# Proofs of Theorems from the Article "Flexible Scheduler-Independent Security"

Heiko Mantel and Henning Sudbrock

Computer Science, TU Darmstadt, Germany,
{mantel,sudbrock}@cs.tu-darmstadt.de

This document contains proofs for the theorems from the article "Flexible Scheduler-Independent Security" [MS10].

## 1 Proof of Theorem 1 (Low Matches)

**Theorem 1.** *The low match of $thr_1$ and $thr_2$ is unique and given by the function*

$$l\text{-}match_{thr_1,thr_2}(k_1) = \min\Big\{k_2 \in \mathbb{N}_0 \mid \big|\{l_1 \leq k_1 \mid thr_1(l_1) \in LCom\}\big| = \big|\{l_2 \leq k_2 \mid thr_2(l_2) \in LCom\}\big|\Big\}.$$

*Proof.*

1. Assume thread pools $thr_1$ and $thr_2$ containing the same number of low threads.
2. As a low match is an order-preserving bijection on finite sets, it maps the $n$th value in its domain to the $n$th value in its range for each $0 < n \leq N$, where $N$ is the finite size of domain and range.
3. The number $k_1$ with $thr_1(k_1) \in LCom$ is the $n$th value of the domain $\{j \mid thr_1(j) \in LCom\}$, if $n = \big|\{l_1 \leq k_1 \mid thr_1(l_1) \in LCom\}\big|$. (For instance, the position $k_1$ is the 1st value in the domain, if there is only 1 low thread, namely $thr(k_1)$ itself, with a position less or equal than $k_1$ contained in $thr$.)
4. The $n$th value of the range $\{j \mid thr_2(j) \in LCom\}$ is given by $\min\{k_2 \in \mathbb{N}_0 \mid n = \big|\{l_2 \leq k_2 \mid thr_2(l_2) \in LCom\}\big|\}$.
5. Thus we obtain

$$l\text{-}match_{thr_1,thr_2}(k_1) = \min\Big\{k_2 \in \mathbb{N}_0 \mid \big|\{l_1 \leq k_1 \mid thr_1(l_1) \in LCom\}\big| = \big|\{l_2 \leq k_2 \mid thr_2(l_2) \in LCom\}\big|\Big\}.$$

□

## 2 Proof of Theorem 2 (Compositionality of FSI-Security)

**Theorem 2.** *Let $thr_1$ and $thr_2$ be FSI-secure thread pools. Then their* parallel composition *$par(thr_1, thr_2)$ is also FSI-secure, where*

$$par(thr_1, thr_2)(k) = \begin{cases} thr_1(k) & , \text{ if } k < \sharp(thr_1) \\ thr_2(k - \sharp(thr_1)) & , \text{ otherwise.} \end{cases}$$

Before proving the compositionality of FSI-security, we establish the following lemma:

**Lemma 1.** *The relation $\sim$ is a low bisimulation modulo low matching.*

*Proof.* Let $thr_1 \sim thr_2$. Then, by the definition of $\sim$, there exists a low bisimulation modulo low matching $R$ with $thr_1\ R\ thr_2$. Execution steps of $thr_1$ can hence be simulated by $thr_2$ according to the requirements of low bisimulations modulo low matching, resulting in thread pools related by $R$, and, hence, also related by $\sim$. □

Now we prove Theorem 2:

*Proof.*

1. We define the symmetric relation $R$ on thread pools with equal numbers of low threads as follows: $thr\ R\ thr'$ if and only if there exist thread pools $thr_1, thr_2, thr_1', thr_2'$ such that $thr_1 \sim thr_1'$, $thr_2 \sim thr_2'$, $thr = par(thr_1, thr_2)$, and $thr' = par(thr_1', thr_2')$.
2. We prove that $R$ is a low bisimulation modulo matching:
   (a) Let $thr\ R\ thr'$, and $thr_1, thr_2, thr_1', thr_2'$ with the properties as defined in step 1 of this proof.
   (b) Assume $l\text{-}match_{thr,thr'}(j) = k$. If $j < \sharp(thr_1)$, then $l\text{-}match_{thr_1,thr_1'}(j) = k$. If $\sharp(thr_1) \leq j < \sharp(thr_1) + \sharp(thr_2)$, then $l\text{-}match_{thr_2,thr_2'}(j - \sharp(thr_1)) = k - \sharp(thr_1')$. This is due to the fact that $thr_1$ and $thr_1'$ respectively $thr_2$ and $thr_2'$ are related by $\sim$ and hence contain the same number of low threads.
   (c) Let $mem, mem' \in Mem$ with $mem =_L mem'$.
   (d) Assume that $j < \sharp(thr_1)$ and that $thr(j) \in LCom$. Assume that $thr$ performs an execution step in $mem$ with the thread at position $j$, resulting in the thread pool $thr''$ and the memory $mem''$. Let $thr_1''$ be the thread pool resulting from performing an execution step in the $j$th thread of $thr_1$ in memory $mem$. Then $thr'' = par(thr_1'', thr_2)$.
   (e) As $thr_1 \sim thr_1'$, the execution step of $thr_1$ can be simulated by an execution step of the $k$th thread of $thr_1'$ in $mem'$ (where $k = l\text{-}match_{thr_1,thr_1'}(j)$), resulting in a thread pool $thr_1'''$ with $thr_1'' \sim thr_1'''$ and a memory $mem'''$ which is low-equal to $mem''$.
   (f) But then the execution step of the $j$th thread in $thr$ in the memory $mem$ can be simulated by an execution step of the $k$th thread $thr'$ in $mem'$ (where $l\text{-}match_{thr,thr'}(j) = k$ by (b)), resulting in the thread pool $thr''' = par(thr_1''', thr_2')$ and the memory $mem''' =_L mem''$.
   (g) Moreover, $thr''\ R\ thr'''$, as $thr'' = par(thr_1'', thr_2)$, $thr''' = par(thr_1''', thr_2')$, and $thr_1'' \sim thr_1'''$.
   (h) Now assume that $j < \sharp(thr_1)$ and $thr(j) \in HCom$, and that $thr$ performs an execution step in $mem$ with the thread at position $j$, resulting in the thread pool $thr''$. Then, as above, this corresponds to an execution step of $thr_1$, resulting in a thread pool $thr_1'' \sim thr_1'$, with $thr'' = par(thr_1'', thr_2)$.
   (i) In consequence, $thr''\ R\ thr'$.

