Tainted Secure Multi-Execution to Restrict Attacker Influence

McKenna McCall
Carnegie Mellon University
Pittsburgh, USA
mckennak@cmu.edu

Abhishek Bichhawat
Indian Institute of Technology Gandhinagar
Gandhinagar, India
abhishek.b@iitgn.ac.in

Limin Jia
Carnegie Mellon University
Pittsburgh, USA
liminjia@cmu.edu

ABSTRACT

Attackers can steal sensitive user information from web pages via third-party scripts. Prior work shows that secure multi-execution (SME) with declassification is useful for mitigating such attacks, but that attackers can leverage dynamic web features to declassify more than intended. The proposed solution of disallowing events from dynamic web elements to be declassified is too restrictive to be practical; websites that declassify events from dynamic elements cannot function correctly.

In this paper, we present SME^T, a new information flow monitor based on SME which uses taint tracking within each execution to remember what has been influenced by an attacker. The resulting monitor is more permissive than what was proposed by prior work and satisfies both knowledge- and influence-based definitions of security for confidentiality and integrity policies (respectively). We also show that robust declassification follows from our influence-based security condition, for free. Finally, we examine the performance impact of monitoring attacker influence with SME by implementing SME^T on top of Featherweight Firefox.

1 INTRODUCTION

Online services for banking, social media, shopping, etc., typically require access to the user's personal information such as their phone number, location, or credit card details. Web attackers have been known to steal sensitive user data [25], sometimes via third-party scripts, which have been observed indiscriminately collecting data from web forms, including personal information [38].

Information flow control (IFC) monitors are a promising way to prevent sensitive information from leaking to attackers [21, 30, 34]. They have been used to secure applications in many domains [20, 22, 26, 27, 39, 43]. The canonical IFC security property is *noninterference*. The simplest form of noninterference says that public outputs (least privileged) should never be influenced by secret inputs (requiring the most privilege). However, in many real-world applications, this definition is too restrictive to be practical. Supporting principled *declassification*, which allows selected sensitive information to be leaked while maintaining an otherwise provably secure system, is important for many useful web services like website analytics. For instance, if a company wants to know which products are being clicked on, they may want to track some of their customers' interactions on their site. Declassification can ensure that these third-party analytics will have access to the information they need (e.g., which products are clicked on), without releasing *other* sensitive information.

Prior work that allowed declassification by web scripts either did not prove formal properties about declassification [11, 12], or used a simplified model missing some dynamic JavaScript features that could be leveraged by an attacker to leak information [40]. Later work explored the threat posed by declassifying events associated with dynamically added page elements and developed a technique using secure multi-execution (SME) to prevent these leaks [29] (detailed discussion in Section 2.2). However, the proposed solution disallows all events from dynamically generated web elements from being declassified. While this technique is provably secure, it risks altering the behavior of secure programs and could prevent declassification in the benign example described above.

This paper aims to develop an IFC monitor that allows flexible declassification without sacrificing security. Since SME enjoys strong security guarantees and do not need to abort the program (as opposed to NSU [7]), which is desirable for web applications, we build on prior work on securing dynamic secrets with SME [29] to develop a more fine-grained technique for protecting dynamic features from leaking secrets due to declassification.

One key insight is that leaks caused by attackers' interactions with declassification can be stopped if the monitor tracks attacker influence on the page, only preventing declassification when it involves code added by the attacker. We provide more detailed examples in Section 3.

We design SME^T by extending prior work [29] with techniques based on integrity labels to enforce robust declassification [19, 42]. Specifically, SME^T uses taint tracking within *each* execution to remember the trustworthiness of page elements and their event handlers via integrity labels. These integrity labels indicate attacker influence and decide whether declassification is allowed.

We define our security conditions based on knowledge-based noninterference [1, 4, 6, 9, 10]. We present a novel knowledge-based security condition where robust declassification follows from influence-based security for free.

The same techniques may be applied to knowledge-based security conditions to prove transparent endorsement [18] (the integrity dual of robust declassification), but in this paper, we focus on robust declassification for ease of understanding. We prove that the design of SME^T is secure. We implement our model on top of Featherweight Firefox as a sanity check on our semantics and to understand the impact of our more complex security lattice on performance.

This paper makes the following technical contributions.

- A novel IFC monitor design that combines SME and taint tracking for more permissive declassification
- Novel security conditions that capture confidentiality, integrity, and robust declassification.
- Proofs that SME^T is secure.
- A prototype implementation in Featherweight Firefox.

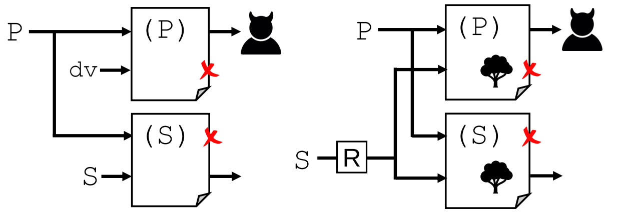
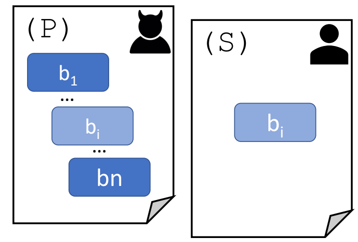


Figure 1: Standard SME (left) and SME with declassification and multiple DOMs (right)

$$
\begin{array}{lll}
\multicolumn{3}{l}{\mathsf{onKeypress}(\mathit{secret}) = \mathsf{case}\ \mathit{secret}:} \\
\quad \mid 1 & \Longrightarrow & \mathsf{new}(b_1); \mathsf{addEH}(b_1, \mathsf{on}\mathit{Click}\{\mathsf{output}_P(1)\}); \\
\quad \dots & & \\
\quad \mid \mathsf{n} & \Longrightarrow & \mathsf{new}(b_n); \mathsf{addEH}(b_n, \mathsf{on}\mathit{Click}\{\mathsf{output}_P(n)\}); \\
\quad \mid \mathsf{dv} & \Longrightarrow & \mathsf{new}(b_1); \mathsf{addEH}(b_1, \mathsf{on}\mathit{Click}\{\mathsf{output}_P(1)\}); \\
 & & \dots \\
 & & \mathsf{new}(b_n); \mathsf{addEH}(b_n, \mathsf{on}\mathit{Click}\{\mathsf{output}_P(n)\});
\end{array}
$$

(a) Event handler added by the attacker.



(b) Resulting attacker view (P) and user view (S) of the page.

Figure 2: Example of dynamic features causing leaks. The dv **case guarantees that the attacker copy will have a matching button (colored light blue) to capture the declassified event and leak the secret.**

2 BACKGROUND AND RELATED WORK

2.1 Reactive systems and IFC monitors

Reactive systems have been widely used to model web applications. A reactive program is a set of event handlers which execute when they are triggered by events [16]. We consider a single-threaded model where event handlers execute one at a time. While an event handler is running, the system is in Producer state. After the event handlers finish execution, the system waits in Consumer state for more events to process.

Secure multi-execution (SME) enforces IFC policies in reactive programs by executing event handlers multiple times—once at each security level. Each execution only receives the inputs it has privilege to see, and only outputs to channels matching the security level of the execution [23, 24]. Consider a two-point security lattice with labels P (Public) and S (Secret), and the ordering $P \sqsubseteq S$ (meaning information can flow from P to S but not vice versa). As shown in Figure 1, SME would run event handlers twice, where both copies of the execution see P-labeled data. The S copy of the execution can see all of the data, but can only output to privileged (Secret) channels. In the P copy of the execution, Secrets are replaced with a default value (dv) and can only output to Public channels.

