Noll: Generalised Regular MSC Languages
2002-04	Jürgen Giesl, Aart Middeldorp: Innermost Termination of Context-Sensitive Rewriting
2002-05	Horst Lichter, Thomas von der Maßen, Thomas Weiler: Modelling Requirements and Architectures for Software Product Lines
2002-06	Henry N. Adorna: 3-Party Message Complexity is Better than 2-Party Ones for Proving Lower Bounds on the Size of Minimal Nondeterministic Finite Automata
2002-07	Jörg Dahmen: Invariant Image Object Recognition using Gaussian Mixture Densities
2002-08	Markus Mohnen: An Open Framework for Data-Flow Analysis in Java
2002-09	Markus Mohnen: Interfaces with Default Implementations in Java
2002-10	Martin Leucker: Logics for Mazurkiewicz traces
2002-11	Jürgen Giesl, Hans Zantema: Liveness in Rewriting
2003-01 *	Jahresbericht 2002
2003-02	Jürgen Giesl, René Thiemann: Size-Change Termination for Term Rewriting
2003-03	Jürgen Giesl, Deepak Kapur: Deciding Inductive Validity of Equations

2003-04 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke: Improving Dependency Pairs
2003-05 Christof Löding, Philipp Rohde: Solving the Sabotage Game is PSPACE-hard
2003-06 Franz Josef Och: Statistical Machine Translation: From Single-Word Models to Alignment Templates
2003-07 Horst Lichter, Thomas von der Maßen, Alexander Nyßen, Thomas Weiler: Vergleich von Ansätzen zur Feature Modellierung bei der Softwareproduktlinienentwicklung
2003-08 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke: Mechanizing Dependency Pairs
2004-01 * Fachgruppe Informatik: Jahresbericht 2003
2004-02 Benedikt Bollig, Martin Leucker: Message-Passing Automata are expressively equivalent to EMSO logic
2004-03 Delia Kesner, Femke van Raamsdonk, Joe Wells (eds.): HOR 2004 – 2nd International Workshop on Higher-Order Rewriting
2004-04 Slim Abdennadher, Christophe Ringeissen (eds.): RULE 04 – Fifth International Workshop on Rule-Based Programming
2004-05 Herbert Kuchen (ed.): WFLP 04 – 13th International Workshop on Functional and (Constraint) Logic Programming
2004-06 Sergio Antoy, Yoshihito Toyama (eds.): WRS 04 – 4th International Workshop on Reduction Strategies in Rewriting and Programming
2004-07 Michael Codish, Aart Middeldorp (eds.): WST 04 – 7th International Workshop on Termination
2004-08 Klaus Indermark, Thomas Noll: Algebraic Correctness Proofs for Compiling Recursive Function Definitions with Strictness Information
2004-09 Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Parameterized Power Domination Complexity
2004-10 Zinaida Benenson, Felix C. Gärtner, Dogan Kesdogan: Secure Multi-Party Computation with Security Modules
2005-01 * Fachgruppe Informatik: Jahresbericht 2004
2005-02 Maximillian Dornseif, Felix C. Gärtner, Thorsten Holz, Martin Mink: An Offensive Approach to Teaching Information Security: "Aachen Summer School Applied IT Security"
2005-03 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp: Proving and Disproving Termination of Higher-Order Functions
2005-04 Daniel Mölle, Stefan Richter, Peter Rossmanith: A Faster Algorithm for the Steiner Tree Problem
2005-05 Fabien Pouget, Thorsten Holz: A Pointillist Approach for Comparing Honeypots
2005-06 Simon Fischer, Berthold Vöcking: Adaptive Routing with Stale Information
2005-07 Felix C. Freiling, Thorsten Holz, Georg Wicherski: Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks
2005-08 Joachim Kneis, Peter Rossmanith: A New Satisfiability Algorithm With Applications To Max-Cut

2005-09 Klaus Kursawe, Felix C. Freiling: Byzantine Fault Tolerance on General Hybrid Adversary Structures

2005-10 Benedikt Bollig: Automata and Logics for Message Sequence Charts

2005-11 Simon Fischer, Berthold Vöcking: A Counterexample to the Fully Mixed Nash Equilibrium Conjecture

2005-12 Neeraj Mittal, Felix Freiling, S. Venkatesan, Lucia Draque Penso: Efficient Reductions for Wait-Free Termination Detection in Faulty Distributed Systems

2005-13 Carole Delporte-Gallet, Hugues Fauconnier, Felix C. Freiling: Revisiting Failure Detection and Consensus in Omission Failure Environments

2005-14 Felix C. Freiling, Sukumar Ghosh: Code Stabilization

2005-15 Uwe Naumann: The Complexity of Derivative Computation

2005-16 Uwe Naumann: Syntax-Directed Derivative Code (Part I: Tangent-Linear Code)

