Addendum for the Paper
“Taming Message-passing Communication in Compositional Reasoning about Confidentiality”

Ximeng Li  Heiko Mantel  Markus Tasch

Department of Computer Science, TU-Darmstadt, Germany
{li, mantel, tasch}@mais.informatik.tu-darmstadt.de

In this document, we provide the supplementary material for the paper “Taming Message-passing Communication for Compositional Reasoning about Confidentiality” [1]. In Sect. 1 we provide the complete structural operational semantics for our programming language, and the complete instrumented semantics supporting our definition of “sound use of assumptions”. In Sect. 2 we provide the proofs for Theorem 1 and Theorem 2 from the paper. In Sect. 3 we present an information-flow type system for local security that does not contain semantic side conditions. In Sect. 4 we discuss the typability of the authentication example and the auction example from the paper, outlining the differences between the type system from the paper and the variant from Sect. 3 of this addendum.

1  Operational Semantics of Message-passing Language

1.1  Basic Semantics

Below we present the rules defining the operational semantics of our message-passing language.

\[
\begin{align*}
\langle\text{skip};\text{mem}\rangle;\sigma & \rightarrow \langle\text{stop};\text{mem}\rangle;\sigma \\
\langle x := e; \text{mem}\rangle;\sigma & \rightarrow \langle\text{stop}';\text{mem}'\rangle;\sigma \\
\end{align*}
\]

\[
b \in \{\text{tt},\text{ff}\} \quad [e]_{\text{mem}} = b
\]

\[
\langle\text{if e then prog}_1\text{ else prog}_2\text{ fi};\text{mem}\rangle;\sigma & \rightarrow \langle\text{prog}_1;\text{mem}\rangle;\sigma
\]

\[
\langle\text{while e do prog od; mem}\rangle;\sigma & \rightarrow \langle\text{prog};\text{while e do prog od; mem}\rangle;\sigma
\]

\[
\langle\text{while e do prog od; mem}\rangle;\sigma & \rightarrow \langle\text{stop};\text{mem}\rangle;\sigma
\]

\[
\langle\text{prog}_1;\text{mem}\rangle;\sigma & \rightarrow \langle\text{prog}_1';\text{mem}'\rangle;\sigma'
\]

\[
\langle\text{prog}_1;\text{prog}_2;\text{mem}\rangle;\sigma & \rightarrow \langle\text{prog}_1';\text{prog}_2';\text{mem}'\rangle;\sigma'
\]

\[
\sigma(ch) = v \cdot \gamma \quad \text{mem}' = \text{mem}[x \mapsto v] \quad \sigma' = \sigma[ch \mapsto \gamma]
\]

\[
\langle\text{recv}(ch,x);\text{mem}\rangle;\sigma & \rightarrow \langle\text{stop};\text{mem}'\rangle;\sigma'
\]

\[
\langle\text{if-recv}(ch,x,x_b);\text{mem}\rangle;\sigma & \rightarrow \langle\text{stop};\text{mem}'\rangle;\sigma'
\]

\[
\sigma(ch) = e \quad \text{mem}' = \text{mem}[x_b \mapsto \text{ff}]
\]

\[
\langle\text{if-recv}(ch,x,x_b);\text{mem}\rangle;\sigma & \rightarrow \langle\text{stop};\text{mem}'\rangle;\sigma'
\]

\[
\sigma(ch) = e \quad \text{mem}' = \text{mem}[x_b \mapsto \text{ff}]
\]

\[
\langle\text{send}(ch,e);\text{mem}\rangle;\sigma & \rightarrow \langle\text{stop};\text{mem}\rangle;\sigma'
\]

\[
\sigma(ch) = e \quad \text{mem}' = \text{mem}[x \mapsto v] \quad \sigma' = \sigma[ch \mapsto \sigma(ch) \cdot v]
\]

\[
\langle\text{send}(ch,e);\text{mem}\rangle;\sigma & \rightarrow \langle\text{stop};\text{mem}\rangle;\sigma'
\]

\[
\sigma(ch) = e \quad \text{mem}' = \text{mem}[x \mapsto v] \quad \sigma' = \sigma[ch \mapsto \sigma(ch) \cdot v]
\]
1.2 Instrumented Semantics

Below we present a calculus defining the judgment \( \text{pcnf}; \sigma; \mu \rightarrow_{\text{chs}} \text{pcnf'}; \sigma'; \mu' \). This judgment says: A process with the process configuration \( \text{pcnf} \) and the set \( \text{chs} \) of channels used in the continuation executes one step from the channel state \( \sigma \), resulting in the process configuration \( \text{pcnf'} \) and channel state \( \sigma' \), updating the instrumentation state from \( \mu \) to \( \mu' \).

In the calculus to be presented, we write \( \mu^* \) for \( \lambda ch: ICh. \mu(ch) \downarrow_1 \), and \( \mu^0 \) for \( \lambda ch: ICh. \mu(ch) \downarrow_2 \). We use the function \( \text{ichs-of} : \text{Prog} \rightarrow \mathcal{Q}^{ICh} \) to obtain the set of internal channels syntactically occurring in a program. This function is inductively defined by \( \text{ichs-of}(\text{send}(ch, e)) = \text{ichs-of}(\mu^* \text{recv}(ch, x)) = \text{ichs-of}(ch, e)) = \{ch\} \), \( \text{ichs-of}(\text{if } e \text{ then } \text{prog}_1 \text{ else } \text{prog}_2 \text{ fi}) = \text{ichs-of}(\text{prog}_1 \cup \text{prog}_2) = \text{ichs-of}(\text{prog}_1) \cup \text{ichs-of}(\text{prog}_2), \text{ichs-of}(\text{while } e \text{ do } \text{prog od}) = \text{ichs-of}(\text{prog}), \text{and \ ichs-of}(\text{prog}) = \emptyset \) otherwise.

\[
\begin{align*}
\langle \langle \text{skip}; \text{mem} \rangle \rangle; \sigma; \mu \rightarrow_{\text{chs}} & \langle \langle \text{stop}; \text{mem} \rangle \rangle; \sigma; \mu' \\
\text{lev}(e) = \mathbb{H} & \quad b \in \{tt, ff\} \quad \llbracket e \rrbracket_{\text{mem}} = b \\
\text{chs}' = \text{ichs-of}(\text{prog}_1) \cup \text{ichs-of}(\text{prog}_2) \cup \text{chs} & \quad \mu' = (\mu^* [\text{chs}' \mapsto \mathbb{H}[\mu^* (\text{chs})]], \mu^0 [\text{chs}' \mapsto \mathbb{H}]) \\
\langle \langle \text{if } e \text{ then } \text{prog}_1 \text{ else } \text{prog}_2 \text{ fi}; \text{mem} \rangle \rangle; \sigma; \mu \rightarrow_{\text{chs}} & \langle \langle \text{prog}_1; \text{mem} \rangle \rangle; \sigma; \mu' \\
\text{lev}(e) = \mathbb{L} & \quad b \in \{tt, ff\} \quad \llbracket e \rrbracket_{\text{mem}} = b \\
\text{chs}' = \text{ichs-of}(\text{prog}_1) \cup \text{ichs-of}(\text{prog}_2) \cup \text{chs} & \quad \mu' = (\mu^* [\text{chs}' \mapsto \mathbb{H}[\mu^* (\text{chs})]], \mu^0 [\text{chs}' \mapsto \mathbb{H}]) \\
\langle \langle \text{while } e \text{ do } \text{prog od}; \text{mem} \rangle \rangle; \sigma; \mu \rightarrow_{\text{chs}} & \langle \langle \text{prog}; \text{while } e \text{ do } \text{prog od}; \text{mem} \rangle \rangle; \sigma; \mu' \\
\text{lev}(e) = \mathbb{H} & \quad \llbracket e \rrbracket_{\text{mem}} = tt \\
\text{chs}' = \text{ichs-of}(\text{prog}_1) \cup \text{ichs-of}(\text{prog}_2) \cup \text{chs} & \quad \mu' = (\mu^* [\text{chs}' \mapsto \mathbb{H}[\mu^* (\text{chs})]], \mu^0 [\text{chs}' \mapsto \mathbb{H}]) \\
\langle \langle \text{while } e \text{ do } \text{prog od}; \text{mem} \rangle \rangle; \sigma; \mu \rightarrow_{\text{chs}} & \langle \langle \text{prog}; \text{while } e \text{ do } \text{prog od}; \text{mem} \rangle \rangle; \sigma; \mu' \\
\text{lev}(e) = \mathbb{L} & \quad \llbracket e \rrbracket_{\text{mem}} = ff \\
\langle \langle \text{while } e \text{ do } \text{prog od}; \text{mem} \rangle \rangle; \sigma; \mu \rightarrow_{\text{chs}} & \langle \langle \text{prog}; \text{while } e \text{ do } \text{prog od}; \text{mem} \rangle \rangle; \sigma; \mu' \\
\text{lev}(e) = \mathbb{L} & \quad \llbracket e \rrbracket_{\text{mem}} = ff \\
\langle \langle \text{prog}_1; \text{mem} \rangle \rangle; \sigma; \mu \rightarrow_{\text{chs}} & \langle \langle \text{stop}; \text{mem} \rangle \rangle; \sigma; \mu' \\
\text{chs}' = \text{chs} \cup \text{ichs-of}(\text{prog}_1) & \quad \text{prog}_1 \neq \text{stop} \\
\langle \langle \text{prog}_1; \text{prog}_2; \text{mem} \rangle \rangle; \sigma; \mu \rightarrow_{\text{chs}} & \langle \langle \text{prog}_1; \text{prog}_2; \text{mem} \rangle \rangle; \sigma; \mu' \\
\langle \langle \text{prog}_1; \text{mem} \rangle \rangle; \sigma; \mu \rightarrow_{\text{chs}'} & \langle \langle \text{stop}; \text{mem}' \rangle \rangle; \sigma'; \mu' \\
\text{chs}' = \text{chs} \cup \text{ichs-of}(\text{prog}_2) & \\
\langle \langle \text{prog}_1; \text{prog}_2; \text{mem} \rangle \rangle; \sigma; \mu \rightarrow_{\text{chs}} & \langle \langle \text{prog}_1; \text{prog}_2; \text{mem} \rangle \rangle; \sigma; \mu'
\end{align*}
\]
In the above, we obtain the trace underlying a rich trace using the function \( \text{trace} \).

We use the judgment

\[ \langle \text{pcnf} \rangle \rightarrow (\text{pcnf}') \]

† pcnf \( \rightarrow \) † pcnf to say that a process with the process configuration \( \text{pcnf} \) executes one step from the channel state \( \sigma \), resulting in the process configuration \( \text{pcnf}' \) and channel state \( \sigma' \), updating the instrumentation state from \( \mu \) to \( \mu' \). We define this judgment by \( \langle \text{pcnf} \rangle \rightarrow (\text{pcnf}') \) if and only if \( \text{pcnf} \rightarrow (\text{pcnf}') \). Our semantic instrumentation is transparent wrt. the basic semantics—it neither forbids any local computation steps, nor enables any additional ones. This result will be formalized in Sect.†

We use the judgment \( rtr \rightarrow \xi \) \( rtr' \) to state that the rich trace \( rtr' \) is extended into the rich trace \( rtr' \) after one step of a distributed program, or of the environment, under the strategy \( \xi \). We define this judgment on the basis of the judgment \( \langle \text{pcnf} \rangle \rightarrow (\text{pcnf}') \) for the process-local instrumented semantics.

\[
\begin{align*}
\text{last}(rtr) &= (\text{pcnfl}; \sigma; \mu) \\
\xi(\text{trace-of}(rtr)) &= \sigma' \\
\xi(rtr) &= (\text{pcnfl}; \sigma'; \mu') \\
\text{last}(rtr) &= (\text{pcnfl}; \sigma; \mu) \\
\xi(\text{trace-of}(rtr)) &= \sigma' \\
\xi(rtr) &= (\text{pcnfl}; \sigma'; \mu') \\
\end{align*}
\]

In the above, we obtain the trace underlying a rich trace using the function \( \text{trace-of} : \text{RTr} \rightarrow T \), where \( \text{trace-of}([([\text{pcnfl}_1; \sigma_1; \mu_1], \ldots, \text{pcnfl}_n; \sigma_n; \mu_n)]) = ([\text{pcnfl}_1; \sigma_1], \ldots, \text{pcnfl}_n; \sigma_n]) \).