Faceted execution [8, 14] is a similar multi-execution technique. Rather than running all of the code multiple times, this approach creates "facets" of values for every level in the lattice, only when they depend on a secret. The code runs once until the control flow depends on a faceted value and the execution splits to evaluate each facet. Later work combines SME and faceted execution [36] (and an optimization [2]) and proposes "generalized" multiple facets [32] to balance the security and performance tradeoffs of the two multi-execution techniques (in the first two cases), consider a more general security lattice (in the last case), and each achieves stronger (termination-sensitive) security guarantees than offered by traditional faceted execution. SME^T also uses a general security lattice. The techniques SME^T uses may be relevant to faceted execution, but we focus on SME because the semantics are simpler.

Taint tracking approaches enforce IFC policies by attaching labels to the data in the system, which indicates their secrecy and trustworthiness. The label on the data determines if an output is permitted (if the channel trusts the data and has enough privilege to receive it) or not. Taint tracking is susceptible to implicit leaks when branching on a secret. One solution is to abort the execution when updating public data in secret contexts (called *no sensitive upgrade* [7]), or simply permit the leaks and block only explicit leaks that output secret information to public channels (satisfying a weaker security condition called *explicit secrecy* [28, 37]). SME^T is a new monitor which composes SME with taint tracking so that we can keep track of the trustworthiness of the event handlers within each execution. These labels are determined when the event handler is initially registered and remain fixed throughout execution, so we don't need to worry about sensitive upgrades, nor do we have to resort to an explicit secrecy security condition.

2.2 Declassification with dynamic features

The monitors described above enforce strict *noninterference*, where secret inputs are never allowed to influence public outputs. But this is often too restrictive for common use cases such as analytics where an online shop wants to learn which products users are clicking on most, or user authentication where a bank wants to know a user's location. Declassification offers a principled way to release some information. Vanhoef et al. developed an approach to *stateful* declassification in SME [40], where declassification policies are flexible enough to release events, as well as aggregated/approximated data. For instance, "the user's approximate location may be released after they give permission" and "the average location of every 100 mouse clicks may be released" are both stateful policies.

However, McCall et al. [29] showed that dynamic features can be used by an attacker to leak more than is allowed by these declassification policies via the following attack. Consider the 2-point security lattice from before and a web page with the policy: all user events are secret, click events and the occurrence of keypress events may be declassified (however, *which* key was pressed should remain secret). The P copy of the page is visible to the attacker, and receives only the public (or declassified) events, while the S copy is visible to the user and receives all of the events. Suppose the attacker registers the event handler shown in Figure 2a which runs whenever a key is pressed and adds a different button to the page, depending on what is typed (stored in *secret*). If the user types i, this event handler would add button b_i to the S copy of the page based on the actual value of the secret. The P copy of the page receives the event with a default value dv to hide what was typed, so the event handler adds *all* possible buttons to the page. When the user clicks on b_i (S copy of the page), the click is declassified to the P copy, which is guaranteed to have a matching button to capture the event. The on*Click* event handler executes the statement $\text{output}_P(i)$. Since outputs to P channels are allowed in the P execution, this leaks what the user typed to the attacker.

To prevent this leak, McCall et al. [29] propose an additional label S_Δ for dynamically-generated elements, and *restrict* declassification to only apply to elements labeled S, i.e., events dispatched on elements labeled S_Δ are never declassified to P. Accordingly, in the previous example, the button b_i is labeled S_Δ; hence, the mouse click on b_i is not declassified to P, which prevents the attacker from learning which key was pressed. While this prevents unintentional leaks, it can be too restrictive to be practical, which is one of the motivations for this work.

3 MOTIVATING EXAMPLES

Recall the scenario from Section 1 where an online shop wants to know which of their products are receiving the most attention. They use JavaScript to dynamically display products on their site depending on what the user has searched. To measure product popularity, they use a third-party analytics library to track where users are clicking on their site. Because they do not want the third-party to have access to *all* of the user's private information, they treat the script as Public. To give the library access to the relevant click information, the shop employs a policy where the coordinates of each click are Secret, but which product is clicked may be declassified. With the solution described above, everything dynamically loaded to the page (even by code not controlled by the attacker) will be labeled S_Δ and excluded from declassification, and thus, the online shop won't be able to perform their analytics.

The reason the earlier example (Figure 2b) leaked more than intended is that the *attacker* leveraged the declassification policy to leak information by adding buttons to the page. Meanwhile, the products added to the web page described above are added by the shop itself, who should be trusted to trigger declassification. The underlying problem is not the dynamic page elements, but their *source*. Instead of disallowing any dynamic features to influence declassification, an intuitive fix would simply restrict the attacker's influence. This involves protecting the *integrity* of the data, which is dual to the *confidentiality* policies we have discussed so far.

3.1 Tracking integrity in SME

Consider a 4-point security lattice with 2 confidentiality labels (Public and Secret) and 2 integrity labels (Trusted and Untrutsed). Information is allowed to flow from Public to Secret and Trusted to Untrusted. The complete security lattice is a diamond with (P, T) at the bottom, (S, U) at the top, and the other labels (P, U) and (S, T) in between. SME can enforce information flow policies drawn from this lattice by running one execution for each of these 4 security levels as shown in Figure 3. In this model, the attacker and other Untrusted parties, like *ad.com*, are only able to influence the code

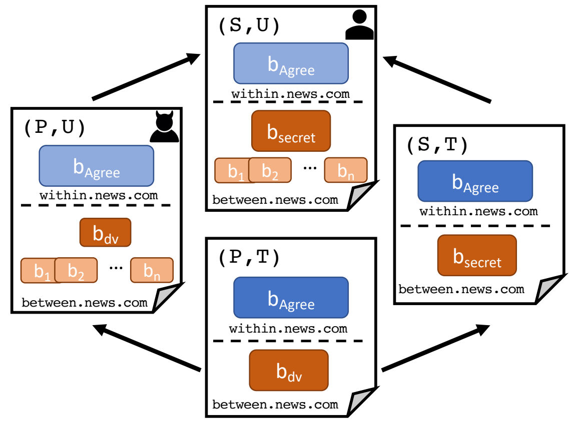


Figure 3: Information is allowed to flow in the direction of the arrows. The attacker can influence Untrusted executions to add page elements or event handlers to try to manipulate declassification directly within an execution (blue case) or indirectly between executions (orange case).

running on the Untrusted executions, while Trusted parties (like *news.com*) may influence code running in any execution. In our examples, the (P, U) execution communicates with the attacker via *ad.com* and the user is shown the (S, U) version of the webpage.

In the following examples, we show that attackers can influence declassification irrespective of whether the user interacts with attacker code directly (similar to the leak from [29]) or indirectly (if a declassification triggers attacker code in *another* execution).

Example 3.1. **Leaks within an execution.** Suppose a user visits a webpage (*within.news.com*) which explains that it will share their account preferences with advertisers (*ad.com*), but only if they click the "Agree" button (identified in the code as b_{Agree}) to consent. When the page loads, *ad.com* adds a large b_{Agree} button at the top of the page with the text "Click me!", as in Figure 3, where the buttons coming from *ad.com* are light blue and the ones from *within.news.com* are dark blue. A user may click the button, not realizing it will declassify their preferences. We call this a leak *within* an execution because the user is interacting directly with attacker-controlled code. This is similar to the attacks from prior work [29], where the user interacts directly with the page element.

