Soundness proofs for the DSD type system
Robert Grabowski

Abstract
This paper presents the soundness proofs for the type system of the Dynamic Security Domains (DSD) language.

Unless otherwise noted, the identifiers and indices used in the proofs directly refer the respective identifiers and indices used in the operational semantics and the typing rules. Also, $\sigma$ and $(s, h)$ shall refer to the same program state, equally $\sigma_1 = (s_1, h_1), \sigma' = (s', h')$, etc.

1 Expression typing soundness

In the following, we simplify notion and ignore the class information of objects. That is, given a heap $h$ and a location $a$, $h(a)$ shall refer directly to the field valuation.

First, we observe an equality between the evaluation of a qualified field type (which is a label) and the interpretation of a field type of the same object. The lemma follows from their definitions.

Lemma 1. In any state $\sigma$, it holds $\mathcal{J}_{ft}(f)_{\sigma} = \mathcal{J}_{ft}(f)_{\sigma} \circ \mathcal{J}_{\pi}(\ell)_{\sigma}$.

The following lemma states that the expression typing rules are sound.

Lemma 2. If $\Gamma \vdash e : \ell$, then for all states $\sigma$ and $\sigma'$ and all partial bijections $\beta$ and domains $k$ such that $\sigma \sim_\beta \sigma'$, $[\ell]_{\sigma} \leq k$ implies $\mathcal{J}_{\ell} \sim_\beta \mathcal{J}_{e}$.

Proof. By induction over $e$.

- $e = n$ or $e = \top$ or $e = \bot$. Then $e$ is a constant and $[e]_{\sigma} \sim_\beta [e]_{\sigma'}$ holds.

- $e = x$. Then $\ell = \Gamma(x)$. If $[\Gamma(x)]_{\sigma} \leq k$, then by definition $s(x) \sim_\beta s'(x)$, hence $[x]_{\sigma} \sim_\beta [x]_{\sigma'}$.

- $e = \pi.f$. $\Gamma \vdash \pi : \ell_\pi$ and $\ell = \mathcal{ft}(f) \sqcup \ell_\pi$. Since $[\ell]_{\sigma} \leq k$, we get $[\ell_\pi]_{\sigma} \leq k$ and by induction $[\pi]_{\sigma} \sim_\beta [\pi]_{\sigma'}$. We define $a = [\pi]_{\sigma}$ and $a' = [\pi]_{\sigma'}$. From $a \sim_\beta a'$ follows $h(a) \sim_{h'(a') \ell} h'(a')$. With lemma 1, we know $[\mathcal{ft}(f)]_{\sigma} = [\mathcal{ft}_\pi(f)]_{\sigma}$, which is lower than $k$ since $[\ell]_{\sigma} \leq k$. With the definition of object equivalence, we have $h(a)(f) \sim_{h'(a')(f)} h'(a')(f)$ and thus $[\pi.f]_{\sigma} \sim_\beta [\pi.f]_{\sigma'}$. 

$\square$
The next lemma states meta-label monotonicity.

**Lemma 3.** If $\Gamma \vdash e : \ell$ and $\Gamma \vdash e : \ell'$, then $[\ell']_\sigma \leq [\ell]_\sigma$ for any state $\sigma$.

**Proof.** By induction over $e$.

- $e = n$ or $e \in \{\top, \bot\}$. Then $\ell = \bot$, hence $\ell' = \bot$, so $\ell'$ evaluates to $L$ and the lemma holds.

- $e = x$. Then $\ell \in \{x, \top, \bot\}$, thus $\ell' = \bot$, so $\ell'$ evaluates to $L$ and the lemma holds.

- $e = \pi.f$. Then $\ell = \text{ft}_\pi(f) \sqcup \ell_\pi$. For $\ell_\pi$, we derive by induction $[\ell'_\pi]_\sigma \leq [\ell_\pi]_\sigma$.
  - If $\text{ft}_\pi(f) \in \{\bot, \top\}$, then $\ell' = \bot \sqcup \ell'_\pi = \ell'_\pi$, and $[\ell'_\pi]_\sigma \leq [\ell_\pi]_\sigma \leq [\ell]_\sigma$.
  - If $\text{ft}_\pi(f) = \pi.f_\delta$, then $\ell = \ell_\pi \sqcup \bot \sqcup \ell'_\pi$, which is lower or equal $\ell_\pi$ and thus $\ell$.