(j) The case for $\sharp(thr_1) \leq j < \sharp(thr_2)$ is proved analogously.
(k) Hence, $R$ is a low bisimulation modulo low matching.

3. Now assume two FSI-secure thread pools $thr_1$ and $thr_2$. By definition, $thr_1 \sim thr_1$ and $thr_2 \sim thr_2$. Then $par(thr_1, thr_2)\ R\ par(thr_1, thr_2)$ by the definition of $R$. Hence,

$$par(thr_1, thr_2) \sim par(thr_1, thr_2),$$

as $R$ is a low bisimulation modulo low matching. Thus, $par(thr_1, thr_2)$ is FSI-secure.

□

## 3 Proof of Theorem 3 (High threads are FSI-secure)

**Theorem 3.** *Let $com \in HCom$. Then $com$ is FSI-secure.*

*Proof.*

1. We define the relation $R$ on thread pools with equal numbers of low threads by $thr_1\ R\ thr_2$ if and only if for all $0 \leq j < \sharp(thr_1)$ and all $0 \leq k < \sharp(thr_2)$ we have $thr_1(j) \in HCom$ and $thr_2(k) \in HCom$.
2. Then $R$ is a low bisimulation modulo low matching. (Assume $thr_1\ R\ thr_2$. As all threads within $thr_1$ and $thr_2$ are high threads, only the case for transitions of high threads has to be considered. As an execution step of a high thread does not spawn low threads by definition, the thread pool resulting after a transition of $thr_1$ again consists only of high threads, and is, hence, also related to $thr_2$ by R.)
3. The thread pool consisting of a single high command is related to itself by $R$, and hence FSI-secure.

□

## 4 Proof of Theorem 4 (Thread Purge Function)

**Theorem 4.** *For a thread pool $thr$, $th\text{-}purge(thr)$ contains no high threads and as many low threads as $thr$. Moreover, if $k \in \mathbb{N}_0$ with $thr(k) \in LCom$ then*

$$thr(k) = th\text{-}purge(thr)(l\text{-}match_{thr,th\text{-}purge(thr)}(k)).$$

*Proof.*

1. Let $thr$ be a thread pool. We firstly show that $th\text{-}purge(thr)$ does not contain any high threads.
   - Let $0 \leq j < \sharp(thr)$. Assume that $l = \min\{k \in \mathbb{N}_0 \mid j = |\{k' < k \mid thr(k') \in LCom \cup \{\mathsf{stop}\}\}|\}$ is the position of a high thread in $thr$ (i.e., $thr(l) \in HCom$). But then $l$ cannot be the minimal value for $k$ with $j = |\{k' < k \mid thr(k') \in LCom \cup \{\mathsf{stop}\}\}|$, as $j = |\{k' < l - 1 \mid thr(k') \in LCom \cup \{\mathsf{stop}\}\}|$ also holds. Hence $thr(l) \in HCom$ cannot hold.
2. We now show that $thr$ and $th\text{-}purge(thr)$ contain equally many low threads.

- Let $l$ be the position of the $n$th low thread in $thr$. Then $l = \min\big\{k \in \mathbb{N}_0 \mid n = \big|\{k' < k \mid thr(k') \in LCom \cup \{\mathsf{stop}\}\}\big|\big\}$. In consequence, the low thread at position $l$ is contained in $th\text{-}purge(thr)$ at position $n$ (i.e., $thr(l) = th\text{-}purge(thr)(n)$).
- Assume that $\big|\{j \mid thr(j) \in LCom\}\big| = n$, i.e., that $thr$ contains $n$ low threads. Then $th\text{-}purge(thr)(n) = \mathsf{stop}$, as $\big|\{k' < k \mid thr(k') \in LCom \cup \{\mathsf{stop}\}\}\big|$ only equals $n$ for $k \geq \sharp(thr)$.

3. We now show that $thr(j) = th\text{-}purge(thr)(l\text{-}match_{thr,th\text{-}purge(thr)}(j))$.
   - The low match maps the $n$th low thread in $thr$ to the $n$th low thread in $th\text{-}purge(thr)$ (compare Theorem 1).
   - Assume that $thr(j)$ is the $n$th low thread of $thr$. Then $j = \min\big\{k \in \mathbb{N}_0 \mid n = \big|\{k' < k \mid thr(k') \in LCom \cup \{\mathsf{stop}\}\}\big|\big\}$. That is, the low match maps $j$ to $n$.
   - But by definition, $th\text{-}purge(thr)(n)$ equals the thread in $thr$ at position $\min\big\{k \in \mathbb{N}_0 \mid n = \big|\{k' < k \mid thr(k') \in LCom \cup \{\mathsf{stop}\}\}\big|\big\}$, which is exactly the position $j$.