Given a distributed program \( \text{dprog} = [\text{prog}_1 || \ldots || \text{prog}_n] \), we define

\[
\text{rtraces}_\xi(\text{dprog}) \equiv \{ rtr \mid ([([\text{prog}_1; \mu_{\text{init}}], \ldots, \text{prog}_n; \mu_{\text{init}}]); \sigma_{\text{init}}; \mu_{\text{init}}) \rightarrow (\xi) \}
\]

where \( \mu_{\text{init}} \) is \( \lambda ch: ICh(\epsilon, \mathbb{L}) \).
2 Proofs

We define a subset of high conditional programs by the following syntax. We denote the set of all high conditional programs by $HCond$.

$$hcond ::= \text{if } e \text{ then } prog_1 \text{ else } prog_2 \text{ fi } \langle lev(e) = \mathbb{H} \rangle \mid \text{while } e \text{ do } prog \text{ od } \langle lev(e) = \mathbb{H} \rangle \mid hcond; prog$$

**Lemma 1.** If $\langle prog; \text{mem}_1; \sigma_1; \mu_1 \rangle \rightarrow_{chs_1} \langle prog'_1; \text{mem}'_1; \sigma'_1; \mu'_1 \rangle$, $\langle prog; \text{mem}_2; \sigma_2; \mu_2 \rangle \rightarrow_{chs_2} \langle prog'_2; \text{mem}'_2; \sigma'_2; \mu'_2 \rangle$, mem$_1$ = mem$_2$, and prog$_1 \neq$ prog$_2$, then prog $\in$ $HCond$.

**Proof.** We proceed with a structural induction on prog $\in$ Prog.

- **Case** if $e$ then prog$_1$ else prog$_2$ fi: From mem$_1$ = mem$_2$ and prog$_1 \neq$ prog$_2$, it is not difficult to see that lev($e$) = $\mathbb{H}$. Thus, prog $\in$ $HCond$.

- **Case** while $e$ do prog$_1$ od: Analogous to the previous case.

- **Case** prog$_1$; prog$_2$: Without loss of generality, we first show prog$_1$ $\in$ $HCond$ by the following case analysis on prog$_1'$ and prog$_2'$ in the lemma statement.

  - **Sub-case** prog$_1' \neq$ prog$_2'$ $\land$ prog$_2' \neq$ prog$_2'$: There exist prog' and prog'' such that prog' $\neq$ prog'' , prog$_1' = prog'$; prog$_2' = prog''$; prog$_2$, and

$$\langle prog_1; \text{mem}_1; \sigma_1; \mu_1 \rangle \rightarrow_{chs_1 \cup \ichs-of(prog_2)} \langle prog'_1; \text{mem}'_1; \sigma'_1; \mu'_1 \rangle$$

$$\langle prog_1; \text{mem}_2; \sigma_2; \mu_2 \rangle \rightarrow_{chs_2 \cup \ichs-of(prog_2)} \langle prog''_1; \text{mem}'_2; \sigma'_2; \mu'_2 \rangle$$

Using the induction hypothesis, we can derive prog$_1 \in$ $HCond$.

  - **Sub-case** prog$_1' = prog_2'$ $\land$ prog$_2' \neq$ prog$_2'$: There exists some prog'' $\neq$ stop such that prog'$_2 = prog''$; prog$_2'$, and

$$\langle prog_1; \text{mem}_1; \sigma_1; \mu_1 \rangle \rightarrow_{chs_1 \cup \ichs-of(prog_2)} \langle (\text{stop}; \text{mem}'_1); \sigma'_1; \mu'_1 \rangle$$

$$\langle prog_1; \text{mem}_2; \sigma_2; \mu_2 \rangle \rightarrow_{chs_2 \cup \ichs-of(prog_2)} \langle (prog''_1; \text{mem}'_2); \sigma'_2; \mu'_2 \rangle$$

Using the induction hypothesis, we can derive prog$_1 \in$ $HCond$.

  - **Sub-case** prog$_1' = prog_2'$ $\land$ prog$_2' \neq$ prog$_2'$: Contradiction with prog$_1' \neq$ prog$_2'$.

Since prog$_1 \in$ $HCond$, prog$_2 \in$ Prog, we have prog$_1$; prog$_2 \in$ $HCond$.

In the remaining cases, the statement of the lemma vacuously holds.

**Lemma 2.** If prog $\in$ $HCond$, and $\langle prog; \text{mem}; \sigma; \mu \rangle \rightarrow_{chs} \langle prog'; \text{mem}'; \sigma'; \mu' \rangle$, then $\forall$ ch $\in$ ichs-of(prog) $\cup$ chs : $\exists$ llst $\in$ Lev$^*$ : $\mu'(ch) = (llst, \mathbb{H})$.

**Proof.** It is sufficient to prove

- If hcond $\rightarrow^* prog$, and $\langle prog; \text{mem}; \sigma; \mu \rangle \rightarrow_{chs} \langle prog'; \text{mem}'; \sigma'; \mu' \rangle$, then $\forall$ ch $\in$ ichs-of(prog) $\cup$ chs : $\exists$ llst $\in$ Lev$^*$ : $\mu'(ch) = (llst, \mathbb{H})$.

We proceed with an induction on the derivation of prog.

- Suppose hcond $\rightarrow$ if $e$ then prog$_1$ else prog$_2$ fi, lev($e$) = $\mathbb{H}$, and prog = if $e$ then prog$_1$ else prog$_2$ fi. It is straightforward to derive the conclusion of the lemma by inspection of the rules of the instrumented semantics.

- Suppose hcond $\rightarrow$ while $e$ do prog$_1$ od, lev($e$) = $\mathbb{H}$, and prog = while $e$ do prog$_1$ od. It is straightforward to derive the conclusion of the lemma by inspection of the rules of the instrumented semantics.
- Suppose $h \mathrel{\text{cond}} \rightarrow h\mathrel{\text{cond}_1}; \mathrel{\text{prog}_2}, h\mathrel{\text{cond}_1} \rightarrow^* \mathrel{\text{prog}_1}$, and $\mathrel{\text{prog}} = \mathrel{\text{prog}_1}; \mathrel{\text{prog}_2}$.
  
We have $(\langle \mathrel{\text{prog}_1}; \mathrel{\text{mem}} \rangle; \sigma; \mu) \mapsto \mathrel{\text{ch}}\mathrel{\text{ich}}\mathrel{\text{ch}}\mathrel{\text{of}}\mathrel{\text{(prog}_2)} \langle \langle \mathrel{\text{prog}_2}; \mathrel{\text{mem}}' \rangle; \sigma'; \mu' \rangle$. Thus by the induction hypothesis, we have $\forall \sigma \in \mathrel{\text{ich}}\mathrel{\text{ch}}\mathrel{\text{of}}\mathrel{\text{(prog}_1)} \cup (\sigma \cup \mathrel{\text{ich}}\mathrel{\text{ch}}\mathrel{\text{of}}\mathrel{\text{(prog}_2)}) : \exists \mathrel{\text{lst}} \in \mathrel{\text{H}} \mathrel{\text{ev}} : \mu'(\sigma) = \langle \mathrel{\text{lst}}, \mathrel{\text{H}} \rangle$. Hence $\forall \sigma \in \mathrel{\text{ich}}\mathrel{\text{ch}}\mathrel{\text{of}}\mathrel{\text{(prog}_1)} \cup \sigma : \exists \mathrel{\text{lst}} \in \mathrel{\text{H}} \mathrel{\text{ev}} : \mu'(\sigma) = \langle \mathrel{\text{lst}}, \mathrel{\text{H}} \rangle$.

This completes the proof. \qed

**Lemma 3.** If $\mathrel{\text{prog}} \in \mathrel{\text{H}} \mathrel{\text{cond}}$, and $\langle \langle \mathrel{\text{prog}}; \mathrel{\text{mem}} \rangle; \sigma; \mu \rangle \mapsto \langle \langle \mathrel{\text{prog}}'; \mathrel{\text{mem}}' \rangle; \sigma'; \mu' \rangle$, then we have $\forall \sigma \in \mathrel{\text{ich}}\mathrel{\text{ch}}\mathrel{\text{of}}\mathrel{\text{(prog)}} : \exists \mathrel{\text{lst}} \in \mathrel{\text{H}} \mathrel{\text{ev}} : \mu'(\sigma) = \langle \mathrel{\text{lst}}, \mathrel{\text{H}} \rangle$.

**Proof.** The result can be directly obtained by instantiating Lemma 2 with $\mathrel{\text{chs}} = \emptyset$. \qed

By writing the low projection $\mathrel{\text{vlst}}(\mathrel{\text{llst}}, \ell)$, we assume that the expression is defined. For a list $l = \langle a_1, \ldots, a_{|l| - 1} \rangle$, and $i \in \{0, \ldots, |l| - 1 \}$, we also write $l(i)$ for the element $a_i$.

**Lemma 4.** If $\mathrel{\text{vlst}}(\mathrel{\text{llst}}, \ell_1) = \mathrel{\text{vlst}}(\mathrel{\text{llst}}, \ell_2)$, then we have

1. $\ell_1 = \ell_2$,
2. $\ell_1 = \ell \Rightarrow (\mathrel{\text{llst}}(\ell_1) = \mathrel{\text{llst}}(\ell_2) \land |\mathrel{\text{vlst}}(\ell_1)| = |\mathrel{\text{vlst}}(\ell_2)|)$, and
3. $\ell_1 = \ell \land j \in \{0, \ldots, |\mathrel{\text{vlst}}(\ell_1)| - 1 \} \land \mathrel{\text{vlst}}(\ell_1)(j) = \mathrel{\text{vlst}}(\ell_2)(j)$.

**Proof.** Straightforward using the facts that $\mathrel{\text{vlst}}(\mathrel{\text{llst}}, \ell_1)$ and $\mathrel{\text{vlst}}(\mathrel{\text{llst}}, \ell_2)$ are defined, $\emptyset \not\in \mathrel{\text{Val}}$ and $\emptyset \not\in \mathrel{\text{Val}}$. \qed

We restate and prove Proposition 1.

**Proposition 1.** If $\forall \sigma \in \mathrel{\text{ICH}} : |\sigma(1)(ch)| = |\sigma(2)(ch)| |\sigma(2)(ch)| \mapsto \mathrel{\text{acs}}_1$, and $(\sigma_2, \mu_2) \mathrel{\text{val}} \mathrel{\text{acs}}_2$, then $\sigma_1 \mathrel{\text{asize}}_{\mathrel{\text{acs}}} = \sigma_2$.

**Proof.** Under the hypotheses of the proposition, we show

1. $\forall \sigma \in \mathrel{\text{ICH}} : (\emptyset^*, \sigma) \mathrel{\text{acs}} \cup \mathrel{\text{acs}}_2 \Rightarrow (|\sigma(1)(ch)| > 0 \lor |\sigma(2)(ch)| > 0) \Rightarrow \mathrel{\text{first}}(\sigma(1)(ch)) = \mathrel{\text{first}}(\sigma(2)(ch))$
2. $\forall \sigma \in \mathrel{\text{ICH}} : (\emptyset^*, \sigma) \mathrel{\text{acs}} \cup \mathrel{\text{acs}}_2 \Rightarrow (|\sigma(1)(ch)| > 0 \lor |\sigma(2)(ch)| > 0) \Rightarrow \mathrel{\text{second}}(\sigma(1)(ch)) = \mathrel{\text{second}}(\sigma(2)(ch))$

Pick an arbitrary $\sigma \in \mathrel{\text{ICH}}$. Suppose $(\emptyset^*, \sigma) \mathrel{\text{acs}} \cup \mathrel{\text{acs}}_2$. Without loss of generality, we assume $(\emptyset^*, \sigma) \mathrel{\text{acs}} \cup \mathrel{\text{acs}}_1$. From $(\sigma, \mu_1) \mathrel{\text{acs}}_1$, we have $|\mu_1(\sigma(1)(ch))| > 0 \Rightarrow \mathrel{\text{first}}(\mu_1(\sigma(1)(ch))) = \emptyset$. Hence $|\mu_1(\sigma(1)(ch))| > 0 \Rightarrow \mu_1(\sigma(1)(ch)) = \emptyset$. Thus we have if $|\sigma(1)(ch)| > 0$ then $\mathrel{\text{first}}(\sigma(1)(ch)) = \mathrel{\text{first}}(\sigma(2)(ch))$, by Lemma 2 and $|\sigma(1)(ch)| \mathrel{\text{asize}}_{\mathrel{\text{acs}}} = |\sigma(2)(ch)| \mathrel{\text{asize}}_{\mathrel{\text{acs}}}$. Suppose $(\emptyset^*, \sigma) \mathrel{\text{acs}} \cup \mathrel{\text{acs}}_2$. Without loss of generality, we assume $(\emptyset^*, \sigma) \mathrel{\text{acs}} \cup \mathrel{\text{acs}}_1$. From $(\sigma, \mu_1) \mathrel{\text{acs}} \cup \mathrel{\text{acs}}_1$, we have $\mu_1(\sigma(1)(ch)) = \emptyset$. Thus it is not difficult to derive $|\sigma(1)(ch)| > 0 \Rightarrow |\sigma(2)(ch)| > 0$ using $|\sigma(1)(ch)| \mathrel{\text{asize}}_{\mathrel{\text{acs}}} = |\sigma(2)(ch)| \mathrel{\text{asize}}_{\mathrel{\text{acs}}}$. \qed