- $e = e_1 \circ e_2$. Then $\ell = \ell_1 \sqcup \ell_2$ and $\ell' = \ell'_1 \sqcup \ell'_2$. By induction $[\ell'_1]_\sigma \leq [\ell_1]_\sigma$ and $[\ell'_2]_\sigma \leq [\ell_2]_\sigma$, so the lemma holds.

For the soundness proofs of program typing rules, we need a corollary that follows directly from meta-label monotonicity (lemma 3) and soundness of expressions (lemma 2).

**Corollary 4.** If $\Gamma \vdash e : \ell$, then for all states $\sigma$ and $\sigma'$ and all partial bijections $\beta$ and domains $k$ such that $\sigma \sim_\beta k \sigma'$, $[\ell]_\sigma \leq k$ implies $[\ell]_\sigma = [\ell']_\sigma$.

## 2 Program typing soundness

In the paper, the soundness results where first presented for DSD without methods. Here, we prove soundness directly for the type system with methods, hence we need to include in every lemma the requirement that all methods are well-typed with respect to their signatures.

**Lemma 5.** Let $\Gamma, pc \vdash Q \{P\} Q'$ and $\sigma \xrightarrow{P} \sigma'$, and let all methods be well-typed with respect to their signatures. Then:

1. $[pc_\ast]_\sigma = [pc_\ast]_{\sigma'}$,
2. $[pc_h]_\sigma = [pc_h]_{\sigma'}$,
3. if $\sigma \models Q$, then $\sigma' \models Q'$.
Proof of 1 and 2. By induction over the operational semantics. For \textbf{skip}, \textbf{if}, \textbf{while} and sequence statements, the induction hypothesis is used. For all assignment statements, the premises $x \not\in pc$ and $f \not\in pc$ ensure that the program counter label is not changed. For \textbf{call}, the we apply the induction hypothesis on the method body.

Proof of 3. We observe that with the given interpretation (satisfiability) of constraint sets, $Q$ are unary state predicates, $\cup$ works as a conjunction and that $\Rightarrow$ is indeed an implication. It is easy to see that the pre- and post-sets can be derived in Hoare logic using the consequence rule appropriately. There are two cases that need to be examined closer:

- For the \textbf{call} rule, the induction hypothesis can be applied, because it can be shown that if $\sigma$ satisfies $Q[p/\text{margs}(m)][\pi/\text{this}]$ (where $Q$ is the required constraint set of the method), then the initial state $\hat{\sigma}$ of the method satisfies $Q$. From the induction hypothesis and from well-typedness we get that the final state $\hat{\sigma}'$ of the method satisfies the ensured constraint set $Q'$, and it is easy to see that $Q'[x/\text{ret}]$ holds in $\sigma'$, because the $\text{ret}$ variable is assigned to $x$.

- For the \textbf{new} statement, the pre-condition $x.f \subseteq_Q Q' \bot$ holds in any state $\sigma$ and for any constraint set $Q$, as the $f$ field of new objects is initialised with $L$, i.e. $\llbracket \bot \rrbracket_\sigma$.

In the following, the notation $(s,h) \sim_{id}^k (s',h')$ refers to a bijection $id = \{(a,a) \mid a \in \text{dom}(h) \cap \text{dom}(h')\}$, i.e. the identity relation ranging over the addresses on which both heaps are defined. (Usually, $h'$ is a post-heap and thus a proper extension of $h$, the pre-heap, so the identity is defined on the addresses that already existed in $h$.)

The following lemma states that no data at a domain below or disjoint from the value of $pc$ is written. We implicitly assume the state equivalences with respect to the given type environment $\Gamma$.

**Lemma 6.** Let $\Gamma, pc \vdash Q \{P\} Q'$ and $(s_1, h_1) \xrightarrow{P} (s_2, h_2)$, where $\sigma_1 = (s_1, h_1)$ and $\sigma_2 = (s_2, h_2)$. Let $\sigma_1 \models Q$, and let all methods be well-typed with respect to their signature. Then:

- If $[pc]_{s_1,h_1} \not\leq_k k$, then $s_1 \sim_{id}^k s_2$.
- If $[pc]_{s_1,h_1} \not\leq_k k$, then $h_1 \sim_{id}^k h_2$.