□

## 5 Proof of Theorems 5 and 6 (Robustness of Uniform and Round-Robin Scheduler)

**Theorem 5.** *The uniform scheduler (see Example 1 in [MS10]) is robust.*

**Theorem 6.** *The Round-Robin scheduler (see Example 2 in [MS10]) is robust.*

Before proving the robustness of the uniform and the Round-Robin scheduler, we prove the following auxiliary lemmas:

**Lemma 2.** *For each scheduler $\mathcal{S}$, the relation $<_{\mathcal{S}}$ is an $\mathcal{S}$-simulation.*

*Proof.* Let $conf_1 <_{\mathcal{S}} conf_2$. Then, by the definition of $<_{\mathcal{S}}$, there exists a $\mathcal{S}$-simulation $<$ with $conf_1 < conf_2$. Execution steps of $conf_1$ can hence be simulated by $conf_2$ according to the requirements of $\mathcal{S}$-simulations, resulting in configurations related by $<$, and, hence, also related by $<_{\mathcal{S}}$. □

**Lemma 3.** *Let $thr$ be an FSI-secure thread pool. Then $thr \sim th\text{-}purge(thr)$.*

*Proof.*

1. Define the relation $R'$ by $thr_1 \; R' \; thr_2$ if and only if

$$(thr_1 \sim thr_1) \;\wedge\; thr_2 = th\text{-}purge(thr_1),$$

   and let $R$ be the symmetric closure of $R'$.
2. We prove that $R$ is a low bisimulation modulo low matching:
   (a) A transition of a low thread in the thread pool $thr_1$ resulting in the thread pool $thr_1'$ can be simulated by a transition in the same thread pool where high threads have been removed (i.e., $th\text{-}purge(thr_1)$), resulting in the thread pool $th\text{-}purge(thr_1')$. Moreover, as $thr_1$ is FSI-secure, $thr_1'$ is also FSI-secure.

(b) A transition of a high thread in the thread pool $thr_1$ resulting in the thread pool $thr_1'$ does not spawn low threads. Hence, $\textit{th-purge}(thr_1') = \textit{th-purge}(thr_1)$ holds. Moreover, as $thr_1$ is FSI-secure, $thr_1'$ is also FSI-secure.

(c) A transition of a low thread in the thread pool $\textit{th-purge}(thr_1)$ results in the thread pool $\textit{th-purge}(thr_1')$, where $thr_1'$ is the thread pool resulting when the same low thread performs a transition in the thread pool $thr_1$.

3. If $thr$ is FSI-secure, then $thr \; R \; \textit{th-purge}(thr)$. As $R$ is a low bisimulation modulo low matching, we have $thr \sim \textit{th-purge}(thr)$.

□

Now we prove Theorems 5 and 6:

*Proof.*

1. Let us first show that the uniform scheduler from Example 1 is robust.
2. We define the relation $<$ on system configurations by $conf_1 < conf_2$ if and only if
   - $getT(conf_1) \sim getT(conf_1)$,
   - $getM(conf_1) =_L getM(aconf_1)$, and
   - $\textit{th-purge}(conf_1) = conf_2$.
3. We will show that $<$ is a uni-simulation.
   (a) Let $conf_1 < conf_2$ and $conf_1', j$, and $p$ such that $conf_1 \Rightarrow_{j,p} conf_1'$.
   (b) Assume firstly that $getT(conf_1)(j) \in HCom$.
      i. As $getT(conf_1) \sim getT(conf_1)$, we have $getT(conf_1) \sim getT(conf_1')$ and hence $getT(conf_1') \sim getT(conf_1')$ and furthermore $getM(conf_1') =_L getM(conf_1) =_L getM(conf_2)$.
      ii. As $getT(conf_1)(j) \in HCom$, the transition to $conf_1'$ does not spawn any low threads. Therefore $\textit{th-purge}(conf_1') = conf_2$.
      iii. Hence, $conf_1' < conf_2$.
   (c) Assume now that $getT(conf_1)(j) \in LCom$.
      i. Denote with $\#(high)$ the number of high threads in $conf_1$ and with $\#(low)$ the number of low threads in $conf_1$.
      ii. Hence, $p = 1/(\#(high) + \#(low))$ by the definition of the uniform scheduler.
      iii. Furthermore, $p/\textit{l-prob}_{\mathcal{S}}(conf_1) = p/(\#(low)/(\#(high) + \#(low)))$, which equals $1/\#(low)$.
      iv. As $getT(conf_1) \sim getT(conf_1)$, we have $getT(conf_1') \sim getT(conf_1')$.
      v. As $getT(conf_1) \sim getT(conf_1)$ and $\textit{th-purge}(conf_1) = conf_2$, we have $getT(conf_1) \sim getT(conf_2)$ by Lemma 3.
      vi. In consequence, the transition in $conf_1$ can be matched by the thread $getT(conf_2)(k)$ with some probability $q$, where

$$k = \textit{l-match}_{getT(conf_1),getT(conf_2)}(j),$$

      resulting in a configuration $conf_2'$ and a low-equal state. Moreover, $\textit{th-purge}(conf_2') = \textit{th-purge}(conf_1')$, as $conf_2 = \textit{th-purge}(conf_1)$.

vii. The probability of this transition equals $1/\#(low)$ by the definition of the uniform scheduler, which according to (iii) equals $p/l\text{-}prob_{\mathcal{S}}(conf_1)$.
viii. Furthermore, $conf'_1 < th\text{-}purge(conf'_2)$, due to (iv) and (vi).

(d) Hence, $<$ is a uni-simulation.

(e) As we have $\langle thr, mem, s\rangle < th\text{-}purge(\langle thr, mem, s\rangle)$ for each FSI-secure thread pool $thr$ and each memory $mem$, the uniform scheduler is robust.

(f) The proof for the Round-Robin scheduler goes along the same lines as the proof for the uniform scheduler, where we define an RR-simulation $<$ as follows: $conf_1 < conf_2$ if and only if
- $getT(conf_1) \sim getT(conf_1)$,
- $getM(conf_1) =_L getM(conf_1)$,
- $th\text{-}purge(conf_1) = conf_2$, and the following two requirements are satisfied:
- Let $getS(conf_1)(size) = n_1$ and $getS(conf_2)(size) = n_2$. Then $n_2$ equals $n_1$ minus the number of high threads in $conf_1$.
- Let $getS(conf_1)(choice) = c_1$ and $getS(conf_2)(choice) = c_2$. Then $c_2$ equals $c_1$ minus the number of high threads in $conf_1$ at a position below $c_1$.