Example 3.2. **Leaks between executions.** Consider another webpage (*between.news.com*) which has the policy that keypress events are Secret, but clicks may be declassified from Secret to Public. *news.com* installs an event handler which adds a different button to the page, depending on which key the user presses (similar to onKeypress in Figure 2, without the dv case). Meanwhile, *ad.com* adds all possible buttons to the page and registers an event handler which is triggered by a click to send them a message, telling them which button was clicked (similar to the dv case from the onKeypress event handler in Figure 2). The resulting page is shown in Figure 3, where the dark orange buttons were added by *news.com* and the light orange buttons were added by *ad.com*. Note that because *news.com* is Trusted, the dark orange buttons are added to all copies of the webpage, including the Untrusted ones.

Like the leak from prior work [29], if the user clicks the b_{secret} button on the (S, U) page, the event will be declassified to the (P, U) execution, which is guaranteed to have a matching button to capture the event and leak the keypress to the attacker. We call this a leak *between* executions because the user is interacting with code added by the host page which triggers attacker-controlled code in *another* execution. This example highlights that it is not enough to only look at the page the user is interacting with, we also need to consider the executions capturing the declassified events.

To prevent the attacker from influencing declassification, one approach would be to extend the solution from prior work [29] to apply to events *originating from* dynamic elements in Untrusted executions (which might include attacker-controlled code), as well as events being *released to* dynamic elements in the Untrusted executions. But as we described above, this would also prevent innocent declassifications, like the online shop in the previous example. Likewise, it wouldn't be enough to prevent the user from interacting directly with attacker-controlled code by showing them the (S, T) copy of the page instead of the (S, U) copy, because this would still be susceptible to the leaks between executions.

Our approach: To prevent these leaks without sacrificing functionality, we develop SME^T (Section 5.2), which is SME with taint tracking to reflect the trustworthiness of the source of the code adding new page elements and event handlers. We check that the user trusts the code they're interacting with directly to decide if a declassification should be triggered (preventing leaks *within* executions), as well as the code in other executions to decide whether they should receive the event (preventing leaks *between* executions).

4 SME WITH DYNAMIC FEATURES

We first describe the syntax and semantics of SME for reactive systems with dynamic features (declassification will be added in the next section). Our semantics are flexible enough to work with any finite security lattice of confidentiality and integrity labels. Following prior work [29], we organize our SME semantics into three levels: the top-most level is responsible for processing inputs and outputs, looking up event handlers, and switching between executions. The mid-level manages the execution for a particular execution. The lowest level runs the current event handler.

4.1 I/O Processing and EH Lookup

The syntax for these rules is summarized in Figure 4. The security lattice includes confidentiality labels, $l_c \in \mathcal{L}_c$, which specifies the privilege needed to access data, and integrity labels, $l_i \in \mathcal{L}_i$, which specifies how trusted a component is. Information may flow from (l_c, l_i) to (l'_c, l'_i) if l'_c has privilege to see data from l_c ($l_c \sqsubseteq l'_c$) and l'_i trusts data from l_i ($l_i \sqsubseteq l'_i$). Our earlier example used a security lattice with $\mathcal{L}_c = \{P, S\}$ and $\mathcal{L}_i = \{T, U\}$ for $P \sqsubseteq S$ and $T \sqsubseteq U$, but our rules are general enough to accommodate any (finite) lattice.

Events are associated with elements given by unique identifiers id. Event handlers of the form on$Ev(x)\{c\}$ run command c with argument x when the system receives event Ev (such as a click). The security label (l_c, l_i) of an event is determined by the security policy $\mathcal{P}$. An execution trace T is zero or more steps of the top-level system. An SME configuration K is a snapshot of the system including the SME state Σ and the configuration stack ks.

Σ keeps track of the persistent state for each execution; each security level $pc = (l_c, l_i)$ has its own store $\sigma^{EH}_{(l_c, l_i)}$ which is the event handler storage (i.e., the DOM) for each execution. The event handler storage maps identifiers id to attributes v and event handler maps M, which maps events Ev to their respective event handlers $eh_1, \ldots, eh_n$. This model allows each execution to have its own copy of the DOM, whose contents may vary in privilege and trust. Each execution runs its event handlers separately, beginning at the top of the configuration stack ks. Each element of the configuration stack determines what event handler to run, given by configuration κ, and in which execution, given by the security level pc.

As the system runs, it may react to/emit various actions, α. In the reactive setting, the system waits until it receives an input which is an event triggering (zero or more) event handlers which may produce some outputs. In our case, inputs are user interactions $id.Ev(v)$ which are events Ev associated with an element id (possibly) carrying some argument (e.g., which key is pressed for a keyPress event or the location of a click). Outputs are given by values sent along a channel ch. The other actions are silent $\bullet$.

The semantics for the top-most level are shown in Figure 5. Rule In receives an event Ev for page element id with parameter v from the principal with privilege and trustworthiness given by pc. The security policy tells us the label on the event is pc'. We run the event handlers associated with the event in each execution with enough privilege to see the event and who trust the event, i.e., at all executions at or above $pc \sqcup pc'$ in the security lattice. The lookup semantics ($\Sigma, E \rightsquigarrow$ ks) looks up the event handlers in Σ and constructs a configuration for each execution in E, resulting in ks.

The output rules run event handlers one at a time. When an event handler is running, the configuration at the top of the stack is in producer state, producer(κ). Rule Out handles outputs produced by the event handler. An execution performs outputs to channels only if the label on the channel matches the execution context, i.e., $\mathcal{P}(ch(v)) = pc$. Otherwise, the output is suppressed. Rule Out-Silent handles steps which don't produce outputs. When the event handler finishes running, the configuration at the top of the stack is in consumer state, consumer(κ), and rule Out-Next pops the configuration off the stack to run the next event handler. The execution state is managed by the mid-level semantics, described next.

Example: Example 3.1 of leaks *within* an execution uses the security policy that click events are considered secret and trusted, $\mathcal{P}(\_.\mathsf{Click}(\_)) = (S, T)$[1] and page load events are public and trusted, $\mathcal{P}(\_.\mathsf{load}(\_)) = (P, T)$. The user interacts with the (S, U) copy of the page and the attacker who serves ads from *ad.com* is listening on (P, U) channels.

Initially, before any events have been triggered, we assume that the SME state is well-formed, meaning the source of the code (l_i) loaded to each execution (l'_i) is trusted ($l_i \sqsubseteq l'_i$). The attacker-controlled code from *ad.com* only appears in Untrusted executions, while the code from Trusted *news.com* will appear in all of the executions. For our example, we also assume that *ad.com* registers

[1] Not to be confused with the isTrusted property distinguishing events which come from a user from events which were generated by an event handler (see [41]).