**Lemma 5.** If $\mathrel{\text{mem}} = \emptyset \mathrel{\text{mem}}_2$, $\forall \sigma \in \mathrel{\text{ICH}} : |\sigma(1)(ch)| \mathrel{\text{asize}}_{\mathrel{\text{acs}}} = |\sigma(2)(ch)| \mathrel{\text{asize}}_{\mathrel{\text{acs}}}, \langle \langle \mathrel{\text{prog}}; \mathrel{\text{mem}}_1 \rangle; \sigma; \mu \rangle \mapsto \langle \langle \mathrel{\text{prog}}'; \mathrel{\text{mem}}_1' \rangle; \sigma'; \mu' \rangle$, and $\langle \langle \mathrel{\text{prog}}; \mathrel{\text{mem}}_2 \rangle; \sigma; \mu \rangle \mapsto \langle \langle \mathrel{\text{prog}}'; \mathrel{\text{mem}}_2' \rangle; \sigma'; \mu' \rangle$, then $\forall \sigma \in \mathrel{\text{ICH}} : |\sigma(1)(ch)| \mathrel{\text{asize}}_{\mathrel{\text{acs}}}$ = $|\sigma(2)(ch)| \mathrel{\text{asize}}_{\mathrel{\text{acs}}}$. \qed

**Proof.** We proceed with an induction on the derivation of

$$
\langle \langle \mathrel{\text{prog}}; \mathrel{\text{mem}}_1 \rangle; \sigma; \mu \rangle \mapsto \langle \langle \mathrel{\text{prog}}'; \mathrel{\text{mem}}_1' \rangle; \sigma'; \mu' \rangle
$$

(1)

We only present a few representative cases.
Lemma 6. The completion of the induction above completes the proof. ⊓⊔

Proof. We proceed with an induction on the derivation

\[ \langle \langle \text{prog; mem} \rangle \rangle; \sigma; \mu \rangle \rightarrow \langle \langle \text{prog'; mem'} \rangle \rangle; \sigma'; \mu' \rangle \]

We only show a few representative cases in detail.

Case (1) is established by a rule for send. Here prog is \text{send}(ch, e). If \( ch \in ECh \), then the conclusion of the lemma is trivial.

Suppose \( ch \in ICh \). We have

\[ \langle \langle \text{send}(ch, e); \text{mem}_1 \rangle \rangle; \sigma_1; \mu_1 \rangle \rightarrow \langle \langle \text{stop}; \text{mem}_1' \rangle \rangle; \sigma_1' ; \mu_1' \rangle \]
\[ \langle \langle \text{send}(ch, e); \text{mem}_2 \rangle \rangle; \sigma_2; \mu_2 \rangle \rightarrow \langle \langle \text{stop}; \text{mem}_2' \rangle \rangle; \sigma_2' ; \mu_2' \rangle \]

By \( \forall ch \in ICh : [\sigma_1(ch)]_{\mu_1(ch)} = [\sigma_2(ch)]_{\mu_2(ch)} \), and Lemma 6, we have \( \mu_1'(ch) = \mu_2'(ch) \) for the output channel ch.

We proceed with a case analysis on \( \mu_1'(ch) \).

Sub-case \( \mu_1'(ch) = H \). In this case we have \( (\mu_1')^o(ch) = \mu_1'(ch) = H \). We obviously have \( [\sigma_1'(ch)]_{\mu_1'(ch)} = [\sigma_2(ch)]_{\mu_2(ch)} \) for the output channel ch. Analogously we have \( [\sigma_1'(ch)]_{\mu_2'(ch)} = [\sigma_2(ch)]_{\mu_2(ch)} \). On this basis, it is not difficult to obtain \( \forall ch \in ICh : [\sigma_1'(ch)]_{\mu_1'(ch)} = [\sigma_2'(ch)]_{\mu_2'(ch)} \).

Sub-case \( \mu_1'(ch) = L \). We have \( (\mu_1')^o(ch) = \mu_1'(ch) = L \), and \( (\mu_2')^o(ch) = \mu_2'(ch) = L \).

If \( [e]_{\text{mem}_1} = [e]_{\text{mem}_2} \), then it is trivial to establish \( \forall ch \in ICh : [\sigma_1(ch)]_{\mu_1(ch)} = [\sigma_2(ch)]_{\mu_2(ch)} \).

Suppose \( [e]_{\text{mem}_1} \neq [e]_{\text{mem}_2} \). Since mem_1 = L, mem_2, we have lev(e) = H, and the two messages \( [e]_{\text{mem}_1} \) and \( [e]_{\text{mem}_2} \) are associated with the same security level H in \( (\mu_1')^*(ch) \) and \( (\mu_2')^*(ch) \).

On this basis, it is not difficult to obtain \( \forall ch \in ICh : [\sigma_1'(ch)]_{\mu_1'(ch)} = [\sigma_2'(ch)]_{\mu_2'(ch)} \).

Case Transition (1) is established by the first rule for sequential composition. We have that prog = prog_a; prog_b for some prog_a and prog_b, such that \( \langle \langle \text{prog_a; mem}_1 \rangle \rangle; \sigma_1; \mu_1 \rangle \rightarrow \langle \langle \text{prog_a'; mem}_1' \rangle \rangle; \sigma_1'; \mu_1' \rangle \) for some prog_a' with prog_a = prog_a' and \( \langle \langle \text{prog_a; mem}_2 \rangle \rangle; \sigma_2; \mu_2 \rangle \rightarrow \langle \langle \text{prog_a'; mem}_2' \rangle \rangle; \sigma_2'; \mu_2' \rangle \) for some prog_a' with prog_a = prog_a', or \( \text{prog_a} = \text{stop} \).

The conclusion of the lemma can now be derived using the induction hypothesis.

Case Transition (1) is established by a rule for if. We have \( \sigma_1' = \sigma_1, \sigma_2 = \sigma_2, \forall ch \in ICh : \mu_1'(ch) \subseteq (\mu_1')^o(ch) \land \mu_2'(ch) \subseteq (\mu_2')^o(ch) \), \( (\mu_1')^o(ch) = (\mu_2')^o(ch) \), and \( (\mu_1')^*(ch') = (\mu_2')^*(ch') = \mu_2'(ch') \) for all channels ch' such that \( (\mu_1')^o(ch') = (\mu_2')^o(ch') = L \).

On this basis, it is not difficult to derive the conclusion of the lemma.

The completion of the induction above completes the proof. □

Lemma 6. If \( \langle \langle \text{prog; mem} \rangle \rangle; \sigma; \mu \rangle \rightarrow \langle \langle \text{prog'; mem'} \rangle \rangle; \sigma'; \mu' \rangle \), \( \forall ch \in \text{ichs-of}(\text{prog}) : \mu^o(ch) = H \), and \( (\mu')^o = \mu^o \), then \( \forall ch \in ICh : [\sigma'(ch)]_{\mu'(ch)} = [\sigma(ch)]_{\mu(ch)} \).

Proof. We proceed with an induction on the derivation

\[ \langle \langle \text{prog; mem} \rangle \rangle; \sigma; \mu \rangle \rightarrow \langle \langle \text{prog'; mem'} \rangle \rangle; \sigma'; \mu' \rangle \] (2)

We only show a few representative cases in detail.

Case (2) is by the rule for send, over an external channel. Here prog is \text{send}(ch, e), with \( ch \in ECh \).

The result of the lemma is straightforward since \( \forall ch \in ICh : \sigma'(ch) = \sigma(ch) \), and \( \mu' = \mu \).

Case (2) is by the rule for send, over an internal channel. Here prog is \text{send}(ch, e) with \( ch \in ICh \).

We have \( ch \in \text{ichs-of}(\text{prog}) \). Hence \( \mu^o(ch) = H \). By the instrumented semantics, we have \( \forall ch' \in ICh \setminus \{ch\} : (\mu')^*(ch') = \mu^*(ch') \). Thus, we can derive the conclusion of the lemma using the hypothesis \( (\mu')^o = \mu^o \).
Case 2 is by a rule for sequential composition. We have \( \text{prog} = \text{prog}_1; \text{prog}_2 \) for some \( \text{prog}_1 \) and \( \text{prog}_2 \), and \( \langle \langle \text{prog}_1; \text{mem}; \sigma; \mu \rangle \rightarrow \langle \langle \text{prog}_1'; \text{mem}'; \sigma'; \mu' \rangle \rangle \) for some \( \text{prog}_1' \). The result of the lemma can now be derived using the induction hypothesis.

Case 2 is by a rule for if. We have \( \sigma' = \sigma \). Thus it is not difficult to derive the conclusion of the lemma using the hypothesis \( (\mu')^o = \mu^o \).

The remaining cases where a non-composite command is executed are straightforward – the cases for communication are analogous to the cases for send. The cases for while are analogous to the cases for if. The induction completes the proof.

**Lemma 7.** If \( \text{prog}_{10} = \text{prog}_{20} \), for all \( k < n \), \( \langle \text{prog}_{1k}; \text{mem}_{1k} \rangle \approx \langle \text{prog}_{2k}; \text{mem}_{2k} \rangle \), \( \langle \langle \text{prog}_{1(k+1)';} \text{mem}_{1(k+1)}; \sigma_1(k+1)'; \mu_1(k+1) \rangle \rightarrow \langle \langle \text{prog}_{2(k+1)';} \text{mem}_{2(k+1)}; \sigma_2(k+1)'; \mu_2(k+1) \rangle \rangle \), \( \forall \text{ch} \in \text{ICH} : [\sigma_1'_{k}(\text{ch})]_{\mu_1'_{k}(\text{ch})} = [\sigma_2'_{k}(\text{ch})]_{\mu_2'_{k}(\text{ch})} \), and \( \forall \text{ch} \in \text{ICH} : \mu_{k+1}(\text{ch}) \subseteq (\mu_{1k})^o(\text{ch}) \wedge \mu_{2k}^o(\text{ch}) \subseteq (\mu_{2k})^o(\text{ch}) \), then \( \forall \text{ch} \in \text{ICH} : [\sigma_{1n}(\text{ch})]_{\mu_{1n}(\text{ch})} = [\sigma_{2n}(\text{ch})]_{\mu_{2n}(\text{ch})} \).

**Proof.** We make a case analysis on whether the derivatives of \( \text{prog}_{10} \) and \( \text{prog}_{20} \) become different prior to \( \text{prog}_{1n} \), and \( \text{prog}_{2n} .

– Suppose there exists some \( k_0 < n \) such that \( \text{prog}_{1k_0} \neq \text{prog}_{2k_0} \). We have some \( k' \leq k_0 \) such that \( \text{prog}_{1(k'-1)} = \text{prog}_{2(k'-1)} \) but \( \text{prog}_{k'} \neq \text{prog}_{2k'} \). By \( \langle \text{prog}_{1(k'-1)}; \text{mem}_{1(k'-1)} \rangle \approx \langle \text{prog}_{2(k'-1)}; \text{mem}_{2(k'-1)} \rangle \), we have \( \text{mem}_{1(k'-1)} = \text{mem}_{2(k'-1)} \). Using Lemma 1, we can obtain \( \text{prog}_{1(k'-1)} \in \text{HCond} \) and \( \text{prog}_{2(k'-1)} \in \text{HCond} \). Using Lemma 2, we obtain that \( \forall \text{ch} \in \text{ichs-of}(\text{prog}_{1(k'-1)}) : \mu_{1k'}(\text{ch}) = \exists, \) and \( \forall \text{ch} \in \text{ichs-of}(\text{prog}_{2(k'-1)}) : \mu_{2k'}^o(\text{ch}) = \exists \). It is not difficult to see that

\[
\text{ichs-of}(\text{prog}_{1(k'-1)}) \supseteq \text{ichs-of}(\text{prog}_{1(n-1)}) \wedge \text{ichs-of}(\text{prog}_{2(k'-1)}) \supseteq \text{ichs-of}(\text{prog}_{2(n-1)})
\]

Using the condition that for all \( k < n \), \( \forall \text{ch} \in \text{ICH} : \mu_{1k}^o(\text{ch}) \subseteq (\mu_{1k}^o(\text{ch}))^o(\text{ch}) \wedge \mu_{2k}^o(\text{ch}) \subseteq (\mu_{2k}^o(\text{ch}))^o(\text{ch}) \), and the instrumented semantics, we derive \( \forall \text{ch} \in \text{ichs-of}(\text{prog}_{1(n-1)}) : (\mu_{1(n-1)}^o(\text{ch}))^o(\text{ch}) = \exists, \) and \( \forall \text{ch} \in \text{ichs-of}(\text{prog}_{2(n-1)}) : (\mu_{2(n-1)}^o(\text{ch}))^o(\text{ch}) = \exists \). It is not difficult to see that \( \mu_{1n}^o = (\mu_{1(n-1)}^o)^o \) and that \( \mu_{2n}^o = (\mu_{2(n-1)}^o)^o \).