Note that the first three requirements are exactly the requirements that we also required for the uni-simulation. The fourth requirement expresses that the thread pool size stored by the scheduler in the execution with high threads differs from the thread pool size stored in the execution without high threads exactly by the current number of high threads. The fifth requirement expresses that the last position chosen by the scheduler in the execution with high threads differs from the position chosen in the execution without high threads exactly by the number of high threads below that position.
The last two requirements are guaranteed to remain invariant during two executions with and without high threads, and ensure that if the $n$th low thread is selected by the scheduler in the execution with high threads, than the $n$th low thread is also selected by the scheduler in the execution without high threads.

□

## 6 Proof of Theorem 7 (Scheduler Independence)

**Theorem 7.** *Let thr be a terminating thread pool that is FSI-secure and let $\mathcal{S}$ be a robust scheduler under a confined observation function. Then the thread pool thr is $\mathcal{S}$-secure.*

For the remainder of this section, let $\mathcal{S} = (sSt, sst_0, sLab, \rightarrow)$ be a scheduler under a confined observation function.
For proving Theorem 7, we establish the following more general proposition:

**Proposition 1.** *Let* $conf_1 = \langle thr_1, mem_1, sst_1\rangle$ *and* $conf_2 = \langle thr_2, mem_2, sst_2\rangle$ *be terminating system configurations,* $mem_{1,p}, mem_{2,p}$ *memories, and* $sst_p$ *a scheduler state, such that*

- $thr_1 \sim thr_2$,
- $mem_1 =_L mem_2 =_L mem_{1,p} =_L mem_{2,p}$,
- $conf_1 <_{\mathcal{S}} \langle th\text{-}purge(thr_1), mem_{1,p}, sst_p\rangle$, *and*
- $conf_2 <_{\mathcal{S}} \langle th\text{-}purge(thr_2), mem_{2,p}, sst_p\rangle$.

*Then the following equality holds for all* $mem \in Mem$*:*

$$\sum_{mem' \in [mem]_{=_L}} \rho_{\mathcal{S}}(conf_1, mem') = \sum_{mem' \in [mem]_{=_L}} \rho_{\mathcal{S}}(conf_2, mem')$$

*Proof.* Let $\sharp(T) = \sum_{tr \in T} \sharp(tr)$ be the sum of the lengths of all traces in a set of traces $T$. We prove the proposition by induction on $n = \sharp(Tr_{\mathcal{S}}(conf_1)) + \sharp(Tr_{\mathcal{S}}(conf_2))$. Note that $n$ is finite as $thr_1$ as well as $thr_2$ are terminating. In the following, we abbreviate the sum $\sum_{mem' \in [mem]_{=_L}} \rho_{\mathcal{S}}(conf, mem')$ with $P(conf)$.

**Induction basis ($n = 0$):**
In this case, $thr_1$ and $thr_2$ have both terminated. Hence the equality is satisfied, as $mem_1 =_L mem_2$.

**Induction step ($n > 0$):**
Let us at first briefly elaborate on the proof idea of the rather lengthy proof of the induction step: We consider transitions of low and high threads in $conf_1$ and $conf_2$ separately. If $conf_1$ makes a transition in a high thread to $conf_1'$, we prove that $conf_1'$ and $conf_2$ together with $mem_1, mem_2, mem_{1,p}, mem_{2,p}$, and $sst_p$ satisfy all the preconditions of this proposition, and apply the induction hypothesis to relate the probabilities to reach a final memory low-equal to $mem$ from $conf_1'$ and $conf_2$. We proceed likewise for transitions of low threads, showing that $P(conf_1') = P(conf_2')$, where $conf_1'$ and $conf_2'$ are reached from $conf_1$ and $conf_2$ by transitions of low threads linked by $l\text{-}match_{thr_1, thr_2}$. This yields a system of two linear equations with unknowns $P(conf_1)$ and $P(conf_2)$. We show that the probabilities $P(conf_1)$ and $P(conf_2)$ must be equal to solve the equation system, which concludes the proof. The last proof step crucially depends on the relation on probabilities required by $\mathcal{S}$-simulations. We now prove the induction step:

1. For each system configuration $conf$, we denote with $enabledPos(conf) \subset \mathbb{N}_0$ the set of thread positions such that $j \in enabledPos(conf)$ if and only if there exist $conf' \in Conf$ and $p \in ]0, 1]$ with $conf \Rightarrow_{j,p} conf'$.
   By the definition of schedulers and the semantics for system configurations, $conf'$ and $p$ are uniquely determined by $conf$ and $j$. We denote those values with $succ_j(conf)$ respectively $\rho_j(conf)$.
2. By the definition of trace probabilities, we have

$$P(conf_1) = \sum \{\!\{\rho_j(conf_1) * P(succ_j(conf_1)) \mid j \in enabledPos(conf_1)\}\!\}.$$

3. We rewrite this sum as $P(conf_1) = P(conf_1, high) + P(conf_1, low)$, where

$$P(conf_1, high) = \sum \{\!| \rho_j(conf_1) * P(succ_j(conf_1)) \mid j \in enabledPos(conf_1) \ \wedge \ thr_1(j) \in HCom |\!\} \text{ and}$$

$$P(conf_1, low) = \sum \{\!| \rho_j(conf_1) * P(succ_j(conf_1)) \mid j \in enabledPos(conf_1) \ \wedge \ thr_1(j) \in LCom |\!\}.$$

4. Consider a position $j \in enabledPos(conf_1)$. Then there exist $com_1, mem_1'$, and $\alpha$ such that $\langle thr_1(j), mem_1\rangle \xrightarrow{\alpha} \langle com_1, mem_1'\rangle$. There furthermore exists $sst_1'$ with $(sst_1, obs(thr_1, mem_1)) \overset{j}{\rightsquigarrow}_p sst_1'$, where $p = \rho_j(conf_1)$. Note that $getT(succ_j(conf_1)) = update_j(thr_1, com_1, \alpha)$, $getM(succ_j(conf_1)) = mem_1'$, and $getS(succ_j(conf_1)) = sst_1'$.