$$
\begin{array}{llcl}
\textit{Security lattice:} & \mathcal{L} & ::= & \mathcal{L}_c \times \mathcal{L}_i \\
\textit{Event:} & Ev & ::= & \mathsf{click} \mid \mathsf{keyPress} \mid \ldots \\
\textit{Event handler:} & eh & ::= & \mathsf{on}Ev(x)\{c\} \\
\textit{Security policy:} & \mathcal{P} & & \\
\text{Individual event handler} & & & \\
\textit{Expression:} & e & ::= & x \mid v \mid id \mid \mathsf{uop}\, e \mid e_1 \,\mathsf{bop}\, e_2 \\
\textit{Command:} & c & ::= & \mathsf{skip} \mid c_1; c_2 \mid x := e \mid id := e \\
 & & \mid & \mathsf{while}\; e \;\mathsf{do}\; c \mid \mathsf{if}\; e \;\mathsf{then}\; c_1 \;\mathsf{else}\; c_2 \\
 & & \mid & \mathsf{output}\; ch\; e \mid \mathsf{new}(id, e) \\
 & & \mid & \mathsf{addEh}(id, eh) \mid \mathsf{trigger}\; id.Ev(e)
\end{array}
$$

$$
\begin{array}{llcl}
\textit{Single configuration:} & \kappa & ::= & \sigma^v, c, s, E \\
\textit{Execution state:} & s & ::= & P \mid C \\
\textit{SME traces:} & T & ::= & K \mid \mathcal{P} \vdash T \overset{\alpha_l}{\Longrightarrow} K \\
\textit{Event queue:} & E & ::= & \cdot \mid E, (id.Ev(v), pc) \\
\textit{SME configuration:} & K & ::= & \Sigma; \mathsf{ks} \\
\textit{SME state:} & \Sigma & ::= & \cdot \mid \Sigma, pc \mapsto \sigma^{EH}_{pc} \\
\textit{EH state:} & \sigma^{EH} & ::= & \cdot \mid \sigma^{EH}, id \mapsto (v, M) \\
\textit{EH map:} & M & ::= & \cdot \mid M, Ev \mapsto \{eh_1, \ldots, eh_n\} \\
\textit{Configuration stack:} & \mathsf{ks} & ::= & \cdot \mid (\kappa, pc) :: \mathsf{ks} \\
\textit{Actions:} & \alpha & ::= & id.Ev(v) \mid ch(v) \mid \bullet
\end{array}
$$

Figure 4: SME Syntax

$$\boxed{\mathcal{P} \vdash K \overset{(\alpha, pc)}{\Longrightarrow} K'}$$

$$\frac{\mathcal{P}(id.Ev(v)) = pc' \qquad E = ((id.Ev(v), pc'') \mid pc'' \in \mathcal{L} \text{ s.t. } pc \sqcup pc' \sqsubseteq pc'') \qquad \Sigma, E \rightsquigarrow \mathsf{ks}}{\mathcal{P} \vdash \Sigma; \cdot \overset{(id.Ev(v), pc)}{\Longrightarrow} \Sigma; \mathsf{ks}}\ \textsc{In}$$

$$\frac{\mathsf{producer}(\kappa) \qquad \Sigma, \kappa \xrightarrow{ch(v)}_{pc} \Sigma', \mathsf{ks}' \qquad \alpha = ch(v) \text{ if } \mathcal{P}(ch) = pc \qquad \alpha = \bullet \text{ otherwise}}{\mathcal{P} \vdash \Sigma; (\kappa, pc) :: \mathsf{ks} \overset{(\alpha, pc)}{\Longrightarrow} \Sigma'; \mathsf{ks}' :: \mathsf{ks}}\ \textsc{Out}$$

$$\frac{\mathsf{producer}(\kappa) \qquad \Sigma, \kappa \xrightarrow{\alpha}_{pc} \Sigma', \mathsf{ks}' \qquad \alpha \neq ch(v)}{\mathcal{P} \vdash \Sigma; (\kappa, pc) :: \mathsf{ks} \overset{(\alpha, pc)}{\Longrightarrow} \Sigma'; \mathsf{ks}' :: \mathsf{ks}}\ \textsc{Out-Silent}$$

$$\frac{\mathsf{consumer}(\kappa)}{\mathcal{P} \vdash \Sigma; (\kappa, pc) :: \mathsf{ks} \overset{(\bullet, pc)}{\Longrightarrow} \Sigma; \mathsf{ks}}\ \textsc{Out-Next}$$

$$\boxed{\Sigma, E \rightsquigarrow \mathsf{ks}}$$

$$\frac{\Sigma(pc)(id.Ev(v)) = c \qquad \kappa = \cdot, c, P, \cdot \qquad \Sigma, E \rightsquigarrow \mathsf{ks}}{\Sigma, (id.Ev(v), pc) :: E \rightsquigarrow (\kappa, pc) :: \mathsf{ks}}\ \textsc{lookup}$$

$$\frac{}{\Sigma, \cdot \rightsquigarrow \cdot}\ \textsc{lookup-empty}$$

Figure 5: Top-level SME rules for processing inputs and outputs, and looking up event handlers

$$\boxed{\Sigma, \kappa \xrightarrow{\alpha}_{pc} \Sigma', \mathsf{ks}}$$

$$\frac{}{\Sigma, \sigma, \mathsf{skip}, P, \cdot \xrightarrow{\bullet}_{pc} \Sigma, ((\sigma, \mathsf{skip}, C, \cdot), pc)}\ \textsc{PtoC}$$

$$\frac{E = (id.Ev(v), pc) :: E' \qquad \Sigma, E \rightsquigarrow \mathsf{ks}}{\Sigma, \sigma, \mathsf{skip}, P, E \xrightarrow{\bullet}_{pc} \Sigma, ((\sigma, \mathsf{skip}, C, \cdot), pc) :: \mathsf{ks}}\ \textsc{PtoLC}$$

$$\frac{\Sigma, \sigma, c \xrightarrow{\alpha}_{pc} \Sigma', \sigma', c', E'}{\Sigma, \sigma, c, P, E \xrightarrow{\alpha}_{pc} \Sigma', ((\sigma', c', P, (E, E')), pc)}\ \textsc{P}$$

Figure 6: Mid-level rules for processing the event queue

an onLoad^U function to add the "Click me!" button (from Figure 3), and *news.com* registers onLoad^T to add the "Agree" button.

Then, the initial SME configuration is $K_0 = \Sigma_0; \mathsf{ks}_0$ where ks_0 runs *body*.load for each execution (ks_0 will be described in more detail in the next section) in the following SME state:

$$
\begin{array}{lll}
\Sigma_0 = & (S, U) \mapsto & body \mapsto (\_, \mathsf{load} \mapsto \{\mathsf{onLoad}^U, \mathsf{onLoad}^T\}), \\
 & (P, U) \mapsto & body \mapsto (\_, \mathsf{load} \mapsto \{\mathsf{onLoad}^U, \mathsf{onLoad}^T\}), \\
 & (S, T) \mapsto & body \mapsto (\_, \mathsf{load} \mapsto \{\mathsf{onLoad}^T\}), \\
 & (P, T) \mapsto & body \mapsto (\_, \mathsf{load} \mapsto \{\mathsf{onLoad}^T\}),
\end{array}
$$

Next, the (S, U) execution runs the onLoad^U event handler. Rule Out-Silent applies and makes a step: $K_0 \overset{(\bullet, (S,U))}{\Longrightarrow} K_1$. The new configuration K_1 has a new button in the (S, U) copy of the store and the other copies remain unchanged:

$$
\begin{array}{ll}
\Sigma_1 = & (S, U) \mapsto \quad body \mapsto (\ldots), b_{\text{Agree}} \mapsto (\text{“Click me!”}, \cdot) \\
 & \text{the rest are the same as } \Sigma_0
\end{array}
$$

The same process will repeat to add the "Click me!" button to the (P, U) store and the "Agree" button to the other executions. Now that the event handlers have finished running, rule Out-Next pops the event handler from ks and the system waits for user input.