**Proof.** By induction over the derivation of the operational semantics.

Without loss of generality, we assume $\Gamma, pc \vdash Q \{P\} Q'$ has not been derived by the subsumption rule. If it has, then there is a judgement $\Gamma, pc \vdash Q_0 \{P\} Q_0'$ that has not been derived by the subsumption rule. Since $Q \Rightarrow Q_0$, $\sigma_1$ also satisfies $Q_0$, and we can show the theorem on that judgement.

3
• $P = \text{skip}$. We have to show $s_1 \sim_{id}^k s_1$ (or $h_1 \sim_{id}^k h_1$), which follows from definition.

• $P = P_1 ; P_2$. Then we have $\sigma_1 \xrightarrow{P_1} \sigma_2$ and $\sigma_2 \xrightarrow{P_2} \sigma_3$ from the operational semantics and $\Gamma, pc \vdash Q \{P_1\} Q'$ and $\Gamma, pc \vdash Q' \{P_2\} Q''$ from the type rules. Store equivalence: By induction and since by lemma 5 the pc labels have not changed and $Q'$ holds in $\sigma_2$, we get $s_1 \sim_{id}^k s_2$ and $s_2 \sim_{id}^k s_3$. Transitivity of equivalence results in $s_1 \sim_{id}^k s_3$. Heap equivalence is shown in a very similar way.

• $P = \text{if } e \text{ then } P_1 \text{ else } P_2$.

  Since $[pc_\ell]_{\sigma_1} \not\subseteq k$, we know $[pc_\ell \cup \ell]_{\sigma_1} \not\subseteq k$ for any label $\ell$. The same holds for $pc_b$. Let $P_i$ be the subprogram $P_1$ or $P_2$ depending on the value of $[e]_{\sigma_1}$. Then by induction on $P_i$, $\sigma_1 \sim_{id}^k \sigma_2$. (When the label test rule is applied, we additionally have to show $\sigma_1$ satisfies $Q, \ell_1 \leq \ell_2$ if the “then” branch is taken, but this follows from the operational semantics.)

• $P = \text{while } e \text{ do } P$.

  If $[pc_\ell]_{\sigma_1} \not\subseteq k$, we know $[pc_\ell \cup \ell]_{\sigma_1} \not\subseteq k$ for any label $\ell$. The same holds for $pc_b$.

    - Case $[e]_{\sigma_1} = 0$: $s_1 \sim_{id}^k s_1$ (or $h_1 \sim_{id}^k h_1$) holds trivially.
    - Case $[e]_{\sigma_1} \neq 0$: We have $\sigma_1 \xrightarrow{P} \sigma_2$ and $\Gamma, pc \cup \ell \vdash Q \{P\} Q$, hence by induction $\sigma_1 \sim_{id}^k \sigma_2$. Also, we have $\sigma_2 \xrightarrow{\text{while } e \text{ do } P} \sigma_3$ and the original $\Gamma, pc \vdash Q \{\text{while } e \text{ do } P\} Q$. Again by induction and with lemma 5, $\sigma_2 \sim_{id}^k \sigma_3$. Transitivity gives $\sigma_1 \sim_{id}^k \sigma_3$.

• $P = x := e$. Store equivalence: We get $s_2 = s_1[x \mapsto [e]_{\sigma_1, h_1}]$. Since $\ell \cup pc_\ell \sqsubseteq_Q \Gamma(x)$ and $\sigma_1$ satisfies $Q$, we get $[\Gamma(x)]_{\sigma_1} \not\subseteq k$. The variable $x$ cannot be $x_3$, as $[\Gamma(x_3)]_{\sigma_1} = L$. By the definition of store equivalence, we have $s_1 \sim_{id}^k s_2$. Heap equivalence: trivial, since $h_1 \sim_{id}^k h_1$.

• $P = \pi.f := e$. Let $\ell_f = \text{ft}_\pi(f)$. Since $pc \sqsubseteq_Q \ell_f$, we know $[\ell_f]_{\sigma_1} \not\subseteq k$. We know $f$ cannot be $f_3$, since $\text{ft}(f_3) = \perp$. By the definition of heap equivalence and since no $f_3$ field is written, it follows $h_1 \sim_{id}^k h_2$. Store equivalence is trivial, since $s_1 \sim_{id}^k s_1$.

• $P = x := \text{new } C$. Store equivalence: For the variable $x$, it holds the same argument as for the ordinary assign rule. Heap equivalence: as the heap has not been changed for all $a \in \text{dom}(h_1) \cap \text{dom}(h_2)$, we get $h_1 \sim_{id}^k h_2$.