Using Lemma 3, we can derive

\[
\forall \text{ch} \in \text{ICH} : [\sigma_{1(n-1)}(\text{ch})]_{\mu_{1(n-1)}(\text{ch})} = [\sigma_{1n}(\text{ch})]_{\mu_{1n}(\text{ch})}
\]

\[
\forall \text{ch} \in \text{ICH} : [\sigma_{2(n-1)}(\text{ch})]_{\mu_{2(n-1)}(\text{ch})} = [\sigma_{2n}(\text{ch})]_{\mu_{2n}(\text{ch})}
\]

Thus it follows from \( \forall \text{ch} \in \text{ICH} : [\sigma_{1(n-1)}(\text{ch})]_{\mu_{1(n-1)}(\text{ch})} = [\sigma_{2(n-1)}(\text{ch})]_{\mu_{2(n-1)}(\text{ch})} \) that \( \forall \text{ch} \in \text{ICH} : [\sigma_{1n}(\text{ch})]_{\mu_{1n}(\text{ch})} = [\sigma_{2n}(\text{ch})]_{\mu_{2n}(\text{ch})} \).

– Suppose for all \( k_0 < n \), \( \text{prog}_{1k_0} = \text{prog}_{2k_0} \). We have \( \text{prog}_{1(n-1)} = \text{prog}_{2(n-1)} \). It follows from Lemma 3 that \( \forall \text{ch} \in \text{ICH} : [\sigma_{1n}(\text{ch})]_{\mu_{1n}(\text{ch})} = [\sigma_{2n}(\text{ch})]_{\mu_{2n}(\text{ch})} \).

The case analysis above completes the proof.

**Proposition 2.** The following statements hold.

– If \( \langle \text{pcnf}; \sigma; \mu \rangle \rightarrow \langle \text{pcnf}; \sigma'; \mu' \rangle \), then \( \langle \text{pcnf}; \sigma \rangle \rightarrow \langle \text{pcnf}; \sigma' \rangle \).

– If \( \langle \text{pcnf}; \sigma \rangle \rightarrow \langle \text{pcnf}; \sigma' \rangle \), then for all \( \mu \), there exists \( \mu' \) such that \( \langle \text{pcnf}; \sigma; \mu \rangle \rightarrow \langle \text{pcnf}; \sigma'; \mu' \rangle \).

**Proof.** We prove augmented versions of the statements with universal quantification over sets \( \text{chs} \) of internal channels:
1. For all \( chs \in 2^{ICh} \), if \( \langle pcnf; \sigma; \mu \rangle \rightarrow_{chs} \langle pcnf'; \sigma'; \mu' \rangle \), then \( \langle pcnf; \sigma \rangle \rightarrow \langle pcnf'; \sigma' \rangle \).
2. If \( \langle pcnf; \sigma \rangle \rightarrow \langle pcnf'; \sigma' \rangle \), then for all \( \mu \), and \( chs \in 2^{ICh} \), there exists \( \mu' \) such that \( \langle pcnf; \sigma; \mu \rangle \rightarrow_{chs} \langle pcnf'; \sigma'; \mu' \rangle \).

Each of 1 and 2 can be proved straightforwardly by induction on the appropriate semantic derivation. We omit the details. It is not difficult to see that the proposition directly follows from 1 and 2. \( \Box \)

We proceed with the proof of Theorem 1 from the paper.

**Theorem 1.** For a distributed program \( dprog = \|i\progs_i \), if \( LSec(progs_i) \) holds for all \( i \), and \( dprog \) ensures a sound use of assumptions, then we have \( KBSec(dprog) \).

We construct the following binary relation on traces.

\[
R_{dprog} \triangleq \\
\{(trace-of(rtr_1), trace-of(rtr_2)) | rtr_1 \in traces_{\xi_1}(dprog) \land rtr_2 \in traces_{\xi_2}(dprog) \land |rtr_1| = |rtr_2| \land \\
\forall k \in \{0, \ldots, |rtr_1| - 1\} : \\
\exists n, prog_{11}, \ldots, prog_{1n}, prog_{21}, \ldots, prog_{2n}, mem_{11}, \ldots, mem_{1n}, mem_{21}, \ldots, mem_{2n}, \sigma_1, \sigma_2, \mu_1, \mu_2 : \\
rtr_1(k) = \langle \langle prog_{11}; mem_{11} \rangle, \ldots, \langle prog_{1n}; mem_{1n} \rangle; \sigma_1; \mu_1 \rangle \land \\
rtr_2(k) = \langle \langle prog_{21}; mem_{21} \rangle, \ldots, \langle prog_{2n}; mem_{2n} \rangle; \sigma_2; \mu_2 \rangle \land \\
(\forall j \in \{1, \ldots, n\} : \langle prog_{1j}; mem_{1j} \rangle \approx \langle prog_{2j}; mem_{2j} \rangle) \land \\
\sigma_1 \simeq \sigma_2 \land \forall ch \in ICh : [\sigma_1(ch)]_{\mu_1(ch)} = [\sigma_2(ch)]_{\mu_2(ch)} \land \\
\left(k < |rtr_1| - 1 \Rightarrow \left(\forall j : \text{step from } rtr_1(k) \text{ is by } prog_{1j} \Leftrightarrow \text{step from } rtr_2(k) \text{ is by } prog_{2j}\right)\right)
\}
\]

Theorem 1 immediately follows from the following lemma, which is stated using \( R_{dprog} \).

**Lemma 8.** For \( dprog = \|i\progs_i \), if \( \forall i \in \{1, \ldots, n\} : LSec(progs_i) \), and \( dprog \) ensures a sound use of assumptions, then for all \( \xi_1, \xi_2 \) satisfying \( \xi_1 \simeq \xi_2 \), all \( tr_1 \in traces_{\xi_1}(dprog) \), there exists \( tr_2 \in traces_{\xi_2}(dprog) \), such that \( (tr_1, tr_2) \in R_{dprog} \).

**Proof.** Fix \( tr_1 \in traces_{\xi_1}(dprog) \). We prove that there exists \( tr_2 \in traces_{\xi_2}(dprog) \) such that \( (tr_1, tr_2) \in R_{dprog} \) by an induction on \( |tr_1| \).

**Base case.** We prove the result for the case where \( |tr_1| = 1 \). By \( tr_1 \in traces_{\xi_1}(dprog) \), we have \( tr_1 = [\langle \langle prog_1; mem_{init} \rangle, \ldots, \langle prog_n; mem_{init} \rangle; \sigma_{init} \rangle] \). Pick \( tr_2 = tr_1 \). By \( \forall i \in \{1, \ldots, n\} : LSec(progs_i) \), we have \( \forall i \in \{1, \ldots, n\} : \langle prog_i; mem_{init} \rangle \approx \langle prog_i; mem_{init} \rangle \). Using \( rtr_1 = rtr_2 = [\langle \langle prog_1; mem_{init} \rangle, \ldots, \langle prog_n; mem_{init} \rangle; \sigma_{init}; mem_{init} \rangle] \), it is not difficult to obtain \( (tr_1, tr_2) \in R_{dprog} \).

**Inductive case.** We prove the existence of a proper \( tr_2 \) in the case where \( |tr_1| = n \) ( \( n > 1 \)).

We know that there exist some \( tr_{10} \) and \( gcnf_1 \) such that \( |tr_{10}| = n - 1 \) and \( tr_1 = tr_{10} \cdot gcnf_1 \). By the induction hypothesis, there exists \( tr_{20} \in traces_{\xi_2}(dprog) \) such that \( (tr_{10}, tr_{20}) \in R_{dprog} \). By
the construction of $R_{dprog}$, there exist $rtr_{10}$ and $rtr_{20}$ such that

$$tr_{10} = \text{trace-of}(rtr_{10}) \land tr_{20} = \text{trace-of}(rtr_{20})$$

$$rtr_{10} \in \text{traces}_{\xi_1}(dprog) \land rtr_{20} \in \text{traces}_{\xi_2}(dprog)$$

$$|rtr_{10}| = |rtr_{20}|$$

$$\forall k \in \{0, \ldots, |rtr_{10}| - 1 \}:$$

$$\exists n, prog_{11}, \ldots, prog_{1n}, \ prog_{21}, \ldots, \ prog_{2n}, \ mem_{11}, \ldots, \ mem_{1n}, \ mem_{21}, \ldots, \ mem_{2n}, \sigma_1, \sigma_2, \sigma_1', \sigma_2', \mu_1, \mu_2 :$$

$$rtr_{10}(k) = (((prog_{11}; mem_{11}), \ldots, (prog_{1n}; mem_{1n}); \sigma_1; \mu_1) \land$$

$$rtr_{20}(k) = (((prog_{21}; mem_{21}), \ldots, (prog_{2n}; mem_{2n}); \sigma_2; \mu_2) \land$$

$$\forall j \in \{1, \ldots, n \} : \langle prog_{1j}; mem_{1j} \rangle \equiv \langle prog_{2j}; mem_{2j} \rangle \land$$

$$\sigma_1 \preceq \sigma_2 \land \forall ch \in ICh : \sigma_1(ch)_{\mu_1(ch)} = \sigma_2(ch)_{\mu_2(ch)} \land$$

$$\left( k < |rtr_{10}| - 1 \Rightarrow \left( \forall j : \text{step from } rtr_{10}(k) \text{ is by } prog_{1j} \Leftrightarrow \text{step from } rtr_{20}(k) \text{ is by } prog_{2j} \right) \land \text{step from } rtr_{20}(k) \text{ is by env.} \right) \right)$$

By (6), there exist $prog_{11}, \ldots, prog_{1n}, \ prog_{21}, \ldots, \ prog_{2n}, \ mem_{11}, \ldots, \ mem_{1n}, \ mem_{21}, \ldots, \ mem_{2n}, \sigma_1, \sigma_2, \mu_1, \mu_2$ such that

$$rtr_{10}(n - 2) = (((prog_{11}; mem_{11}), \ldots, (prog_{1n}; mem_{1n}); \sigma_1; \mu_1)$$

$$rtr_{20}(n - 2) = (((prog_{21}; mem_{21}), \ldots, (prog_{2n}; mem_{2n}); \sigma_2; \mu_2)$$

$$\forall j \in \{1, \ldots, n \} : \langle prog_{1j}; mem_{1j} \rangle \equiv \langle prog_{2j}; mem_{2j} \rangle$$

$$\sigma_1 \preceq \sigma_2$$

$$\forall ch \in ICh : \sigma_1(ch)_{\mu_1(ch)} = \sigma_2(ch)_{\mu_2(ch)}$$

We make a case analysis as to whether the step from the last configuration of $tr_{10}$ in $tr_1$ is by a process or by the environment.

- Suppose the step from the last configuration of $tr_{10}$ in $tr_1$ is by the $j$-th process. That is, we have $\langle prog_{1j}; mem_{1j} \rangle; \sigma_1 \rightharpoonup \langle prog'_{1j}; mem'_{1j} \rangle; \sigma_1'$ for some $prog'_{1j}$, $mem'_{1j}$, and $\sigma_1'$. Hence, there exists $\mu_1'$ such that

$$\langle prog_{1j}; mem_{1j} \rangle; \sigma_1; \mu_1 \Rightarrow \langle prog'_{1j}; mem'_{1j} \rangle; \sigma_1'; \mu_1'$$

by Proposition 4. Let $rtr_1 = rtr_{10} \cdot \left[ \ldots, (prog'_{1j}; mem'_{1j}), \ldots; \sigma_1'; \mu_1' \right]$. We then have $rtr_1 \in traces_{\xi_1}(dprog)$. We also have $tr_1 = \text{trace-of}(rtr_1)$.