Suppose the attacker also installed an event handler in the (S, U) and (P, U) executions which directly sends them the user's account preferences. Since they are listening on a (P, U) channel, the rule out would suppress the output from the (S, U) execution which knows the real preferences (since $\mathcal{P}(ch) \neq (S, U)$). The same rule allows the output from the (P, U) execution, which would instead output a default value dv, with no access to the real preferences.

4.2 Execution State and EH Queue

A single configuration κ is a snapshot of one execution, including the local variables σ^v (which are only accessible to the event handler currently running), the current command c being executed, the execution state s of the event handler, and the event queue E. The execution state is either P for producer (meaning an event handler is running) or C for consumer (meaning the event handlers have finished and the execution is ready to process a new event). Here, the event queue, E, is a list of the events triggered by other event handlers. The events will run in the same execution, so the pc on each event in the queue will match the current execution context.

$$\boxed{\Sigma, \sigma, c \xrightarrow{\alpha}_{pc} \Sigma', \sigma', c', E}$$

$$\frac{[\![e]\!]^{pc}_{\sigma,\Sigma} = v}{\Sigma, \sigma, \mathsf{output}\ ch\ e \xrightarrow{ch(v)}_{pc} \Sigma, \sigma, \mathsf{skip}, \cdot}\ \textsc{output}$$

$$\frac{[\![e]\!]^{pc}_{\sigma,\Sigma} = v \qquad E = (id.Ev(v), pc)}{\Sigma, \sigma, \mathsf{trigger}\ id.Ev(e) \xrightarrow{\bullet}_{pc} \Sigma, \sigma, \mathsf{skip}, E}\ \textsc{trigger}$$

$$\frac{\begin{array}{c}[\![e]\!]^{pc}_{\sigma,\Sigma} = v \qquad \Sigma(pc) = \sigma^{EH} \\ id \notin \sigma^{EH} \qquad \Sigma' = \Sigma[pc \mapsto \sigma^{EH}[id \mapsto (v, \cdot)]]\end{array}}{\Sigma, \sigma, \mathsf{new}(id, e) \xrightarrow{\mathsf{new}(id)}_{pc} \Sigma', \sigma, \mathsf{skip}, \cdot}\ \textsc{new}$$

$$\frac{\begin{array}{c}\Sigma(pc) = \sigma^{EH} \qquad \sigma^{EH}(id) = (v, M) \\ \sigma^{EH'} = \sigma^{EH}[id \mapsto (v, M[Ev \mapsto M(Ev) \cup eh])] \\ \Sigma' = \Sigma[pc \mapsto \sigma^{EH'}]\end{array}}{\Sigma, \sigma, \mathsf{addEh}(id, Ev, eh) \xrightarrow{\mathsf{new}(id, eh)}_{pc} \Sigma', \sigma, \mathsf{skip}, \cdot}\ \textsc{add-eh}$$

Figure 7: (Selected) rules for running event handers

The semantics for managing the event handler queue and execution state are shown in Figure 6. Rule PtoC handles the case where an event handler has finished running (c = skip) and no other event handlers have been triggered ($E = \cdot$). In this case, the execution state is changed to C for consumer state. On the other hand, if an event handler has triggered other event handlers to run ($E \neq \cdot$), rule PtoLC will additionally look up the event handlers in E and return these event handlers in ks. Finally, rule P runs an event handler using the event handler semantics, described below.

Example: The top-level I/O rules use the execution state to decide whether they should continue running the event handler (rules Out and Out-Silent) or pop the event handler off ks to run the next event handler, if one exists (rule Out-Next), or wait for another input (rule In) if one doesn't. From the leaks *between* executions example in Section 3.1, before the user presses a key on their keyboard or clicks the button, the system is in Consumer state, waiting for user interaction (ks = $\cdot$). When the user clicks the *secret* button, the input rule In looks up the event handler for *secret*.Click() and the rule lookup sets the execution state to Producer. The rule P in the mid-level semantics run the event handler to completion and then PtoC switches the execution state back to Consumer state.

4.3 Individual Event Handlers

Expressions in the body of an event handler include variables, values (integers and booleans), page element identifiers, *id*, unary, and binary operators. Commands are mostly standard and include outputs to channels and dynamic behaviors for adding new page elements (new(id, e)), registering new event handlers (addEh(id, eh)), and triggering event handlers (trigger $id.Ev(e)$).

Selected event handler operational semantic rules are in Figure 7. Expression evaluation is denoted $[\![e]\!]^{pc}_{\sigma,\Sigma}$ where pc tells us which copy of the shared storage to access in Σ and σ is the store local to the current event handler. Candidate outputs are produced by rule output. The other rules are for handling dynamic elements, including triggering event handlers (rule trigger), generating new page elements (rule new), and registering a new event handler (rule add-eh). In each of these rules, we interact with the copy of the global storage that matches the current execution context. Event handlers run in the same context they were triggered in, denoted by pc. New page elements must have a unique identifier, $id \notin \sigma^{EH}$, and are initialized with the given attribute and no event handlers, $M = \cdot$. When registering a new event handler, the existing event handlers associated with the event are looked up in the event handler map, $M(Ev)$. The event handler map is updated to include the original event handlers plus the new one, $M[Ev \mapsto M(Ev) \cup eh]$.

5 DECLASSIFICATION AND SME^T

We extend the syntax and semantics from Section 4 to include declassification. Due to space constraints, we describe the changes to the rules in this section and present the full rules in Appendix A.

$$\boxed{\mathcal{P}, \mathcal{D} \vdash K \overset{(\alpha, pc)}{\Longrightarrow} K'}$$

$$\frac{\begin{array}{c}\mathcal{P}(id.Ev(v)) = pc' \\ E = ((id.Ev(v), pc'') \mid pc'' \in \mathcal{L} \text{ s.t. } pc \sqcup pc' \sqsubseteq pc'') \\ (\mathcal{R}', E_d) = \mathsf{declassify}(\mathcal{D}, \mathcal{R}, \Sigma, (id.Ev(v), pc), pc') \\ \Sigma, E :: E_d \rightsquigarrow \mathsf{ks}\end{array}}{\mathcal{P}, \mathcal{D} \vdash \mathcal{R}; \Sigma; \cdot \overset{(id.Ev(v), pc)}{\Longrightarrow} \mathcal{R}'; \Sigma; \mathsf{ks}}\ \textsc{In}$$

$$\boxed{\mathsf{declassify}(\mathcal{D}, \mathcal{R}, \Sigma, (id.Ev(v), pc)) = (\mathcal{R}', E)}$$

$$\frac{\begin{array}{c}\mathcal{R} = (\rho, d) \qquad \mathcal{D}((id.Ev(v), pc), pc', \rho) = (\rho', v_d, E_d) \\ d' = \mathsf{update}(d, v_d)\end{array}}{\mathsf{declassify}(\mathcal{D}, \mathcal{R}, \Sigma, (id.Ev(v), pc), pc') = ((\rho', d'), E_d)}\ \textsc{declassify}$$

Figure 8: Updated input rule for declassification. Key changes are shown in red text.

5.1 Stateful Declassification

We use *stateful* declassification [29, 40]. A stateful policy is one that may involve the system state when deciding whether to declassify. Here, we describe the syntax for declassification, shown below.

$$\begin{array}{llll} \textit{Declass. policy:} & \mathcal{D} & & \\ \textit{Declass. module:} & \mathcal{R} & ::= & (\rho, d) \\ \textit{Declass. state:} & \rho & ::= & \cdot \mid \rho, (id_1.Ev_1, n_1) \\ \textit{Declass. channel:} & d & ::= & (\iota_1, v_1), \cdots, (\iota_n, v_n) \end{array}$$

The declassification policy is given by $\mathcal{D}$. Given an event and the current state, as well as information from the security policy, $\mathcal{P}$, $\mathcal{D}$ updates the current state and decides whether the event should be declassified. The declassification module $\mathcal{R}$ keeps track of the current state for making decisions about declassification as well as channels for event handlers to access released values. A declassification state ρ keeps track of relevant state conditions, such as the number of times an event has been seen, and the declassification channel d associates locations ι (such as a line number in the code) with the released value accessible by that location.