• $P = x := \pi.m(\tau)$. Store equivalence: For the variable $x$, it holds the same argument as for the ordinary assign rule. Other than that, no variables are changed in the caller’s store.
Heap equivalence: Since $pc_h \subseteq_Q t_h[\tau_{\#1}/x_4]$ and $\sigma_1$ satisfies $Q$, we know $[t_h]_{\sigma_1} \not\leq k$ in the method’s initial state $\hat{\sigma}_1$. Since the method $m$ is well-typed, we have $\Gamma, (\bot, t_h) \vdash Q \{P\} Q'$ for a new type environment $\Gamma = [\text{this} \mapsto k, \overline{y} \mapsto \overline{t}, \text{ret} \mapsto t_{\text{ret}}]$. Hence we get by induction $h_1 \sim_h h_2$.

\begin{proof}
The following is almost the main soundness theorem.

**Theorem 7.** If $\Gamma, pc \vdash Q \{P\} Q'$ and all methods are well-typed with respect to their signatures, then for all states $\sigma_1, \sigma_2, \sigma_1', \sigma_2'$ such that $\sigma_1$ and $\sigma_1'$ satisfy $Q$ and for all partial bijections $\beta$,

$$\sigma_1 \sim_{\beta} \sigma_1' \land \sigma_1 \xrightarrow{P} \sigma_2 \land \sigma_1' \xrightarrow{P} \sigma_2' \Rightarrow \sigma_2 \sim_{\gamma} \sigma_2'$$

for some partial bijection $\gamma \geq \beta$.

\begin{itemize}
  \item $P = \text{skip}$. From the assumption it follows $\sigma_1 \sim_h \sigma_1'$.
  \item $P = P_1 \mid P_2$. Then we have $\sigma_1 \xrightarrow{P_1} \sigma_2$ and $\sigma_2 \xrightarrow{P_2} \sigma_3$ as well as $\sigma_1' \xrightarrow{P_1} \sigma_2'$ and $\sigma_2' \xrightarrow{P_2} \sigma_3'$ from the operational semantics and $\Gamma, pc \vdash Q \{P_1\} Q'$ and $\Gamma, pc \vdash Q' \{P_2\} Q''$ from the type rules. By induction, we get $\sigma_2 \sim_h \sigma_2'$. From lemma 5, we know $\sigma_2$ and $\sigma_2'$ satisfy $Q'$. Again by induction, we get $\sigma_3 \sim_h \sigma_3'$.
  \item $P = \text{if } e \text{ then } P_1 \text{ else } P_2$. (This includes label tests.) We have $\Gamma \vdash e : \ell$.
    \begin{itemize}
      \item Let $[k]_{\sigma_1} \not\leq k$. With lemma 4, we get $[k]_{\sigma_1'} \not\leq k$. Therefore, $[pc \cup \ell]_{\sigma_1} \not\leq k$ and $[pc \cup \ell]_{\sigma_1'} \not\leq k$. We can thus infer by lemma 6 that $\sigma_1 \sim_{h_1} \sigma_2$ and $\sigma_1' \sim_{h_1} \sigma_2'$. By definition of the equivalence relations, we get $\sigma_2 \sim_h \sigma_2'$.
      \item Let therefore $[\ell]_{\sigma_1} \leq k$. Then by lemma 2, we have $[e]_{\sigma_1}, bequ [e]_{\sigma_1'}$, and even $[e]_{\sigma_1} = [e]_{\sigma_1'}$, because one cannot branch over a memory location. Therefore, the same branch $P_1$ is taken in both executions.
        $\sigma_1 \xrightarrow{P_1} \sigma_2$ and $\sigma_1' \xrightarrow{P_1} \sigma_2'$ and $\Gamma, pc \vdash Q \{P_1\} Q'$ imply by induction $s_2 \sim_\beta s_2'$.
    \end{itemize}
\end{itemize}

If the label test rule is applied, we can apply the induction hypothesis only if we show that $\sigma_1$ or $\sigma'$ satisfy the extended constraint set $Q, \ell_1 \subseteq \ell_2$ when the “then” branch is taken, but this follows from the operational semantics.
• $P = \text{while } e \text{ do } \ell$. We get $\Gamma \vdash e : \ell$.