From the hypothesis that $dprog$ ensures a sound use of assumptions, (4), and (7), we have

$$(\sigma_1, \mu_1) \models \text{asm-of}(prog_{1j}) \land (\sigma_2, \mu_2) \models \text{asm-of}(prog_{2j})$$

From (11), (13), and Proposition 1, we have

$$\sigma_1 \xrightarrow{\text{asm-of}(prog_{1j}) \cup \text{asm-of}(prog_{2j})} \sigma_2$$

By (13), we also have

$$\sigma_1 \preceq^{NE} \text{asm-of}(prog_{1j}) \land \sigma_2 \preceq^{NE} \text{asm-of}(prog_{2j})$$
By (9), (10), (14), (15), and (12), there exist \( prog'_2, \ mem'_2, \sigma'_2 \) and \( \mu'_2 \) such that
\[
\langle \langle prog_2; \ mem_2 \rangle; \sigma_2; \mu_2 \rangle \rightarrow \langle \langle prog'_2; \ mem'_2 \rangle; \sigma'_2; \mu'_2 \rangle
\]
(16)
\[
\sigma'_2 \simeq \Xi \sigma'_2
\]
(17)
\[
\langle \langle prog'_1; \ mem'_1 \rangle \rangle \simeq \langle \langle prog'_2; \ mem'_2 \rangle \rangle
\]
(18)

We proceed to show \( \forall ch \in ICh \colon [\sigma'_1(ch)]_{\mu'_1(ch)} = [\sigma'_2(ch)]_{\mu'_2(ch)} \). Without loss of generality, we assume that the indices \( k \) such that the steps from \( \tau r_{10}(k) \) and \( \tau r_{20}(k) \) are taken by the \( j \)-th process are \( k_1, \ldots, k_m \) for some \( m \geq 0 \), where \( k_m = n - 2 \). For each \( r \), supposing the instrumentation states right after the steps from \( \tau r_{10}(k_r) \) and \( \tau r_{20}(k_r) \) are \( \mu'_{1k_r} \) and \( \mu'_{2k_r} \), and the instrumentation states right before the steps from \( \tau r_{10}(k_{r+1}) \) and \( \tau r_{20}(k_{r+1}) \) are \( \mu_{1k_{r+1}} \) and \( \mu_{2k_{r+1}} \), we have \( \forall ch \in ICh \colon (\mu'_{1k_r})^{\circ}(ch) \subseteq \mu_{1k_{r+1}}(ch) \wedge (\mu'_{2k_r})^{\circ}(ch) \subseteq \mu_{2k_{r+1}}(ch) \), by (11). Using the above conditions together with (6), we now have two related executions of \( prog \) (that might be interrupted in the middle by steps from other programs or from the overall environment of the distributed program) that satisfy the conditions of Lemma 7. Thus we have
\[
\forall ch \in ICh \colon [\sigma'_1(ch)]_{\mu'_1(ch)} = [\sigma'_2(ch)]_{\mu'_2(ch)}
\]
(19)
by Lemma 7.

Let \( \tau r_2 = \tau r_{10} \cdot \langle [\ldots, \langle prog'_2; \ mem'_2 \rangle, \ldots]; \sigma'_2; \mu'_2 \rangle \) and \( \tau r_2 = \text{trace-of}(\tau r_2) \). It is not difficult to see that \( \tau r_2 \in \text{traces}_{\xi_2}(dprog) \) with the help of Proposition 2. We can derive \( (\tau r_1, \tau r_2) \in D_{dprog} \) by using \( \tau r_1 \) and \( \tau r_2 \).

Suppose the step from the last configuration of \( \tau r_{10} \) in \( \tau r_1 \) is by the environment. We have
\[
\tau r_1 = \text{trace-of}(\tau r_{10} \cdot \langle [\langle prog_{11}; \ mem_{11} \rangle, \ldots, \langle prog_{1m}; \ mem_{1m} \rangle]; \sigma'_1; \mu'_1 \rangle)
\]
where \( \sigma'_1 = [ECh \mapsto \xi_1(\tau r_{10})(ECh)] \).

Let
\[
\tau r_2 = \text{trace-of}(\tau r_{20} \cdot \langle [\langle prog_{21}; \ mem_{21} \rangle, \ldots, \langle prog_{2m}; \ mem_{2m} \rangle]; \sigma'_2; \mu'_2 \rangle)
\]

where \( \sigma'_2 = [ECh \mapsto \xi_2(\tau r_{20})(ECh)] \). From (6), we have \( \tau r_{10} \simeq \tau r \tau r_{20} \). Thus we have \( \sigma'_1 \simeq \sigma'_2 \) from (10), and the hypothesis \( \xi_1 \simeq \Xi \xi_2 \).

On this basis, it is not difficult to establish \( (\tau r_1, \tau r_2) \in D_{dprog} \).

The induction on \(|\tau r_1|\) above completes the proof.

We define pairings \( (\mathbb{C}[\uplus_1, \ldots, \uplus_n], \varphi(\uplus_1, \ldots, \uplus_n)) \) of secure contexts and conditions, with holes \( \uplus_1, \ldots, \uplus_n \) to be filled in with programs.

\[
(\mathbb{C}[\uplus_1, \ldots, \uplus_n], \varphi(\uplus_1, \ldots, \uplus_n)) \ ::= \begin{cases} 
\text{(skip, true)} & \\
| (x := e, \text{lev}(e) \subseteq \text{lev}(x)) & \\
| (\text{send}(ch, e), \text{lev}(e) \subseteq \text{lev}^\ast(ch, \emptyset)) & \\
| (as_{\text{rec}}(ch, x), (\text{NE} \notin \text{as} \Rightarrow \text{lev}^\circ(ch, as) \subseteq \text{ll}) \wedge \text{lev}^\circ(ch, as) \subseteq \text{lev}(x)) & \\
| (as_{\text{if-rec}}(ch, x, x_0), (\text{NE} \notin \text{as} \Rightarrow \text{lev}^\circ(ch, as) \subseteq \text{lev}(x_0)) \wedge \text{lev}^\circ(ch, as) \subseteq \text{lev}(x)) & \\
| (\text{if e then } \uplus_1 \text{ else } \uplus_2 \text{ fi}, \text{lev}(e) = \text{ll} \Rightarrow \uplus_1 \sim \uplus_2) & \\
| (\text{while e do } \uplus_1 \text{ od}, \text{lev}(e) = \text{ll}) & \\
| (\uplus_1; \uplus_2, \text{ true}) & 
\end{cases}
\]

We prove the following hook-up property of \( LSec(\cdot) \).
Proposition 3. For all indices \( n \geq 0 \), pairs \((C[\mathbf{\Sigma}], \phi(\mathbf{\Sigma}))\), programs \( \text{prog}_1, \ldots, \text{prog}_n \), we have \( \text{LSec}(C[\text{prog}_1, \ldots, \text{prog}_n]) \), if we have \( \phi(\text{prog}_1, \ldots, \text{prog}_n) \) and \( \forall j \in \{1, \ldots, n\} : \text{LSec}(\text{prog}_j) \).

Proof. We first consider the non-composite cases \((C[\text{prog}_1, \ldots, \text{prog}_n], \phi(...)) = (\text{prog}, \phi(...))\) where \text{prog} is one of \text{skip}, \text{send}(\text{ch}, e), \text{recv}(\text{ch}, x), \text{as-recv}(\text{ch}, x, x_b).

For each of the non-composite cases, we construct the relation

\[
\mathcal{R}_{\text{prog}} = \{((\text{prog}; \text{mem}_1), (\text{prog}; \text{mem}_2)) \mid \text{mem}_1 \subseteq \text{mem}_2\} \cup \{((\text{stop}; \text{mem}_1), (\text{stop}; \text{mem}_2)) \mid \text{mem}_1 \subseteq \text{mem}_2\}
\]

It is trivial that for all \text{mem}_1 and \text{mem}_2 such that \text{mem}_1 \subseteq \text{mem}_2, we have \langle \text{prog}; \text{mem}_1 \rangle \mathcal{R}_{\text{prog}} \langle \text{prog}; \text{mem}_2 \rangle.

We proceed to show that \( \mathcal{R}_{\text{prog}} \) is an assumption-aware bisimulation. It is not difficult to see that \( \mathcal{R}_{\text{prog}} \) is symmetric. Pick arbitrary pair \((\langle \text{prog}_1; \text{mem}_1 \rangle, \langle \text{prog}_2; \text{mem}_2 \rangle)\) from \( \mathcal{R}_{\text{prog}} \). We have \text{mem}_1 \subseteq \text{mem}_2. \) We also have \text{prog}_1 = \text{stop} \iff \text{prog}_2 = \text{stop}.

Pick arbitrary channel states \( \sigma_1 \) and \( \sigma_2 \) such that \( \sigma_1 \simeq_{\text{Sigma}} \sigma_2 \).

We show how the remaining proof obligations are discharged for the interesting cases for \text{prog} below.

Case \text{send}(\text{ch}, e). We assume the hypothesis

\[
\text{lev}(e) \subseteq \text{lev}^*(\text{ch}, \emptyset)
\]

We also assume \( \sigma_1 \equiv \sigma_2 \), \( \sigma_1 \equiv_{\text{NE}} \emptyset \), and \( \sigma_2 \equiv_{\text{NE}} \emptyset \).

Suppose for some \( \sigma'_1 \) we have

\[
\langle (\text{send}(\text{ch}, e); \text{mem}_1); \sigma_1 \rangle \rightarrow \langle (\text{stop}; \text{mem}_1); \sigma'_1 \rangle
\]

It is not difficult to see that there is some \( \sigma'_2 \) such that

\[
\langle (\text{send}(\text{ch}, e); \text{mem}_2); \sigma_2 \rangle \rightarrow \langle (\text{stop}; \text{mem}_2); \sigma'_2 \rangle
\]

We have \langle \text{stop}; \text{mem}_1 \rangle \mathcal{R}_{\text{prog}} \langle \text{stop}; \text{mem}_2 \rangle by \text{mem}_1 \subseteq \text{mem}_2. \) We proceed to show \( \sigma'_1 \simeq_{\text{Sigma}} \sigma'_2 \) with a case analysis on whether \( \text{ch} \in \text{ECh} \).

Sub-case \( \text{ch} \in \text{ECh} \). We make a further case analysis on the class of \( \text{ch} \).

- Suppose \( \text{ch} \in \text{PriCh} \). We have \( \text{ob}(\text{ch}, \sigma'_1) = \text{ob}(\text{ch}, \sigma'_2) = \emptyset \). It is not difficult to see \( \sigma'_1 \simeq_{\text{Sigma}} \sigma'_2 \), since the message queues for the other channels are unchanged in \( \sigma'_1 \) and \( \sigma'_2 \).

- Suppose \( \text{ch} \in \text{EncCh} \). From \( \sigma_1 \simeq_{\text{Sigma}} \sigma_2 \), we have \( \text{ob}(\text{ch}, \sigma_1) = \text{ob}(\text{ch}, \sigma_2) = \bigcirc^n \) for some \( n \). We then have \( \text{ob}(\text{ch}, \sigma'_1) = \text{ob}(\text{ch}, \sigma'_2) = \bigcirc^{n+1} \). It is not difficult to see that \( \sigma'_1 \simeq_{\text{Sigma}} \sigma'_2 \).

- Suppose \( \text{ch} \in \text{PubCh} \). From \( \sigma_1 \simeq_{\text{Sigma}} \sigma_2 \), we have \( \text{ob}(\text{ch}, \sigma_1) = \text{ob}(\text{ch}, \sigma_2) = \text{vlst} \) for some list \( \text{vlst} \) of values. We have \( \text{lev}^*(\text{ch}, \emptyset) = \text{L} \). We thus have \( \text{lev}(e) = \text{L} \) by \( [22] \). Thus we have \( \lbrack e \rbrack_{\text{mem}_1} = \lbrack e \rbrack_{\text{mem}_2} \) by \( \text{mem}_1 \subseteq \text{mem}_2 \). We have \( \text{ob}(\text{ch}, \sigma'_1) = \text{ob}(\text{ch}, \sigma'_2) = \text{vlst} \cdot v \), where \( v = \lbrack e \rbrack_{\text{mem}_1} = \lbrack e \rbrack_{\text{mem}_2} \). It is not difficult to see that \( \sigma'_1 \simeq_{\text{Sigma}} \sigma'_2 \).