5. Firstly, we assume that $thr_1(j) \in HCom$. We now show that in this case $P(succ_j(conf_1)) = P(conf_2)$ holds.
   (a) As $thr_1 \sim thr_2$, by the definition of low bisimulations modulo low matching we have $getT(succ_j(conf_1)) \sim thr_2$.
   (b) As $thr_1(j) \in HCom$, we have $getM(succ_j(conf_1)) =_L mem_2$.
   (c) As $thr_1(j) \in HCom$, $\alpha$ contains only high commands. Hence, $th\text{-}purge(thr_1) = th\text{-}purge(getT(succ_j(conf_1)))$.
   (d) As $conf_1 <_{\mathcal{S}} \langle th\text{-}purge(thr_1), mem_{1,p}, sst_p\rangle$, we have by the definition of $\mathcal{S}$-simulations that $succ_j(conf_1) <_{\mathcal{S}} \langle th\text{-}purge(thr_1), mem_{1,p}, sst_p\rangle$, and hence (using (c)) that
   $$succ_j(conf_1) <_{\mathcal{S}} \langle th\text{-}purge(getT(succ_j(conf_1))), mem_{1,p}, sst_p\rangle .$$
   (e) As $conf_1$ makes a transition to $succ_j(conf_1)$, we obtain that
   $$\sharp(Tr_{\mathcal{S}}(succ_j(conf_1))) + \sharp(Tr_{\mathcal{S}}(conf_2)) < n,$$
   and that $succ_l(conf_1)$ is terminating.
   (f) Due to (a), (b), (d), and (e), we can apply the induction hypothesis for the configurations $succ_j(conf_1)$ and $conf_2$, the memories $mem_{1,p}$ and $mem_{2,p}$, and the scheduler state $sst_p$, and obtain
   $$P(succ_j(conf_1)) = P(conf_2).$$

6. Using (5.), we rewrite $P(conf_1, high)$ as:

$$P(conf_1, high) = P(conf_2) * \sum \{\!| \rho_j(conf_1) \mid j \in enabledPos(conf_1) \ \wedge \ thr_1(j) \in HCom |\!\}.$$

   Note that this equals $P(conf_2) * (1 - l\text{-}prob_{\mathcal{S}}(conf_1))$.
   Analogously, we obtain $P(conf_2, high) = P(conf_1) * (1 - l\text{-}prob_{\mathcal{S}}(conf_2))$.

7. Now, we assume that $thr_1(j) \in LCom$ and that $k = \textit{l-match}_{thr_1,thr_2}(j)$.

8. We now show that

$$\rho_j(conf_1)/\textit{l-prob}_{\mathcal{S}}(conf_1) = \rho_k(conf_2)/\textit{l-prob}_{\mathcal{S}}(conf_2).$$

   (a) As $conf_1 <_{\mathcal{S}} \langle \textit{th-purge}(thr_1), mem_{1,p}, sst_p\rangle$, we obtain from the definition of $\mathcal{S}$-simulations that

$$\rho_j(conf_1)/\textit{l-prob}_{\mathcal{S}}(conf_1) = \rho_{j'}(\langle \textit{th-purge}(thr_1), mem_{1,p}, sst_p\rangle),$$

   where $j' = \textit{l-match}_{thr_1,\textit{th-purge}(thr_1)}(j)$.
   (b) Analogously, we have

$$\rho_k(conf_2)/\textit{l-prob}_{\mathcal{S}}(conf_2) = \rho_{k'}(\langle \textit{th-purge}(thr_2), mem_{2,p}, sst_p\rangle),$$

   where $k' = \textit{l-match}_{thr_2,\textit{th-purge}(thr_2)}(k)$.
   (c) Let $m$ be such that $thr_1(j)$ is the $m$th low thread in $thr_1$. Then $thr_2(k)$ is the $m$th low thread in $thr_2$ (as $k$ is obtained by computing the low match between $thr_1$ and $thr_2$ for $j$).
   By the definition of the purge function, we thus have $m = j' = k'$.
   (d) As $mem_{1,p} =_L mem_{2,p}$ and $\sharp(\textit{th-purge}(thr_1)) = \sharp(\textit{th-purge}(thr_2))$, we have $obs(\textit{th-purge}(thr_1), mem_{1,p}) = obs(\textit{th-purge}(thr_2), mem_{2,p})$ as $obs$ is confined. Hence,

$$\rho_m(\langle \textit{th-purge}(thr_1), mem_{1,p}, sst_p\rangle) = \rho_m(\langle \textit{th-purge}(thr_2), mem_{2,p}, sst_p\rangle).$$

   (e) Combining the equalities from (a), (b), and (d), we obtain the equality that we want to prove.

9. As $j \in \textit{enabledPos}(conf_1)$, we have $\rho_j(conf_1) > 0$. Then, by (8.), we also have $\rho_k(conf_2) > 0$, and hence $k \in \textit{enabledPos}(conf_2)$.