Rules for the I/O semantics is updated to include $\mathcal{D}$ and $\mathcal{R}$:

$$\mathcal{P}, \mathcal{D} \vdash \mathcal{R}; \Sigma; \mathsf{ks} \overset{\alpha_I}{\Longrightarrow} \mathcal{R}'; \Sigma'; \mathsf{ks}'$$

A declassification function (declassify), shown in Figure 8 is added to the input rule. It uses the declassification policy $\mathcal{D}$ to determine whether the new event should be released to run event handlers in additional execution contexts E_d, whether the system state ρ should be updated, and what values should be updated on the declassification channel d (if any).

Example: Recall Example 3.1 of leaks *within* an execution, where the security policy says that clicks are (S, T), and the declassification policy says that the user's preferences may be declassified from S to P when b_{Agree} is clicked.

When the user clicks b_{Agree} in the (S, U) execution, In will share the event with all the executions with enough privilege trust the user (just (S, U)), but we also use declassify to determine whether the event should be declassified to additional executions:

$$\mathcal{D}((b_{\mathsf{Agree}}.\mathsf{Click}(\ ), (S, U)), (S, T), (b_{\mathsf{Agree}}.\mathsf{Click}, n)) = \\ ((b_{\mathsf{Agree}}.\mathsf{Click}, n + 1), \mathit{pref}, \cdot)$$

This indicates that the state ρ has been updated to reflect that one more click has been seen (n becomes $n + 1$), the user's preferences should be released on the declassification channel (*pref*), and the click event should not be released to any additional executions.

For Example 3.2 of leaks *between* executions, the security policy says that button clicks and keypresses are both (S, T), but now, the declassification policy says that button clicks may be released from S to P. When the user clicks b_{secret}, In runs the event as-is in the (S, U) execution and declassifies the event as follows:

$$\mathcal{D}((b_{\mathsf{secret}}.\mathsf{click}(\ ), (S, T)), (S, T), (b_{\mathsf{secret}}.\mathsf{click}, m)) = \\ ((b_{\mathsf{secret}}.\mathsf{click}, m + 1), \mathsf{none}, (b_{\mathsf{secret}}.\mathsf{click}(\ ), (P, U)))$$

Here, ρ is updated to reflect the click, nothing is updated on the declassification channel (none), and the click event is released to the P executions who trust the event. That is, the event is released to all executions with label l_i s.t. l_i trusts the event l_i' (determined by the security policy) and the source of the event l_i'' (formally, $l_i' \sqcup l_i'' \sqsubset l_i$). Here, this is just $(b_{\mathsf{secret}}.\mathsf{Click}(\ ), (P, U))$. The result is that the onClick event handler will run in both the (S, U) and (P, U) executions. The rule Out will suppress the output from the (S, U) execution, but permit the output from the (P, U) execution, which is guaranteed to have a matching button to capture the event.

5.2 Robust Declassification in SME^T

In the presence of an active attacker who may control some of the code, we need to ensure that they do not control what/whether data is declassified [42]. For the declassifications to be robust against attacker influence, we need to ensure that the source of the event l_i trusts the code l_i' on the *same* execution they're interacting with ($l_i' \sqsubseteq l_i$). Additionally, we need to check that the source of the event trusts the code which added the page element in the *other* execution receiving the declassified event.

SME^T composes taint tracking with the SME semantics presented in the previous section to also keep track of the source of the page elements in each execution. First, we modify the event handler storage σ^{EH} so that page elements and event handlers have labels indicating the trustworthiness of their source:

$$\begin{array}{llll} \textit{EH state: } \sigma^{EH} & ::= & \cdot \mid \sigma^{EH}, id \mapsto (v, M)^l \\ \textit{EH map: } \ M & ::= & \cdot \mid Ev \mapsto \{eh_1^{l_1}, \ldots, eh_n^{l_n}\} \end{array}$$

The input rules prevent leaks *within* executions by using the labels in σ^{EH} to decide whether to proceed with a declassification. In order to declassify, the source of the event must trust the source of the page element. We use the shorthand $\mathsf{labelOf}(\sigma^{EH}(id))$ to represent the label on the element identified by id in σ^{EH}, and we write $pc \downarrow^i$ to mean the integrity label in pc. Then, an event from a user at security level pc associated with a page element given by id in σ^{EH} is allowed to be declassified when the following holds: $\mathsf{labelOf}(\sigma^{EH}(id)) \sqsubseteq pc \downarrow^i$ (rule In-Release). Otherwise, rule In-No-Release only runs the event in the executions which have enough privilege to see the event and who trust the user.

We use the declassification function described in Section 5.1 to prevent leaks *between* executions. The updated declassification rules are shown in Figure 9. In addition to looking up the declassified event(s) and the execution(s) they will run in, robust throws out any executions where the source of the event doesn't trust the source of the page element. Rule robust handles the case where the user trusts the source of the code (the event is sent to the execution), and rule not-robust handles the case where they do not (the execution does not receive the event). Then, the lookup semantics (judgement $l, \mathsf{r} \vdash \Sigma, E \rightsquigarrow \mathsf{ks}$) ensure only the trusted event handlers run. We define $(eh, l') \downarrow_l$ as eh when $l' \sqsubseteq l$ and $\cdot$ otherwise. When there is at least one event handler the user trusts ($(\Sigma(pc)(id.Ev(v)) \downarrow_l = c$), rule lookup-R adds the trusted event handlers to ks and attaches a label $\Sigma(pc) \sqcup \Sigma(pc)(id.Ev(v))$ reflecting the source of the code. When there are no trusted event handlers ($(\Sigma(pc)(id.Ev(v)) \downarrow_l = \cdot$), rule lookup-notR moves to the next execution receiving the declassified event. The rules adding a new page element (new) or event handler (add-eh) from the command semantics are responsible for assigning the labels in the event handler store, where l_{src} is the label from rule lookup-R.

Example: We assume that the initial SME state is well-formed, i.e., page elements and event handlers are trusted by the execution context they appear in: execution (l_c, l_i) should trust the page elements and their event handlers, from source l_i', that is, $l_i' \sqsubseteq l_i$.