2005-17 Uwe Naumann: Syntax-directed Derivative Code (Part II: Intraprocedural Adjoint Code)

2005-18 Thomas von der Maßen, Klaus Müller, John MacGregor, Eva Geisberger, Jörg Dörr, Frank Houdek, Harbhajan Singh, Holger Wußmann, Hans-Veit Bacher, Barbara Paech: Einsatz von Features im Software-Entwicklungsprozess - Abschlußbericht des GI-Arbeitskreises "Features"

2005-19 Uwe Naumann, Andre Vehreschild: Tangent-Linear Code by Augmented LL-Parsers

2005-20 Felix C. Freiling, Martin Mink: Bericht über den Workshop zur Ausbildung im Bereich IT-Sicherheit Hochschulausbildung, berufliche Weiterbildung, Zertifizierung von Ausbildungsangeboten am 11. und 12. August 2005 in Köln organisiert von RWTH Aachen in Kooperation mit BITKOM, BSI, DLR und Gesellschaft fuer Informatik (GI) e.V.

2005-21 Thomas Noll, Stefan Rieger: Optimization of Straight-Line Code Revisited

2005-22 Felix Freiling, Maurice Herlihy, Lucia Draque Penso: Optimal Randomized Fair Exchange with Secret Shared Coins

2005-23 Heiner Ackermann, Alantha Newman, Heiko Röglin, Berthold Vöcking: Decision Making Based on Approximate and Smoothed Pareto Curves

2005-24 Alexander Becher, Zinaida Benenson, Maximillian Dornseif: Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks

2006-01 * Fachgruppe Informatik: Jahresbericht 2005

2006-02 Michael Weber: Parallel Algorithms for Verification of Large Systems

2006-03 Michael Maier, Uwe Naumann: Intraprocedural Adjoint Code Generated by the Differentiation-Enabled NAGWare Fortran Compiler

2006-04 Ebadollah Varnik, Uwe Naumann, Andrew Lyons: Toward Low Static Memory Jacobian Accumulation

2006-05 Uwe Naumann, Jean Utke, Patrick Heimbach, Chris Hill, Derya Ozyurt, Carl Wunsch, Mike Fagan, Nathan Tallent, Michelle Strout: Adjoint Code by Source Transformation with OpenAD/F

2006-06 Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Divide-and-Color

2006-07 Thomas Colcombet, Christof Löding: Transforming structures by set interpretations

2006-08 Uwe Naumann, Yuxiao Hu: Optimal Vertex Elimination in Single-Expression-Use Graphs

2006-09 Tingting Han, Joost-Pieter Katoen: Counterexamples in Probabilistic Model Checking

2006-10 Mesut Günes, Alexander Zimmermann, Martin Wenig, Jan Ritzerfeld, Ulrich Meis: From Simulations to Testbeds - Architecture of the Hybrid MCG-Mesh Testbed

2006-11 Bastian Schlich, Michael Rohrbach, Michael Weber, Stefan Kowalewski: Model Checking Software for Microcontrollers

2006-12 Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker: Replaying Play in and Play out: Synthesis of Design Models from Scenarios by Learning

2006-13 Wong Karianto, Christof Löding: Unranked Tree Automata with Sibling Equalities and Disequalities

2006-14 Danilo Beuche, Andreas Birk, Heinrich Dreier, Andreas Fleischmann, Heidi Galle, Gerald Heller, Dirk Janzen, Isabel John, Ramin Tavakoli Kolagari, Thomas von der Maßen, Andreas Wolfram: Report of the GI Work Group "Requirements Management Tools for Product Line Engineering"

2006-15 Sebastian Ullrich, Jakob T. Valvoda, Torsten Kuhlen: Utilizing optical sensors from mice for new input devices

2006-16 Rafael Ballagas, Jan Borchers: Selexels: a Conceptual Framework for Pointing Devices with Low Expressiveness

2006-17 Eric Lee, Henning Kiel, Jan Borchers: Scrolling Through Time: Improving Interfaces for Searching and Navigating Continuous Audio Timelines

2007-01 * Fachgruppe Informatik: Jahresbericht 2006

2007-02 Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl: SAT Solving for Termination Analysis with Polynomial Interpretations

2007-03 Jürgen Giesl, René Thiemann, Stephan Swiderski, and Peter Schneider-Kamp: Proving Termination by Bounded Increase

2007-04 Jan Buchholz, Eric Lee, Jonathan Klein, Jan Borchers: coJIVE: A System to Support Collaborative Jazz Improvisation

2007-05 Uwe Naumann: On Optimal DAG Reversal

2007-06 Joost-Pieter Katoen, Thomas Noll, and Stefan Rieger: Verifying Concurrent List-Manipulating Programs by LTL Model Checking

2007-07 Alexander Nyßen, Horst Lichter: MeDUSA - MethoD for UML2-based Design of Embedded Software Applications

* These reports are only available as a printed version.
Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.