10. We will now show that $P(succ_j(conf_1)) = P(succ_k(conf_2))$.
   (a) As $thr_1 \sim thr_2$ and $mem_1 =_L mem_2$, we have $getT(succ_j(conf_1)) \sim getT(succ_k(conf_2))$ as well as $getM(succ_j(conf_1)) =_L getM(succ_k(conf_2))$ by the definition of low bisimulations modulo low matching.
   (b) As $conf_1 <_{\mathcal{S}} \langle \textit{th-purge}(thr_1), mem_{1,p}, sst_p\rangle$, the execution step from $conf_1$ to $succ_j(conf_1)$ can be simulated by an execution step of the configuration $\langle \textit{th-purge}(thr_1), mem_{1,p}, sst_p\rangle$ by the definition of $\mathcal{S}$-simulations, resulting in the configuration

$$\left\langle \textit{th-purge}(getT(succ_j(conf_1))), mem'_{1,p}, sst'_p\right\rangle$$

   for some $sst'_p$. Moreover, we have

$$succ_j(conf_1) <_{\mathcal{S}} \left\langle \textit{th-purge}(getT(succ_j(conf_1))), mem'_{1,p}, sst'_p\right\rangle.$$

   Finally, $getM(succ_j(conf_1)) =_L mem'_{1,p}$, as $thr_1$ is FSI-secure.

(c) We repeat step (b) with $conf_2$ instead of $conf_1$. Note that as *obs* is confined, we obtain the same scheduler state $sst'_p$, as *th-purge*$(thr_1)$ and *th-purge*$(thr_2)$ have the same number of threads, and $mem_{1,p} =_L mem_{2,p}$.

(d) As $conf_1$ and $conf_2$ are both terminating, $succ_j(conf_1)$ and $succ_k(conf_2)$ are also terminating. Moreover, we obtain

$$\sharp(Tr_{\mathcal{S}}(succ_j(conf_1))) + \sharp(Tr_{\mathcal{S}}(succ_k(conf_2))) < n.$$

(e) Using (a), (b), (c), and (d), we apply the induction hypothesis for the configurations $succ_j(conf_1)$ and $succ_k(conf_2)$, for the memories $mem'_{1,p}$ and $mem'_{2,p}$, and the scheduler state $sst'_p$, and obtain

$$P(succ_j(conf_1)) = P(succ_k(conf_2)).$$

11. Expressing $P(conf_1)$ and $P(conf_2)$ by using the equations from (6.), we obtain an equation system in $X = P(conf_1)$ and $Y = P(conf_2)$:

$$X = \sum \{\!|\rho_j(conf_1) * P(succ_j(conf_1)) \mid j \in enabledPos(conf_1) \ \wedge \ thr_1(j) \in LCom|\!\} + Y * (1 - l\text{-}prob_{\mathcal{S}}(conf_1))$$

$$Y = \sum \{\!|\rho_k(conf_1) * P(succ_k(conf_2)) \mid k \in enabledPos(conf_2) \ \wedge \ thr_2(k) \in LCom|\!\} + X * (1 - l\text{-}prob_{\mathcal{S}}(conf_1))$$

To complete the induction step it remains to show that the solution $(x, y)$ of this equation system satisfies $x = y$.

12. For solving the equation system, we introduce some abbreviations to make the formulas more readable. We define $S_i = P(conf_i, low)$ and $\rho_i = 1 - l\text{-}prob_{\mathcal{S}}(conf_i)$ for $i \in \{1, 2\}$. Using these notations, the equation system can be written as follows:

$$X = S_1 + Y * \rho_1 \tag{1}$$
$$Y = S_2 + X * \rho_2 \tag{2}$$

13. Solving the above equation system for $Y$ yields

$$Y = \frac{\rho_2 * S_1 + S_2}{1 - \rho_1 * \rho_2}.$$

We need to show that $X = Y$. To show this we show that this solution for $Y$ equals $X$ as represented in (1), where we substitute $Y$ with the above solution for $Y$. This resolves to

$$\frac{\rho_2 * S_1 + S_2}{1 - \rho_1 * \rho_2} = S_1 + \frac{\rho_2 * S_1 + S_2}{1 - \rho_1 * \rho_2} * \rho_1.$$

Multiplying both sides by $(1 - \rho_1 * \rho_2)$ yields

$$\rho_2 * S_1 + S_2 = \Big[1 - \rho_1 * \rho_2\Big] * S_1 + \rho_1 * \Big[\rho_2 * S_1 + S_2\Big].$$

The term $\rho_1 * \rho_2 * S_1$ can be canceled on the right side of the equation, we obtain

$$\rho_2 * S_1 + S_2 = S_1 + \rho_1 * S_2.$$

Expanding $\rho_1$ and $\rho_2$ with their definitions we obtain

$$S_1 - \textit{l-prob}_{\mathcal{S}}(\textit{conf}_2) * S_1 + S_2 = S_1 + S_2 - \textit{l-prob}_{\mathcal{S}}(\textit{conf}_1) * S_2,$$

which simplifies to

$$\textit{l-prob}_{\mathcal{S}}(\textit{conf}_2) * S_1 = \textit{l-prob}_{\mathcal{S}}(\textit{conf}_1) * S_2.$$

We expand $S_1$ and $S_2$ and obtain

$$\begin{aligned}
&\textit{l-prob}_{\mathcal{S}}(\textit{conf}_2) * \sum \{\!|\rho_j(\textit{conf}_1) * P(\textit{succ}_j(\textit{conf}_1)) \mid \\
&\qquad\qquad j \in \textit{enabledPos}(\textit{conf}_1) \ \wedge \textit{thr}_1(j) \in \textit{LCom} \wedge |\!\} \\
&= \textit{l-prob}_{\mathcal{S}}(\textit{conf}_1) * \sum \{\!|\rho_k(\textit{conf}_1) * P(\textit{succ}_k(\textit{conf}_2)) \mid \\
&\qquad\qquad k \in \textit{enabledPos}(\textit{conf}_2) \ \wedge \textit{thr}_2(k) \in \textit{LCom}|\!\}.
\end{aligned}$$

Now, we use the equalities derived in (8.) and in (10.), viz

$$\rho_j(\textit{conf}_1)/\textit{l-prob}_{\mathcal{S}}(\textit{conf}_1) = \rho_k(\textit{conf}_2)/\textit{l-prob}_{\mathcal{S}}(\textit{conf}_2),$$

and

$$P(\textit{succ}_j(\textit{conf}_1)) = P(\textit{succ}_k(\textit{conf}_2)),$$

where $k = \textit{l-match}_{\textit{thr}_1,\textit{thr}_2}(j)$. These equalities directly show that the two sides of the above equation are indeed equal. This finishes the proof of the induction step.