Sub-case \( \text{ch} \in \text{ICh} \). Since the message queues of all the external channels are unaffected, we trivially have \( \sigma'_1 \simeq_{\text{Sigma}} \sigma'_2 \).
Case $\text{as}_{\text{recv}}(ch, x)$. We assume the following hypotheses

$$NE \notin as \Rightarrow \text{lev}^=(ch, as) = \emptyset$$  \hspace{1cm} (23)

$$\text{lev}(ch, as) \cup \text{lev}^*(ch, as) \subseteq \text{lev}(x)$$  \hspace{1cm} (24)

We also assume

$$\sigma_1 \overset{\text{as}(ch)}{\longrightarrow} \sigma_2$$  \hspace{1cm} (25)

$$\sigma_1 \not\equiv_{\text{as}} (ch) \land \sigma_2 \equiv_{\text{as}} (ch)$$  \hspace{1cm} (26)

Suppose for some $\text{mem}_1'$ and $\sigma_1'$ we have

$$\langle \langle \text{as}_{\text{recv}}(ch, x); \text{mem}_1'; \sigma_1 \rangle \rangle \rightarrow \langle \langle \text{stop}; \text{mem}_1'; \sigma_1' \rangle \rangle$$  \hspace{1cm} (27)

We make a further case analysis on whether $ch \in ECh$ to discharge the remaining proof obligations.

Sub-case $ch \in ECh$. We make a further case analysis on whether $NE \in as$.

- Suppose $NE \in as$.
  
  By (26), we have $|\sigma_2(ch)| > 0$. Hence there exists some $\sigma_2'$ and $\text{mem}_2'$ such that

  $$\langle \langle \text{as}_{\text{recv}}(ch, x); \text{mem}_2'; \sigma_1 \rangle \rangle \rightarrow \langle \langle \text{stop}; \text{mem}_2'; \sigma_2' \rangle \rangle$$

  We make a further case analysis on the class of $ch$.

- Suppose $ch \in \text{PriCh}$. By (24), we have $\text{lev}(x) = \emptyset$. Thus we vacuously have $\text{lev}(x) = \emptyset \Rightarrow \text{mem}_1'(x) = \text{mem}_2'(x)$. It is not difficult to establish $\text{mem}_1' =_L \text{mem}_2'$ since the values for the other variables are unchanged in $\text{mem}_1'$ and $\text{mem}_2'$. Hence we have $\langle \text{stop}; \text{mem}_1' \rangle \equiv_{\text{prog}} \langle \text{stop}; \text{mem}_2' \rangle$.

  We have $\text{ob}(ch, \sigma_1') = \text{ob}(ch, \sigma_2') = \emptyset$. Thus, it is not difficult to establish $\sigma_1' \equiv_{\Sigma} \sigma_2'$, since the message queues for the other channels are unchanged in $\sigma_1'$ and $\sigma_2'$.

- Suppose $ch \in \text{EncCh}$. That $\langle \text{stop}; \text{mem}_1' \rangle \equiv_{\text{prog}} \langle \text{stop}; \text{mem}_2' \rangle$ can be established analogously to the case where $ch \in \text{PriCh}$.

  From $\sigma_1 \equiv_{\Sigma} \sigma_2$, we have $\text{ob}(ch, \sigma_1) = \text{ob}(ch, \sigma_2) = \bigcirc^n$ for some $n$, where $n > 0$ since $NE \in as$. Hence, $\text{ob}(ch, \sigma_1') = \text{ob}(ch, \sigma_2') = \bigcirc^{n-1}$. On this basis, it is not difficult to see that $\sigma_1' \equiv_{\Sigma} \sigma_2'$.

- Suppose $ch \in \text{PubCh}$.

  From $\sigma_1 \equiv_{\Sigma} \sigma_2$, we have $\text{ob}(ch, \sigma_1) = \text{ob}(ch, \sigma_2) = \text{vlst}$, where $\text{vlst}$ is a non-empty list of values since $NE \in as$.

  Suppose $\text{vlst} = v \cdot \text{vlst}'$ for some $v \in \text{Val}$ and $\text{vlst}' \in \text{Val}^\ast$. We have $\text{mem}_1'(x) = \text{mem}_2'(x) = v$. On this basis, it is not difficult to see that $\text{mem}_1' =_L \text{mem}_2'$ holds. Hence we have $\langle \text{stop}; \text{mem}_1' \rangle \equiv_{\text{prog}} \langle \text{stop}; \text{mem}_2' \rangle$. In addition, we have $\sigma_1'(ch) = \sigma_2'(ch) = \text{vlst}'$. On this basis, it is not difficult to establish $\sigma_1' \equiv_{\Sigma} \sigma_2'$.

- Suppose $NE \notin as$. By (23), we have $\text{lev}(ch, as) = \emptyset$. Hence $ch \in \text{PubCh}$ or $ch \in \text{EncCh}$.

  By (27), we have $|\sigma_2(ch)| > 0$. Thus we have $|\sigma_2(ch)| > 0$ by $\sigma_1 \equiv_{\Sigma} \sigma_2$ and the possible classes of $ch$. Hence there exists some $\sigma_2'$ and $\text{mem}_2'$ such that

  $$\langle \langle \text{as}_{\text{recv}}(ch, x); \text{mem}_2'; \sigma_1 \rangle \rangle \rightarrow \langle \langle \text{stop}; \text{mem}_2'; \sigma_2' \rangle \rangle$$
The remaining proof obligations can be discharged by a case analysis on whether \( ch \in PubCh \) or \( ch \in EncCh \), analogously to the case where \( NE \in as \).

**Sub-case** \( ch \in ICh \). We make a further case analysis on whether \( NE \in as \).

- Suppose \( NE \in as \).
  
  By \([26]\), we have \( |\sigma_2(ch)| > 0 \). Hence there exist some \( \sigma'_2 \) and \( \text{mem}'_2 \) such that
  \[
  \langle \langle \text{recv}(ch, x); \text{mem}_2 \rangle; \sigma_2 \rangle \rightarrow \langle \langle \text{stop}; \text{mem}'_2 \rangle; \sigma'_2 \rangle
  \]
  Since no message queues for the external channels are affected in \( \sigma'_1 \) or \( \sigma'_2 \), we trivially have \( \sigma'_1 \simeq_{\Sigma} \sigma'_2 \).

  We make a further case analysis on whether \( L^o \) and \( L^* \) are in \( as \), to show \( \langle \text{stop}; \text{mem}'_1 \rangle R_{prog} \langle \text{stop}; \text{mem}'_2 \rangle \), which boils down to showing \( \text{mem}'_1 =_L \text{mem}'_2 \).

  - Suppose \( L^* \not\in as \) or \( L^o \not\in as \).
    
    From \([24]\), we have \( \text{lev}(x) = \emptyset \). Hence it vacuously holds that \( \text{lev}(x) = L \Rightarrow \text{mem}'_1(x) = \text{mem}'_2(x) \). Thus we have \( \text{mem}'_1 =_L \text{mem}'_2 \) from \( \text{mem}_1 =_L \text{mem}_2 \), and the fact that the values of the other variables are unchanged in \( \text{mem}'_1 \) and \( \text{mem}'_2 \).

  - Suppose \( L^* \in as \) and \( L^o \in as \).
    
    By \([26]\) and \( NE \in as \), we have \( |\sigma_1(ch)| > 0 \), and \( |\sigma_2(ch)| > 0 \). Hence we can obtain \( \text{first}(\sigma_1(ch)) = \text{first}(\sigma_2(ch)) \) using \([25]\). Hence we can derive \( \text{mem}'_1(x) = \text{mem}'_2(x) \). On this basis, it is not difficult to derive \( \text{mem}'_1 =_L \text{mem}'_2 \).

- Suppose \( NE \not\in as \).

  Using \([23]\), we derive \( L^o \in as \). By \([27]\), we have \( |\sigma_1(ch)| > 0 \). Thus we have \( |\sigma_2(ch)| > 0 \) by \([25]\). Hence there exist some \( \sigma'_2 \) and \( \text{mem}'_2 \) such that
  \[
  \langle \langle \text{recv}(ch, x); \text{mem}_2 \rangle; \sigma_2 \rangle \rightarrow \langle \langle \text{stop}; \text{mem}'_2 \rangle; \sigma'_2 \rangle
  \]
  Since no message queues for the external channels are affected in \( \sigma'_1 \) or \( \sigma'_2 \), we trivially have \( \sigma'_1 \simeq_{\Sigma} \sigma'_2 \).

  We make a case analysis on whether \( L^* \in as \) to show \( \langle \text{stop}; \text{mem}'_1 \rangle R_{prog} \langle \text{stop}; \text{mem}'_2 \rangle \), which boils down to showing \( \text{mem}'_1 =_L \text{mem}'_2 \).

  - Suppose \( L^* \not\in as \).
    
    By \([24]\), we have \( \text{lev}(x) = \emptyset \). Hence we have \( \text{lev}(x) = L \Rightarrow \text{mem}'_1(x) = \text{mem}'_2(x) \). On this basis, \( \text{mem}'_1 =_L \text{mem}'_2 \) can be derived.

  - Suppose \( L^* \in as \).
    
    By \([25]\), \( |\sigma_1(ch)| > 0 \), and \( |\sigma_2(ch)| > 0 \), we have \( \text{first}(\sigma_1(ch)) = \text{first}(\sigma_2(ch)) \). Hence, we have \( \text{mem}'_1(x) = \text{mem}'_2(x) \). On this basis, \( \text{mem}'_1 =_L \text{mem}'_2 \) can be derived.

**Case** \( \text{as-if-rev}(ch, x, x_b) \). The reasoning can be structured similarly to the case for \( \text{as-rev}(ch, x) \).

For each of the **composite cases**, we construct a specific relation that is shown to be an assumption-aware bisimulation, which serves to justify the local security of the corresponding program.
Case if $e$ then $\text{prog}_a$ else $\text{prog}_b$ fi. We construct the following relation where if is a shorthand for the program if $e$ then $\text{prog}_a$ else $\text{prog}_b$ fi.

$$R_d = \{(\text{if}_d; \text{mem}_1), (\text{if}_d; \text{mem}_2) \mid \text{mem}_1 =_L \text{mem}_2\} \cup \approx$$

For all $\text{mem}_1$, $\text{mem}_2$ such that $\text{mem}_1 =_L \text{mem}_2$, we have $(\text{if}_d; \text{mem}_1) R_d (\text{if}_d; \text{mem}_2)$. We proceed to show that $R_d$ is an assumption-aware bisimulation. It is obvious that $R_d$ is symmetric. Pick an arbitrary pair $p = ((\text{prog}_1; \text{mem}_1), (\text{prog}_2; \text{mem}_2))$ from $R_d$. We can also derive $\text{mem}_1 =_L \text{mem}_2$ and $\text{prog}_1 = \text{stop} \Rightarrow \text{prog}_2 = \text{stop}$.

To discharge the remaining proof obligations, we make a case analysis on which part of $R_d p$ belongs to. We assume the following hypotheses.

$$\text{LSec}(\text{prog}_a)$$

$$\text{LSec}(\text{prog}_b)$$

$$\text{lev}(e) = \mathbb{I} \Rightarrow \text{prog}_a \sim \text{prog}_b$$

Sub-case $p \in \{(\text{if}_d; \text{mem}_1), (\text{if}_d; \text{mem}_2) \mid \text{mem}_1 =_L \text{mem}_2\}$. We have $\text{prog}_1 = \text{prog}_2 = \text{if}$.

Pick arbitrary channel states $\sigma_1$ and $\sigma_2$ such that $\sigma_1 \simeq \Sigma, \sigma_1 =_L \sigma_2, \sigma_1 =_L \sigma_2, \sigma_1 \times^{ne} \emptyset$, and $\sigma_2 \times^{ne} \emptyset$.

Suppose without loss of generality that

$$\langle (\text{if}_d; \text{mem}_1); \sigma_1 \rangle \Rightarrow \langle (\text{prog}_a; \text{mem}_1); \sigma_1 \rangle$$

We make a case analysis on whether $\llbracket e \rrbracket_{\text{mem}_2}^{\text{mem}_1}$ is equal to $\llbracket e \rrbracket_{\text{mem}_1}$.