This case is similar to the case for IF. With theorem 4, we know that if $\ell$ evaluates to a domain that is not below or equal to $k$, it also does so in the other state. Hence $pc \cup \ell$ evaluates to a domain that is not below or equal to $k$ in both states, and with theorem 6, each final state is id-equivalent to its corresponding initial state, and thus the final states are $\beta$-equivalent.

Let therefore $[\ell]_{\sigma_1} \leq k$. Then with lemma 2 $[e]_{\sigma_1} = [e]_{\sigma'_1}$.

- Case $[e]_{\sigma_1} = [e]_{\sigma'_1} = 0$: We have to show $\sigma_1 \sim^k_\beta \sigma'_1$, which follows from the assumption.

- Case $[e]_{\sigma_1} = [e]_{\sigma'_1} \neq 0$: We have $\sigma_1 \xrightarrow{P} \sigma_2$ and $\sigma'_1 \xrightarrow{P} \sigma'_2$ and $\Gamma, pc \cup \ell \vdash Q \{ P \} Q$, hence by induction $\sigma_2 \sim^k_\beta \sigma'_2$. Also, we have $\sigma_2 \xrightarrow{\text{while } e \text{ do } P} \sigma_3$ and $\sigma'_2 \xrightarrow{\text{while } e \text{ do } P} \sigma'_3$ and the original $\Gamma, pc \vdash Q \{ \text{while } e \text{ do } P \} Q$. Also, we know from lemma 5 that $\sigma_2$ and $\sigma'_2$ satisfy $Q$, and the $pc$ did not change. By induction, $\sigma_3 \sim^k_\beta \sigma'_3$.

• $P = x := e$. Since the heaps do not change, they remain $\beta$-equivalent.

We show heap equivalence as follows: Let $x^*$ be a variable such that $[[\Gamma(x^*)]]_{\sigma_1} \leq k$. We need to show $s_2(x^*) \sim_\beta s'_2(x^*)$.

It is easy to see that $[[\Gamma'(x^*)]]_{\sigma_1} \leq [[\Gamma'(x^*)]]_{\sigma'_2}$ if it is the case that $s_1(x_3) \leq s_2(x_3)$. The last condition holds because $x_3$ is either unchanged, or it is set to a higher domain value because of the assignment rule requirement $x_3 \in Q e$. Therefore $[[\Gamma'(x^*)]]_{\sigma_1} \leq k$, and thus $s_1(x^*) \sim_\beta s'_1(x^*)$ by store equivalence.

Let $x^* \neq x$. Then the variable is unchanged, hence $s_2(x^*) \sim_\beta s'_2(x^*)$.

If $x^* = x$, then with $\ell \subseteq Q \Gamma(x)$, we get $[[\ell]]_{\sigma_1} \leq k$, and with lemma 2, $[e]_{\sigma_1} \sim_\beta [e]_{\sigma'_1}$, which implies $s_2(x) \sim_\beta s'_2(x)$.

• $P = \pi.f := e$. As the stores are not changed, they remain $\beta$-equivalent.

We show heap equivalence as follows: Let $(a, a') \in \beta$ be a pair. Then $h_2(a) \sim^k_\beta h'_2(a')$.

- If $[[\pi]]_{\sigma_1} \neq a$ and $[[\pi]]_{\sigma'_1} \neq a'$, then the objects are not changed, hence $h_2(a) \sim^k_\beta h'_2(a')$.

- Let $[[\pi]]_{\sigma_1} = a$. Let $f^* \in \text{dom}(h_2(a))$ be a field. We show that if $[[\text{ft}(f^*)]]_{h_2(a)} \leq k$, then $h_2(a) \sim^k_\beta h'_2(a')$.

Very similar to the argument for variable assignment above, we get $[[\text{ft}(f^*)]]_{h_1(a)} \leq [[\text{ft}(f^*)]]_{h_2(a)}$, because if $f_3$ is changed, it is changed only “upwards”. It follows $[[\text{ft}(f^*)]]_{h_1(a)} \leq k$ and thus $h_1(a)(f^*) \sim_\beta h'_1(a')(f^*)$.

Let $f^* \neq f$. Since $f^*$ is not changed, we get $h_2(a)(f^*) \sim^k_\beta h'_2(a')(f^*)$.

Let $f^* = f$. With lemma 1, we have $[[\text{ft}(f)]]_{\sigma_1} = [[\text{ft}(f)]]_{h_1(a)} \leq k$.