For our example of leaks *within* an execution, there are three event handlers. onLoad^U is added by the attacker via *ad.com*, who is Untrusted, and onLoad^T is added by the host via *news.com*, who is Trusted. These event handlers are associated with the *body* of the page, which we treat as Trusted. Recall that we assume that the source of the code is trusted by the execution, meaning code from *ad.com* only runs in the Untrusted executions and code from *news.com* runs in both the Untrusted and Trusted executions. Then, the initial SME state with integrity labels is:

$$\begin{array}{lll} \Sigma_0 = & (S, U) \mapsto & \textit{body} \mapsto (\_, \mathsf{load} \mapsto \{\mathsf{onLoad}^U, \mathsf{onLoad}^T\})^T \\ & (P, U) \mapsto & \textit{body} \mapsto (\_, \mathsf{load} \mapsto \{\mathsf{onLoad}^U, \mathsf{onLoad}^T\})^T \\ & (S, T) \mapsto & \textit{body} \mapsto (\_, \mathsf{load} \mapsto \{\mathsf{onLoad}^T\})^T \\ & (P, T) \mapsto & \textit{body} \mapsto (\_, \mathsf{load} \mapsto \{\mathsf{onLoad}^T\})^T \end{array}$$

Now, when the onLoad^U event handler runs, the execution knows the code came from an Untrusted source because of the label U. When the event handler adds the "Click me!" button, rule

$\boxed{\mathcal{P}, \mathcal{D} \vdash K \xRightarrow{(\alpha, pc)} K'}$

$$\frac{\begin{array}{c}\mathcal{P}(id.Ev(v)) = pc' \qquad {\color{red}\mathsf{labelOf}(\Sigma(pc)(id)) \sqsubseteq pc \downarrow^{i}} \\ E = ((id.Ev(v), pc'') \mid pc'' \in \mathcal{L} \text{ s.t. } pc \sqcup pc' \sqsubseteq pc'') \\ (\mathcal{R}', E_d) = \mathsf{declassify}(\mathcal{D}, \mathcal{R}, \Sigma, (id.Ev(v), pc), pc') \\ \Sigma, E \rightsquigarrow \mathsf{ks} \qquad {\color{red} pc \downarrow^{i}, \mathsf{r} \vdash \Sigma, E_d \rightsquigarrow \mathsf{ks}_d}\end{array}}{\mathcal{P}, \mathcal{D} \vdash \mathcal{R}; \Sigma; \cdot \xRightarrow{(id.Ev(v), pc)} \mathcal{R}'; \Sigma; \mathsf{ks} :: \mathsf{ks}_d}\ \textsc{In-Release}$$

$$\frac{\begin{array}{c}\mathcal{P}(id.Ev(v)) = pc' \qquad {\color{red}\mathsf{labelOf}(\Sigma(pc)(id)) \not\sqsubseteq pc \downarrow^{i}} \\ E = ((id.Ev(v), pc'') \mid pc'' \in \mathcal{L} \text{ s.t. } pc \sqcup pc' \sqsubseteq pc'') \\ \Sigma, E \rightsquigarrow \mathsf{ks}\end{array}}{\mathcal{P}, \mathcal{D} \vdash \mathcal{R}; \Sigma; \cdot \xRightarrow{(id.Ev(v), pc)} \mathcal{R}'; \Sigma; \mathsf{ks}}\ \textsc{In-No-Release}$$

$\boxed{\mathsf{declassify}(\mathcal{D}, \mathcal{R}, \Sigma, (id.Ev(v), pc), pc_{Ev}) = (\mathcal{R}', E)}$

$$\frac{\begin{array}{c}\mathcal{D}((id.Ev(v), pc), pc', \rho) = (\rho', v_d, E_d) \\ d' = \mathsf{update}(d, v_d) \qquad {\color{red} E = \mathsf{robust}(\Sigma, E_d, pc \downarrow^{i})}\end{array}}{\mathsf{downgrade}_{\mathcal{D}}((\rho, d), \Sigma, (id.Ev(v), pc), pc') = ((\rho', d'), E)}\ \textsc{declassify}$$

${\color{red}\boxed{\mathsf{robust}(\Sigma, E, pc_{Ev}) = E'}}$

$$\frac{\mathsf{labelOf}(\Sigma(pc)(id)) \sqsubseteq l}{\begin{array}{c}\mathsf{robust}(\Sigma, ((id.Ev(v), pc) :: E), l) = \\ (id.Ev(v), pc) :: \mathsf{robust}(\Sigma, E, l)\end{array}}\ \textsc{robust}$$

$$\frac{\mathsf{labelOf}(\Sigma(pc)(id)) \not\sqsubseteq l}{\begin{array}{c}\mathsf{robust}(\Sigma, ((id.Ev(v), pc) :: E), l) = \\ \mathsf{robust}(\Sigma, E, l)\end{array}}\ \textsc{not-robust}$$

${\color{red}\boxed{pc, \mathsf{r} \vdash \Sigma, E \rightsquigarrow \mathsf{ks}}}$

$$\frac{\Sigma(pc)(id.Ev(v)) \downarrow_{l} = c \qquad \kappa = \cdot, c, P, \cdot \qquad l, \mathsf{r} \vdash \Sigma, E \rightsquigarrow \mathsf{ks}}{\begin{array}{c}l, \mathsf{r} \vdash \Sigma, (id.Ev(v), pc) :: E \rightsquigarrow \\ (\kappa, pc, \Sigma(pc) \sqcup \Sigma(pc)(id.Ev(v))) :: \mathsf{ks}\end{array}}\ \textsc{lookup-R}$$

$$\frac{\Sigma(pc)(id.Ev(v)) \downarrow_{l} = \cdot \qquad l, \mathsf{r} \vdash \Sigma, E \rightsquigarrow \mathsf{ks}}{l, \mathsf{r} \vdash \Sigma, (id.Ev(v), pc) :: E \rightsquigarrow \mathsf{ks}}\ \textsc{lookup-NotR}$$

$$\frac{}{l, \mathsf{r} \vdash \Sigma, \cdot \rightsquigarrow \cdot}\ \textsc{lookup-Remp}$$

$\boxed{{\color{red} l_{src}, d} \vdash \Sigma, \sigma, c \xrightarrow{\alpha}_{pc} \Sigma', \sigma', c', E}$

$$\frac{\begin{array}{c}[\![e]\!]^{pc}_{\sigma,\Sigma} = v \\ \Sigma(pc) = \sigma^{EH} \qquad id \notin \sigma^{EH} \qquad \Sigma' = \Sigma[pc \mapsto \sigma^{EH}[id \mapsto (v, \cdot)^{\color{red} l_{src}}]]\end{array}}{{\color{red} l_{src}, d} \vdash \Sigma, \sigma, \mathsf{new}(id, e) \xrightarrow{\bullet}_{pc} \Sigma', \sigma, \mathsf{skip}, \cdot}\ \textsc{new}$$

$$\frac{\begin{array}{c}\Sigma(pc) = \sigma^{EH} \qquad \sigma^{EH}(id) = (v, M)^{\color{red} l_{id}} \\ \Sigma' = \Sigma[pc \mapsto \sigma^{EH}[id \mapsto (v, M[Ev \mapsto M(Ev) \cup eh^{\color{red} l_{src}}])^{\color{red} l_{id}}]]\end{array}}{{\color{red} l_{src}, d} \vdash \Sigma, \sigma, \mathsf{addEh}(id, Ev, eh) \xrightarrow{\bullet}_{pc} \Sigma', \sigma, \mathsf{skip}, \cdot}\ \textsc{add-EH}$$

Figure 9: Robust declassification. Key changes are in red.

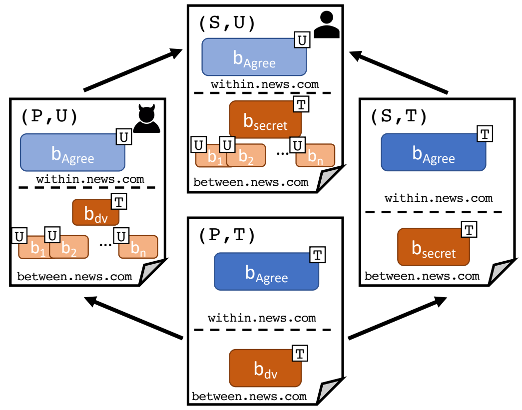


Figure 10: Insecure example from Section 3 with robustness checks. The labels tell us the trustworthiness of the source of the page elements and event handlers, depicted here as small white labels on each page element.