- Suppose $\llbracket e \rrbracket_{\text{mem}_2}^{\text{mem}_1} \neq \llbracket e \rrbracket_{\text{mem}_1}$. We have

$$\langle (\text{if}_d; \text{mem}_2); \sigma_2 \rangle \Rightarrow \langle (\text{prog}_b; \text{mem}_2); \sigma_2 \rangle$$

From $\text{mem}_1 =_L \text{mem}_2$, we have $\text{lev}(e) = \mathbb{I}$. Hence we have $\text{prog}_a \sim \text{prog}_b$ by $\llbracket 30 \rrbracket$. Hence we have $\langle \text{prog}_a; \text{mem}_1 \rangle \approx \langle \text{prog}_b; \text{mem}_2 \rangle$, which gives $\langle \text{prog}_a; \text{mem}_1 \rangle R_d \langle \text{prog}_b; \text{mem}_2 \rangle$. We also have $\sigma_1 \simeq \Sigma, \sigma_2$, which is preserved from before.

- Suppose $\llbracket e \rrbracket_{\text{mem}_2}^{\text{mem}_1} = \llbracket e \rrbracket_{\text{mem}_1}$. We have

$$\langle (\text{if}_d; \text{mem}_2); \sigma_2 \rangle \Rightarrow \langle (\text{prog}_a; \text{mem}_2); \sigma_2 \rangle$$

By $\llbracket 28 \rrbracket$ and $\text{mem}_1 =_L \text{mem}_2$, we have $\langle \text{prog}_a; \text{mem}_1 \rangle \approx \langle \text{prog}_a; \text{mem}_2 \rangle$, which directly gives $\langle \text{prog}_a; \text{mem}_1 \rangle R_d \langle \text{prog}_a; \text{mem}_2 \rangle$. We also have $\sigma_1 \simeq \Sigma, \sigma_2$, which is preserved from before.

Sub-case $p \in \approx$. Straightforward from the fact that $\approx$ is an assumption-aware bisimulation.

Case while $e$ do $\text{prog}_a$ od. We construct the relation $R_{\text{while}} = R_0 \cup R_1 \cup R_2$, where while is a shorthand for while $e$ do $\text{prog}_a$ od, and

$$R_0 = \{\langle \text{while}_1; \text{mem}_1 \rangle, \langle \text{while}_1; \text{mem}_2 \rangle \mid \text{mem}_1 =_L \text{mem}_2\}$$

$$R_1 = \{\langle \text{prog}_1; \text{while}_1 \rangle, \langle \text{prog}_2; \text{while}_2 \rangle \mid \langle \text{prog}_1; \text{mem}_1 \rangle \approx \langle \text{prog}_2; \text{mem}_2 \rangle\}$$

$$R_2 = \{\langle \text{stop}; \text{mem}_1 \rangle, \langle \text{stop}; \text{mem}_2 \rangle \mid \text{mem}_1 =_L \text{mem}_2\}$$

For all $\text{mem}_1$ and $\text{mem}_2$ such that $\text{mem}_1 =_L \text{mem}_2$, we have $\langle \text{while}; \text{mem}_1 \rangle R_{\text{while}} \langle \text{while}; \text{mem}_2 \rangle$. We proceed to show that $R_{\text{while}}$ is an assumption-aware bisimulation. We have that $R_{\text{while}}$ is
symmetric. Pick an arbitrary pair $p = ((\text{prog}_1; \text{mem}_1), (\text{prog}_2; \text{mem}_2))$ from $\text{R}_{\text{while}}$. We have $\text{mem}_1 =_L \text{mem}_2$, and $\text{prog}_1 \neq \text{stop} \iff \text{prog}_2 \neq \text{stop}$.

To discharge the remaining proof obligations, we make a case analysis on which part of $\text{R}_{\text{while}}$ $p$ belongs to. We assume the following hypotheses.

$$L\text{Sec}(\text{prog}_a)$$  \hspace{1cm} (32)

$$\text{lev}(e) = L$$  \hspace{1cm} (33)

**Sub-case** $p \in R_0$. Pick arbitrary channel states $\sigma_1$ and $\sigma_2$ such that $\sigma_1 \simeq \Sigma \sigma_2$, $\sigma_1 = \emptyset$, $\sigma_1 \leftarrowNE \emptyset$, and $\sigma_2 \leftarrowNE \emptyset$.

We make a further case analysis on whether $\text{while}$ continues or terminates in one step.

- Suppose $((\text{while}; \text{mem}_1); \sigma_1) \rightarrow ((\text{prog}_a; \text{while}; \text{mem}_1); \sigma_1)$. By (33) and $\text{mem}_1 =_L \text{mem}_2$, we have $\llbracket e \rrbracket_{\text{mem}_2} = \llbracket e \rrbracket_{\text{mem}_1}$. Hence we have

  $$((\text{while}; \text{mem}_2); \sigma_2) \rightarrow ((\text{prog}_a; \text{while}; \text{mem}_2); \sigma_2)$$

  By (32) and $\text{mem}_1 =_L \text{mem}_2$, we have $(\text{prog}_a; \text{mem}_1) \approx (\text{prog}_a; \text{mem}_2)$. Hence, we have $(\text{prog}_a; \text{while}; \text{mem}_1) \text{R}_{\text{while}} (\text{prog}_a; \text{while}; \text{mem}_2)$. We also have $\sigma_1 \simeq \Sigma \sigma_2$, which is trivially preserved.

- Suppose $((\text{while}; \text{mem}_1); \sigma_1) \rightarrow ((\text{stop}; \text{mem}_1); \sigma_1)$. This case is straightforward.

**Sub-case** $p \in R_1$. We have $\text{prog}_1 = \text{prog}_{10}; \text{while}$, and $\text{prog}_2 = \text{prog}_{20}; \text{while}$ for some $\text{prog}_{10}$ and $\text{prog}_{20}$ such that

$$\langle \text{prog}_{10}; \text{mem}_1 \rangle \approx \langle \text{prog}_{20}; \text{mem}_2 \rangle$$  \hspace{1cm} (34)

Pick arbitrary channel states $\sigma_1$ and $\sigma_2$ such that

$$\sigma_1 \simeq \Sigma \sigma_2 \land \sigma_1 \leftarrowNE \text{asm-of}(\text{prog}_{10}) \land \sigma_2 \leftarrowNE \text{asm-of}(\text{prog}_{20})$$  \hspace{1cm} (35)

We make a further case analysis on whether $\text{prog}_{10}$ continues or terminates after one step.

- Suppose $((\text{prog}_{10}; \text{while}; \text{mem}_1); \sigma_1) \rightarrow ((\text{prog}_{10}; \text{while}; \text{mem}_1'); \sigma_1')$, with $(\langle \text{prog}_{10}; \text{mem}_1 \rangle; \sigma_1) \rightarrow (\langle \text{prog}_{10}; \text{mem}_1'; \sigma_1' \rangle$, and $\text{prog}_{10} \neq \text{stop}$. By (34) and (35), there exist $\text{prog}_{20}$, $\text{mem}_2$, and $\sigma_2'$ such that $(\langle \text{prog}_{20}; \text{mem}_2 \rangle; \sigma_2) \rightarrow (\langle \text{prog}_{20}; \text{mem}_2'; \sigma_2' \rangle$, $\sigma_1' \simeq \Sigma \sigma_2'$, and $(\text{prog}_{10}; \text{mem}_1') \approx \langle \text{prog}_{20}; \text{mem}_2' \rangle$. Hence, we have $\text{prog}_{10} = \text{stop} \iff \text{prog}_{20} = \text{stop}$, which gives $\text{prog}_{20} \neq \text{stop}$.

Thus, we have $((\text{prog}_{20}; \text{while}; \text{mem}_2); \sigma_2) \rightarrow (\langle \text{prog}_{20}; \text{while}; \text{mem}_2' \rangle; \sigma_2')$, $\sigma_1' \simeq \Sigma \sigma_2'$, and $\langle \text{prog}_{10}; \text{while}; \text{mem}_1' \rangle \text{R}_{\text{while}} (\langle \text{prog}_{20}; \text{while}; \text{mem}_2' \rangle$.

- Suppose $((\text{prog}_{10}; \text{while}; \text{mem}_1); \sigma_1) \rightarrow ((\text{while}; \text{mem}_1'); \sigma_1')$, with $((\text{prog}_{10}; \text{mem}_1); \sigma_1) \rightarrow (\langle \text{stop}; \text{mem}_1'; \sigma_1' \rangle$. By (34) and (35), we know that there exist $\text{prog}_{20}$, $\text{mem}_2$, and $\sigma_2'$ such that $(\langle \text{prog}_{20}; \text{mem}_2 \rangle; \sigma_2) \rightarrow (\langle \text{prog}_{20}; \text{mem}_2'; \sigma_2' \rangle$, $\sigma_1' \simeq \Sigma \sigma_2'$, and $(\text{stop}; \text{mem}_1') \approx \langle \text{prog}_{20}; \text{mem}_2' \rangle$. Hence, we have $\text{mem}_1' =_L \text{mem}_2'$, and $\text{prog}_{20} = \text{stop}$.

Thus, we have $((\text{prog}_{20}; \text{while}; \text{mem}_2); \sigma_2) \rightarrow (\langle \text{while}; \text{mem}_2'; \sigma_2' \rangle$, $\sigma_1' \simeq \Sigma \sigma_2'$, as well as $\langle \text{while}; \text{mem}_1' \rangle \text{R}_{\text{while}} (\langle \text{while}; \text{mem}_2' \rangle$.

**Sub-case** $p \in R_2$. Trivial.
Case $prog_a; prog_b$. We construct the relation $R_i = R_0 \cup \approx$, where
\[
R_0 = \{ \langle \langle prog_{10}; prog_b; mem_1 \rangle, \langle prog_{20}; prog_b; mem_2 \rangle \rangle \mid \langle prog_{10}; mem_1 \rangle \approx \langle prog_{20}; mem_2 \rangle \}.
\]
We assume the following hypotheses
\[
\begin{align*}
LSec(prog_a) \tag{36} \\
LSec(prog_b) \tag{37}
\end{align*}
\]
Pick arbitrary $mem_1$ and $mem_2$ such that $mem_1 \equiv mem_2$. By (36), we have $\langle prog_a; mem_1 \rangle \approx \langle prog_a; mem_2 \rangle$. Hence we have $\langle prog_a; prog_b; mem_1 \rangle \approx \langle prog_a; prog_b; mem_2 \rangle$. We proceed to show that $R_i$ is an assumption-aware bisimulation. It is obvious that $R_i$ is symmetric. Pick arbitrary pair $p = (\langle prog_1; mem_1 \rangle, \langle prog_2; mem_2 \rangle)$ from $R_i$. We have $mem_1 \equiv mem_2$, and $prog_1 = \text{stop} \leftrightarrow prog_2 = \text{stop}$.

We discharge the remaining proof obligations with a case analysis on which part of $R_i$, $p$ belongs.

Sub-case $p \in R_0$. We have $prog_1 = prog_{10}; prog_b$, and $prog_2 = prog_{20}; prog_b$, for some $prog_{10}$ and $prog_{20}$ such that
\[
\langle prog_{10}; mem_1 \rangle \approx \langle prog_{20}; mem_2 \rangle \tag{38}
\]
Pick arbitrary channel states $\sigma_1$ and $\sigma_2$ such that
\[
\begin{align*}
\sigma_1 & \equiv \sigma_2 \land \sigma_1 \equiv \text{asm-of}(prog_{10}) \land \text{asm-of}(prog_{20}) \\
& \land \sigma_1 \equiv \text{LSec-of}(prog_{10}) \land \sigma_2 \equiv \text{LSec-of}(prog_{20}) \tag{39}
\end{align*}
\]
We make a case analysis on whether $prog_{10}$ is exhausted in the next step of $prog_{10}; prog_b$.

– Suppose for some $prog'_{10}$, $mem'_1$, and $\sigma'_1$, we have
\[
\langle \langle prog_{10}; prog_b; mem_1 \rangle; \sigma_1 \rangle \rightarrow \langle \langle prog'_{10}; prog_b; mem'_1 \rangle; \sigma'_1 \rangle
\]
with $\langle \langle prog_{10}; mem_1 \rangle; \sigma_1 \rangle \rightarrow \langle \langle prog'_{10}; mem'_1 \rangle; \sigma'_1 \rangle$, and $prog'_{10} \neq \text{stop}$.