With $\ell_1 \cup \ell_2 \subseteq Q \text{ft}(f)$, we get that $[[\ell_1]]_{\sigma_1} \leq k$ and $[[\ell_2]]_{\sigma_1} \leq k$, hence
Therefore, if to show equivalence for the stores, i.e. \( \hat{\Gamma}(a) = a' \), we need to show that the initial states of the method \( x = x_i \) are equivalent, and therefore \( h_2(a) = h_2'(a') \).

- \( P = x := \text{new} \ C \). Let \( a \) and \( a' \) be the fresh locations for the new object in \( h_1 \) and \( h_1' \). We define a partial bijection \( \gamma = \beta \cup \{(a, a')\} \) \( \supseteq \beta \). For \( \beta \)-equivalence, it holds a similar argument as for the ordinary variable assignment rule; for the assigned variable, we have \( s_2(x) = a \sim_\gamma a' = s_2'(x) \), thus \( s_2 \sim_\beta s_2' \).

As the newly created objects contain the same class information \((C)\) and the same default data in both runs, we can infer that the new objects are \( \gamma \)-equivalent, and therefore \( h_2 \sim_\beta h_2' \).

\( P = x := \pi.m(\overline{r}) \).

Let \( \hat{\Gamma} \) be the type environment for the method, and \( s_1 \) and \( s_1' \) be the initial stores of the called methods built from the arguments. To apply the lemma inductively, we need to show that the initial states of the method are \( \beta \)-equivalent. Since the heaps are passed without change, we only need to show equivalence for the stores, i.e. \( s_1 \sim_\beta s_1' \) with respect to \( \hat{\Gamma} \).

Let \( x_i \) be a formal method parameter from the sequence \( x_i \). We observe

\[
\left[ \hat{\Gamma}(x_i) \right]_{s_1, h_1} = \left[ \hat{\Gamma}(x_i) \right]_{s_1[x_i := r], h_1} = \left[ \hat{\Gamma}(x_i) \right]_{s_1[x_i := r' \#1]} = \left[ \ell'_i \right]_{s_1}.
\]

Therefore, if \( \left[ \hat{\Gamma}(x_i) \right]_{s_1} \leq k \), then \( \left[ \ell'_i \right]_{s_1} \leq k \), and since \( \ell_i \subseteq \ell'_i \), \( \left[ \ell_i \right]_{s_1} \leq k \).

It follows from lemma 2 that \( \left[ r' \#1 \right]_{s_1} \sim_\beta \left[ r' \#1 \right]_{s_1'} \) and thus \( s_1(x_i) \sim_\beta s_1'(x_i) \).

A similar argument holds for the \( \text{this} \) variable and the \( \ell_{\text{this}} \) and \( \ell'_{\text{this}} \) labels. Since the return value is initialized with the same value, \( s_1(\text{return}) \sim_\beta s_1'(\text{return}) \) holds trivially. Therefore, \( s_1 \sim_\beta s_1' \).

With the argumentation of lemma 5, the new states also satisfy the new constraint sets. Therefore, we can apply the lemma inductively, and get that the final states of the methods are \( \gamma \)-equivalent for some \( \gamma \supseteq \beta \).

As the heaps are directly passed back to the caller, we immediately get \( h_2 \sim_\beta h_2' \). We still need to show \( s_2 \sim_\beta s_2' \).

For the assignment of the variable \( x \), a similar argument holds as for ordinary assignment. We still need to show that if \( \left[ \Gamma(x) \right]_{s_2} \leq k \), then \( s_2(x) \sim_\gamma s_2'(x) \). With \( \ell_{\text{return}} \subseteq \Gamma(x) \), we get \( \left[ \ell_{\text{return}} \right]_{s_2} \leq k \). Since \( \ell_{\text{return}} \in \{\bot, \top\} \), we get \( \ell_{\text{return}} = \ell_{\text{return}} = \hat{\Gamma}(\text{return}) \). As by induction the final stores of the method are \( \gamma \)-equivalent, we have \( s_2(\text{return}) \sim_\gamma s_2'(\text{return}) \), therefore \( s_2(x) \sim_\gamma s_2'(x) \).
The main soundness theorem is only a corollary of the above:

**Corollary 8.** If $\Gamma, pc \vdash Q \{P\} Q'$, then $P$ is $(Q, Q')$ secure.

*Proof.* Follows from the definition of $(Q, Q')$-security, theorem 7, and the fact that the final states satisfy $Q$ (lemma 5). \qed