□

*Proof of Theorem 7.* Theorem 7 follows directly by applying Proposition 1 to the thread pools $\textit{thr}_1 = \textit{thr}_2 = \textit{thr}$, memories $\textit{mem}_1 = \textit{mem}_{1,p}$ respectively $\textit{mem}_2 = \textit{mem}_{2,p}$ with $\textit{mem}_1 =_L \textit{mem}_2$, and scheduler states $\textit{sst}_1, \textit{sst}_2$, and $\textit{sst}_p$ that are all equal to $\textit{sst}_0$.

## 7 Proof of Theorem 8 (Soundness of Security Type System)

**Theorem 8.** *If the judgment $\vdash \textit{com} : (\textit{ass}, \textit{stp})$ is derivable in the type system for some $\textit{com} \in \textit{Com}$ and $\textit{ass}, \textit{stp} \in \{\textit{low}, \textit{high}\}$ then com is FSI-secure.*

We firstly prove three auxiliary lemmas:

**Lemma 4.** *Let $com \in Com$ be typable as $(high, stp)$. Then $com \in HCom$.*

*Proof.* The proof is by induction over the derivation of the judgment $\vdash com : (high, stp)$:

1. Assume that $com = \mathsf{skip}$. Then $com \in HCom$ is trivially satisfied.
2. Assume that $com = var{:=}exp$. Then $dom(var) = high$ as $com$ is typable with $ass = high$, and therefore $com \in HCom$.
3. Assume that $com = \mathsf{if}\ (exp)\ \mathsf{then}\ com_1\ \mathsf{else}\ com_2\ \mathsf{fi}$. As $com$ is typable with $ass = high$, $com_1$ and $com_2$ are both typable with $ass_1, ass_2 = high$. Hence, by the induction hypothesis both $com_1$ and $com_2$ are contained in $HCom$. Thus, by the definition of the set $HCom$, $com \in HCom$.
4. Assume that $com = \mathsf{while}\ (exp)\ \mathsf{do}\ com'\ \mathsf{od}$. Then, as $com$ is typable with $ass = high$, $com'$ is also typable with $ass = high$. By the induction hypothesis, $com' \in HCom$. Thus, $com \in HCom$.
5. Assume that $com = com_1; com_2$. Then, as in the previous cases, one obtains that $com_1$ and $com_2$ are contained in $HCom$. Thus, by the definition of the set $HCom$, $com \in HCom$.
6. Assume that $com = \mathsf{spawn}(com_0, \ldots, com_k)$. Then, as $com$ is typable with $ass = high$, all the $com_i$ are typable with $ass_i = high$. Hence, by the induction hypothesis for all $i$ we have $com_i \in HCom$. Hence, $com \in HCom$.

□

**Lemma 5.** *Let $com \in Com$ be typable with $(ass, low)$. Let $mem_1 =_L mem_2$ and $com_1, com_2, \alpha_1, \alpha_2, mem_1', mem_2'$ with $\langle com, mem_1\rangle \xrightarrow{\alpha_1} \langle com_1, mem_1'\rangle$ and $\langle com, mem_2\rangle \xrightarrow{\alpha_2} \langle com_2, mem_2'\rangle$. Then $com_1 = com_2$, $\alpha_1 = \alpha_2$, and $mem_1' =_L mem_2'$.*

*Proof.* We prove the lemma by induction over the derivation of the judgment $\vdash com : (ass, low)$.

1. The cases for skip-statements, spawn-statements, and sequential composition are obvious.
2. Assume that $com = var{:=}exp$. The low memory is only changed by the assignment if $dom(var) = low$. In this case, by the typing rule for assignments, $dom(exp) = low$, and hence the new value of $var$ is equal when executing the assignment in low-equal states.
3. Assume that $com = \mathsf{if}\ (exp)\ \mathsf{then}\ com_1\ \mathsf{else}\ com_2\ \mathsf{fi}$. By the typing rule for conditionals, we have $dom(exp) = low$. Hence, when executing $com_1$ in two low-equal states, one arrives either in $com_1$ with both execution steps, or in $com_2$ with both execution steps, where the memory remains unchanged.
4. Assume that $com = \mathsf{while}\ (exp)\ \mathsf{do}\ com'\ \mathsf{od}$. By the typing rule for loops, we have $dom(exp) = low$. Hence, when executing $com_1$ in two low-equal states, one arrives either in $\mathsf{stop}$ with both execution steps, or in $com'; com$ with both execution steps, where the memory remains unchanged.

□

**Lemma 6.** *Let com be a typable command, and assume that* $\langle com, mem\rangle \xrightarrow{\alpha}\!\!\triangleright \langle com', mem'\rangle$ *for memories* $mem, mem'$, $com' \in Com$, *and a label* $\alpha$. *Then* $com'$ *is typable.*

*Proof.* We prove the statement by induction over the derivation of the typing of *com*.

1. The interesting case is $com = \mathsf{while}\ (exp)\ \mathsf{do}\ com_1\ \mathsf{od}$. By the typing rule for loops, *com* is typable as $(ass, stp \sqcup dom(exp))$ if $com_1$ is typable as $(ass, stp)$ and $stp \sqcup dom(exp) \sqsubseteq ass$.
2. If $com' \in Com$, then $com' = com_1; com$. This is typable by the rule for sequential composition, if $com'$ is typable as $(ass_1, stp_1)$, *com* is typable as $(ass_2, stp_2)$, and $stp_1 \sqsubseteq ass_1$.
3. Taking the types from (1.) for *com* and $com_1$, we hence need to show that $stp \sqcup dom(exp) \sqsubseteq ass$, which holds according to (1.).