By (38), and (39), there exist $prog'_{20}$, $mem'_2$, $\sigma'_2$ such that
\[
\langle \langle prog_{20}; mem_2 \rangle; \sigma_2 \rangle \rightarrow \langle \langle prog'_{20}; mem'_2 \rangle; \sigma'_2 \rangle
\]
$\sigma'_1 \equiv \sigma'_2$, and $\langle prog'_{10}; mem'_1 \rangle \approx \langle prog'_{20}; mem'_2 \rangle$. Hence $prog'_{10} = \text{stop} \leftrightarrow prog'_{20} = \text{stop}$. Hence $prog'_{20} \neq \text{stop}$.

Thus, we have $\langle \langle prog_{20}; prog_b; mem_2 \rangle; \sigma_2 \rangle \rightarrow \langle \langle prog'_{20}; prog_b; mem'_2 \rangle; \sigma'_2 \rangle$, $\sigma'_1 \equiv \sigma'_2$, and $\langle prog'_{10}; prog_b; mem'_1 \rangle \approx \langle prog'_{20}; prog_b; mem'_2 \rangle$. We proceed to show

– Suppose for some $mem'_1$, and $\sigma'_1$, we have
\[
\langle \langle prog_{10}; prog_b; mem_1 \rangle; \sigma_1 \rangle \rightarrow \langle \langle stop; mem'_1 \rangle; \sigma'_1 \rangle
\]
with $\langle \langle prog_{10}; mem_1 \rangle; \sigma_1 \rangle \rightarrow \langle \langle stop; mem'_1 \rangle; \sigma'_1 \rangle$.

By (38), and (39), there exist $prog'_{20}$, $mem'_2$, $\sigma'_2$ such that
\[
\langle \langle prog_{20}; mem_2 \rangle; \sigma_2 \rangle \rightarrow \langle \langle prog'_{20}; mem'_2 \rangle; \sigma'_2 \rangle
\]
$\sigma'_1 \equiv \sigma'_2$, and $\langle stop; mem'_1 \rangle \approx \langle prog'_{20}; mem'_2 \rangle$. Hence $mem'_1 \equiv mem'_2$, and $prog'_{20} = \text{stop}$.

Hence we have $\langle \langle prog_{20}; prog_b; mem_2 \rangle; \sigma_2 \rangle \rightarrow \langle \langle prog_b; mem'_2 \rangle; \sigma'_2 \rangle$. By (37) and $\equiv mem'_1 \equiv mem'_2$, we also have $\langle prog_b; mem'_1 \rangle \approx \langle prog_b; mem'_2 \rangle$. Thus $\langle prog_b; mem'_1 \rangle R_i \langle prog_b; mem'_2 \rangle$. 16
Sub-case \( p \in \approx \). Trivial.

The above case analysis on the pairs of programs with holes and conditions completes the proof. \( \square \)

We next prove Theorem 2.

**Theorem 2.** If \( \text{lev} \vdash \text{prog} \), then \( \text{LSec}(\text{prog}) \).

**Proof.** The proof is by induction on the derivation of \( \text{lev} \vdash \text{prog} \). In each case we build on the hook-up property of \( \text{LSec}(-) \) (Proposition 5) to obtain the desired result. We only present one base case and one inductive case below.

**Case** \( \text{as recv}(ch, x) \): By \( \text{lev} \vdash \text{as recv}(ch, x) \), we have

\[
\begin{align*}
\text{NE} \notin \text{as} & \Rightarrow \text{lev}^0(ch, as) = L \quad (40) \\
\text{lev}^0(ch, as) \sqcup \text{lev}^*(ch, as) & \subseteq \text{lev}(x) \quad (41)
\end{align*}
\]

Instantiating Proposition 3 with \( n = 0 \), and the fourth pair of context and condition, we know that \( \text{LSec}(\text{as recv}(ch, x)) \) holds, if \( \text{(NE} \notin \text{as} \Rightarrow \text{lev}^0(ch, as) = L) \land \text{lev}^0(ch, as) \sqcup \text{lev}^*(ch, as) \subseteq \text{lev}(x) \), which is ensured by (40), (41).

**Case** if \( e \) then \( \text{prog}_1 \) else \( \text{prog}_2 \) fi: By \( \text{lev} \vdash \text{if} e \text{ then } \text{prog}_1 \text{ else } \text{prog}_2 \text{ fi} \), we have

\[
\begin{align*}
\text{lev} \vdash \text{prog}_1 & \quad (42) \\
\text{lev} \vdash \text{prog}_2 & \quad (43) \\
\text{lev}(e) = \mathbb{H} & \Rightarrow \text{prog}_1 \sim \text{prog}_2 \quad (44)
\end{align*}
\]

Instantiating Proposition 3 with \( n = 2 \), the sixth pair of context and condition, \( \text{prog}_1 \), and \( \text{prog}_2 \), we have \( \text{LSec}(\text{if} e \text{ then } \text{prog}_1 \text{ else } \text{prog}_2 \text{ fi}) \) holds if \( \text{lev}(e) = \mathbb{H} \Rightarrow \text{prog}_1 \sim \text{prog}_2 \), \( \text{LSec}(\text{prog}_1) \), and \( \text{LSec}(\text{prog}_2) \). Using the induction hypothesis on (42) and (43), we can derive \( \text{LSec}(\text{prog}_1) \) and \( \text{LSec}(\text{prog}_2) \). Combining this with (44), it is not difficult to see that all the conditions for \( \text{LSec}(\text{if} e \text{ then } \text{prog}_1 \text{ else } \text{prog}_2 \text{ fi}) \) to be established are satisfied.

In the remaining cases, the premises of the typing rules give rise to the premises of the corresponding statements in the hook-up property in an analogous fashion to the cases above. The induction completes the proof of this theorem. \( \square \)
\[
\begin{align*}
\text{NE} \not\in as \Rightarrow & \quad \text{lev}^\circ(ch, as) = \bot \\
\text{lev}^\circ(ch, as) \sqcup \text{lev}^*\text{rec}(ch, x) \leq & \quad \text{lev}(x) \\
\text{lev}^\circ(ch, as) \sqcup & \quad \text{lev}^*\text{if-recv}(ch, x, x_b) \\
\text{lev}(e) \subseteq & \quad \text{lev}^\circ(ch, \emptyset) \\
\text{lev}(e) \subseteq & \quad \text{lev}(x) \\
\text{lev} \vdash S & \quad \text{send}(ch, e) \\
\text{lev} \vdash S & \quad x := e \\
\text{lev} \vdash S & \quad \text{skip}
\end{align*}
\]

Fig. 1: The Fully Syntactic Security Type System

3 Fully Syntactic Security Type System

We present a security type system that is sound wrt. local security, yet has no semantic side condition. This type system establishes the judgment \( \text{lev} \vdash_S \text{prog} \), saying that the program \( \text{prog} \) is (fully syntactically) typable in the environment \( \text{lev} \).

The typing rules are presented in Fig. 1. Except for the if rule, all the rules in Fig. 1 are identical to the rules of our original type system (in Fig. 1 of [1]) for the same language constructs. For the if rule, the side condition \( \text{lev}(e) = \bot \Rightarrow \text{slice-of}(\text{prog}_1) = \text{slice-of}(\text{prog}_2) \) is used to replace the semantic side condition \( \text{lev}(e) = \bot \Rightarrow \text{prog}_1 \sim \text{prog}_2 \) in our original type system. Here, the function \( \text{slice-of} : \text{Prog} \rightarrow \text{LS} \cup \{\perp\} \) gives the slice of each program (defined in Fig. 2). The set \( \text{LS} \) of low slices is the same as the set of programs, except that the set of expressions is extended with the special element \( \circ \), which represents an expression that is not contained in a low slice. Intuitively, the low slice of a program is a slice that captures the effects on the low parts of the memory, and the public content or presence of messages over communication channels, created by executing the program. Hence, the side condition \( \text{lev}(e) = \bot \Rightarrow \text{slice-of}(\text{prog}_1) = \text{slice-of}(\text{prog}_2) \) captures the requirement that the execution of the two branches of the if should have the same effects on the public parts of the memory and channel states.

We have the following theoretical results about the syntactic security type system.

**Theorem 3.** If \( \text{lev} \vdash_S \text{prog} \), then \( \text{lev} \vdash \text{prog} \).

**Corollary 1.** If \( \text{lev} \vdash_S \text{prog} \), then \( \text{LSec}(\text{prog}) \).

The theorem says that the fully syntactic security type system is sound wrt. the security type system presented in Fig. 1 of the paper. The corollary says that the fully syntactic security type system is sound wrt. local security. Hence, the fully syntactic security type system can be used as a replacement of the security type system presented in the paper to the benefit of fully operational type checking, and verification of local security.

**Theorem 3** can be shown with a straightforward structural induction on \( \text{prog} \) once the following lemma is shown. **Corollary 1** then follows immediately from **Theorem 2** and **Theorem 3**.
Lemma 9. If lev ⊢ prog₁, lev ⊢ prog₂, and low-slice-of(prog₁) = low-slice-of(prog₂), then prog₁ ∼ prog₂.

This lemma can be shown with a structural induction on prog₁, using the two following lemmas in the case for if. The proofs of these two lemmas are straightforward and omitted here.

Lemma 10. If lev ⊢ prog, low-slice-of(prog) = skip, σ × asm-of(prog), then for all mem, there exist mem’ and σ’, such that ⟨⟨prog; mem⟩; σ⟩ → ⟨⟨stop; mem’⟩; σ’⟩.

Lemma 11. If lev ⊢ prog, low-slice-of(prog) = skip, and ⟨⟨prog; mem⟩; σ⟩ → ⟨⟨prog’; mem’⟩; σ’⟩, then the following statements hold:
1. ∀x ∈ Var : mem’(x) ≠ mem(x) ⇒ lev(x) = H,
2. ∀ch ∈ ECh : σ’(ch) ≠ σ(ch) ⇒ ch ∈ PriCh.
4 Typability of the Auction Example and Authentication Example

Type-Checking Using the Type System from the Paper. As outlined in the paper, the authentication example and the auction example can successfully be type-checked using the type system presented in Fig. 1 of the paper.

While for the authentication example the semantic condition in the premise of the if-rule is not used in the type-checking, the auction example contains two high if-branches where the semantic condition is used. We provide the bisimulation relations that can be used to established the semantic condition for the two if-branches, respectively.

Client Program. With the symmetric closure of the following relation the conditional branching in auct-cl can be type-checked.

\[\{(\text{send}(\text{int}_1, \text{calc}(\text{min}, \text{thres})); \text{mem}_1), \langle \text{skip}; \text{mem}_2 \rangle) \mid \text{mem}_1 =_L \text{mem}_2\} \cup \{((\text{stop}; \text{mem}_1), (\text{stop}; \text{mem}_2)) \mid \text{mem}_1 =_L \text{mem}_2\}\]

Server Program. With the symmetric closure of the following relation the conditional branching in auct-srv can be type-checked.

\[\{(\text{send}(\text{pri}, \text{bid}); \text{mem}_1), \langle \text{skip}; \text{mem}_2 \rangle) \mid \text{mem}_1 =_L \text{mem}_2\} \cup \{((\text{stop}; \text{mem}_1), (\text{stop}; \text{mem}_2)) \mid \text{mem}_1 =_L \text{mem}_2\}\]

Type-Checking Using the Fully-Syntactic Type System of this Addendum. The fully-syntactic type system provided in Sect. 3 of this addendum allows one to type-check both the auction example and the authentication example without constructing bisimulation relations.

For the authentication example, the type-checking can be conducted analogously to the type-checking using the type system from the paper. This is because neither the client program nor the server program contains a high conditional branching. Hence, the semantic condition and also its syntactic replacement are not used in the type-checking.

For the auction example, the type-checking of each high conditional branching can be conducted by a comparison of the low-slices of the two branches. Since for each high conditional branching these two low-slices are equal, both the client program and the server program can be successfully typed.

We give the computed low-slices below.

Client Program. For the client program, the relevant low-slices are the following:

\[\text{low-slice-of} (\text{send}(\text{int}_1, \text{calc}(\text{min}, \text{thres}))) = \text{skip}\]

\[\text{low-slice-of} (\text{skip}) = \text{skip}\]

Server Program. For the server program, the relevant low-slices are the following:

\[\text{low-slice-of} (\text{send}(\text{pri}, \text{bid})) = \text{skip}\]

\[\text{low-slice-of} (\text{skip}) = \text{skip}\]

It is worth mentioning that all the example programs from Sect. 2 of our paper [1] are also typable using the fully syntactic security type system.

References

1. Ximeng Li, Heiko Mantel, and Markus Tasch. Taming message-passing communication in compositional reasoning about confidentiality. In 15th Asian Symposium on Programming Languages and Systems (APLAS), 2017. Accepted for publication.