□

We now prove Theorem 8:

*Proof.*

1. Let $thr_{com}$ be the thread pool consisting of the single command *com* (see Definition 8 in [MS10]). Assume without loss of generality that *com* is typable as $(ass, stp)$ with $ass = low$ (if $ass = high$, then *thr* is FSI-secure by Lemma 4 and Theorem 3), and that $com \in LCom$ (as otherwise $thr_{com}$ is FSI-secure by Theorem 3). We prove the FSI-security of $thr_{com}$ by induction over the derivation of the judgment $\vdash com : (ass, stp)$.
2. Assume that $com = \mathsf{skip}$. Then $thr_{com}$ is obviously FSI-secure.
3. Assume that $com = var{:=}exp$. As $ass = low$, by the typing rule for assignments we have $dom(var) = low$ and, hence, $dom(exp) = low$. Evaluating *exp* in two low-equal states yields the same value, and therefore executing $thr_{com}$ in two low-equal states equally modifies the low memory and results in the empty thread pool. Hence, $thr_{com}$ is FSI-secure.
4. Assume that $com = \mathsf{spawn}(com_1, \ldots, com_k)$. By the typing rule for the spawn-command, all $com_i$ are typable, and, hence, FSI-secure by the induction hypothesis. Executing *com* in two low-equal memories results in low-equal memories and equal thread pools consisting of FSI-secure threads which are hence FSI-secure by Theorem 2. In consequence, $thr_{com}$ is FSI-secure.
5. Assume that $com = \mathsf{if}\ (exp)\ \mathsf{then}\ com_1\ \mathsf{else}\ com_2\ \mathsf{fi}$.
   (a) If $dom(exp) = low$, then $thr_{com}$ always performs either a transition to $thr_{com_1}$ or a transition to $thr_{com_2}$ in low-equal states (as *exp* evaluates to the same value in low-equal states). By the induction hypothesis, the thread pools $thr_{com_1}$ and $thr_{com_2}$ are FSI-secure (and hence related to themselves by the relation $\sim$).
   (b) If $dom(exp) = high$, then $stp = high$ by the definition of the typing rule for conditionals. Furthermore, both $com_1$ and $com_2$ are typed with $(high, stp_1)$ and $(high, stp_2)$ for some $stp_1, stp_2$ due to the typing rule for conditionals. Hence, both $com_1$ and $com_2$ are high commands by

Lemma 4. Hence, $com$ is already a high command, which is FSI-secure by Theorem 3.

6. Assume that $com = com_1; com_2$. As $com \in LCom$, we have $com_1 \in LCom$. Furthermore, by the definition of the typing rule for sequential composition and the induction hypothesis, we have that $com_1$ and $com_2$ are FSI-secure. We will exhibit a low bisimulation modulo low matching that relates $thr_{com}$ to itself. We define a relation $R$ on thread pools with equal numbers of low threads as follows: $thr_1 \; R \; thr_2$ if and only if the following two conditions hold:
   (a) $thr_1(0) = thr_2(0) = com_i; com_{ii}$ for two commands $com_i$ and $com_{ii}$ and $thr_1(0)$ is typable with $stp = low$, or both $thr_1(0)$ and $thr_2(0)$ are high commands.
   (b) When removing the threads at position 0 from $thr_1$ respectively $thr_2$, the remaining thread pools are related by the relation $\sim$.

   We now show that the relation $R$ is a low bisimulation modulo low matching.
   - We only consider transitions of the thread at position 0. Transitions of the remaining threads comply with the requirements of low bisimulations modulo low matching by the requirement (b) in the definition of $R$.
   - If both $thr_1(0)$ and $thr_2(0)$ are high commands, then the requirements of low bisimulations modulo low matching are satisfied for those threads (compare proof of Theorem 3).
   - Now assume that $thr_1(0) = thr_2(0) = com_i; com_{ii}$ and that $thr_1(0)$ is typable with $stp = low$. Assume furthermore that $\langle com_i; com_{ii}, mem_1\rangle \xrightarrow{\alpha} \langle com_{iii}, mem_1'\rangle$. By the definition of the operational semantics for sequential composition, there are two cases:
     Case 1. $\langle com_i, mem_1\rangle \xrightarrow{\alpha} \langle \mathsf{stop}, mem_1'\rangle$ and $com_{iii} = com_{ii}$.
     Case 2. $\langle com_i, mem_1\rangle \xrightarrow{\alpha} \langle com_{iv}, mem_1\rangle$ and $com_{iii} = com_{iv}; com_{ii}$.
     By Lemma 5, in both cases the execution step can be simulated in a low-equal memory, resulting in equal commands and low-equal memories. Hence, the execution step of $thr_1(0)$ can be simulated by $thr_2(0)$. The resulting threads satisfy the requirements from the definition of $R$ (the command $com_{iii}$ is typable due to Lemma 6).

   Hence, $R$ is a low bisimulation modulo low matching. As $thr_{com} \; R \; thr_{com}$, $thr_{com}$ is FSI-secure.
7. Assume that $com = \mathsf{while}\ (exp)\ \mathsf{do}\ com_1\ \mathsf{od}$. This case is proven by combining the arguments from the cases for conditionals (5.) and sequential composition (6.).

□

## References


[MS10] H. Mantel and H. Sudbrock. Flexible Scheduler-Independent Security. In *Proceedings of the 15th European Symposium on Research in Computer Security (ESORICS)*, LNCS. Springer, 2010. to appear.