NEW uses the label on the page element T and event handler U to determine the trustworthiness of the new button $T \sqcup U = U$. The state after adding the "Click me!" button to the (S, U) execution is:

$$\Sigma_1 = \quad (S,U) \mapsto \quad body \mapsto (\ldots)^{T}, b_{\mathrm{Agree}} \mapsto (\text{“Click me!”}, \{\ \})^{U} \\ \quad \ldots$$

Figure 10 shows the resulting page after all of the buttons are loaded, including their labels. When the user clicks the "Click me!" button on the (S, U) copy of the page, the input rules will use the label on the button to determine if the declassification is allowed. The user is treated as a Trusted source of events, so because $U \not\sqsubseteq T$, rule In-No-Release prevents the event from being declassified and the attacker doesn't learn the user's settings.

For our example of leaks *between* executions, the host installs an onKeypress event handler to some field which adds a different button to the page depending on what the user types, and the attacker adds all possible buttons to the page. After the user presses a key, the SME store has one Trusted button per execution, and several Untrusted buttons in the Untrusted executions:

$$\begin{array}{rl}\Sigma_0 = & (S,U) \mapsto b_{\mathrm{secret}} \mapsto (\ldots)^{T}, b_1 \mapsto (\ldots)^{U}, \ldots, b_n \mapsto (\ldots)^{U} \\ & (P,U) \mapsto b_{\mathrm{dv}} \mapsto (\ldots)^{T}, b_1 \mapsto (\ldots)^{U}, \ldots, b_n \mapsto (\ldots)^{U} \\ & (S,T) \mapsto b_{\mathrm{secret}} \mapsto (\ldots)^{T} \\ & (P,T) \mapsto b_{\mathrm{dv}} \mapsto (\ldots)^{T}\end{array}$$

When the user clicks the b_{secret} button on the (S, U) copy of the page, rule In-Release attempts to declassify the event to the (P, U) execution since the button is Trusted. Next, the robust rules use the labels on the button capturing the event to determine if the (P, U) execution should receive the declassified event. In this case, the button b_i capturing the event was added by the attacker. Since $U \not\sqsubseteq T$, rule NOT-ROBUST skips the (P, U) execution and the attacker does not learn which key the user pressed.

6 SECURITY

We define two security conditions and prove that SME^{T} satisfies them. First, we define a knowledge-based progress-insensitive noninterference with declassification (Section 6.1) which ensures that the attacker's knowledge of the secret inputs is not refined as the system runs outside of what is declassified (and the fact that the system makes progress). Second, we describe a novel influence-based progress-insensitive noninterference (Section 6.2) which is the integrity dual to the knowledge-based security condition to demonstrate that SME^T do not allow the attacker to influence the more trusted components of the system (except the fact that the system makes progress). Finally, we show if we treat declassification as a trusted behavior, the influence-based security condition may be extended so that robust declassification follows.

6.1 Knowledge-based security (confidentiality)

Knowledge-based security conditions allow precise specification of what information (if any) is leaked. We informally define several knowledge conditions (summarized in Figure 11) to set up our knowledge-based progress-insensitive noninterference definition. Formal definitions may be found in Appendix C- H.

For someone with enough privilege to observe data up to label l, their knowledge is the set of all possible inputs which might have produced the observations they made. Knowledge can also be thought of as a measure of *uncertainty*. As the attacker learns more, they will become more confident about the inputs received by the system and the knowledge set will become smaller (i.e., the attacker has become more certain about what the inputs might have been). We define the knowledge of an observer with privilege $l \in \mathcal{L}_c$:

$$\mathcal{K}(T, \Sigma_0, \mathcal{R}, \mathcal{P}, l) = \{\tau \mid \exists T' \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{P}), T \approx_l^c T', \tau = \mathsf{in}(T')\}$$

The knowledge of an observer with privilege l is the set of all inputs from execution traces T' ($\tau = \mathsf{in}(T')$) that have *observationally equivalent at* l to T ($T \approx_l^c T'$) and start from the same initial state with the same security and declassification policies ($T' \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{P})$). For now, an input is a user-generated event ($id.Ev(v)$). We say that two runs are *observationally equivalent at* l, $T \approx_l^c T'$, if they look the same to an observer with privilege l (i.e., they make the same outputs on any l-visible channel and the l-visible executions behave the same) $T \downarrow_l^c = T' \downarrow_l^c$. The *observation* of a trace is the sequence of actions observable by an attacker and include inputs, outputs, silent actions,[2] and declassifications $\mathsf{rls}(\ldots)$.

$$\textit{Sequence of actions} : \tau \quad ::= \quad \cdot \mid \tau :: \alpha \mid \tau :: \mathsf{rls}(id.Ev(v), \mathcal{R}, E)$$

The rules for the observation of a trace are shown in Figure 12. Note that $T \downarrow_l^p$ is parameterized by p, where $p = c$ is for confidentiality and $T \downarrow_l^c$ is the observation of a trace at l, and $p = i$ will be for integrity (Section 6.2) and $T \downarrow_l^i$ is the behavior of a trace at l. The observation of an output is $ch(v)$ if the output is made on an observable channel $\mathcal{P}(ch) \sqsubseteq l$ or by an observable execution $pc \downarrow^p \sqsubseteq l$ (rule TP-OUT2), otherwise the output is skipped (rule TP-OUT-S1). Inputs are observable if the security policy and source is observable (rule TP-IN), and declassifications are observable if they are successful (rule TP-IN-R). Other actions are observable if they happen in an observable execution (rules TP-OUT1); otherwise, they are skipped (rule TP-OUT-S2).

A knowledge-based progress-sensitive noninterference says that an attacker should not be able to refine their knowledge of the secret inputs by watching the system run:

$$\mathcal{K}(T, \Sigma_0, \mathcal{R}, \mathcal{P}, l) \subseteq_\leq \mathcal{K}(T \Longrightarrow K, \Sigma_0, \mathcal{R}, \mathcal{P}, l)$$

[2]We consider silent actions observable only when they come from an observable execution, which makes proofs for observable executions more uniform. Since our equivalence definitions force observable executions to be the same anyway, this choice does not effect our security conditions.

We write $A \subseteq_\leq B$ to mean that each element of A is a prefix of an element in B (since the last step of $T \Longrightarrow K$ may be an input). When the system takes a step ($T \Longrightarrow K$), the attacker's knowledge should not be refined; they should be equally uncertain about the possible secret inputs before and after the step. Because we run event handlers in a single-thread, it is possible for an event handler to get "stuck" in an infinite loop, which could leak something to the attacker if the loop condition is secret. Therefore, we will permit this leak and prove *progress-insensitive* noninterference instead. We define *progress knowledge* as the set of traces producing the same outputs *and* making enough progress to accept another input.

A knowledge-based progress-insensitive security condition is:

$$\mathcal{K}_p(T, \Sigma_0, \mathcal{R}, \mathcal{P}, l) \subseteq_\leq \mathcal{K}(T \Longrightarrow K, \Sigma_0, \mathcal{R}, \mathcal{P}, l)$$

When the system takes a step, the attacker's knowledge should not be refined (except that they learn the system makes progress). This definition has yet to capture declassification. For example, if a user's click on a hat b_{Hat} is declassified for analytics (like for the shop from Section 3), the attacker's knowledge would be refined to inputs that include the click on b_{Hat}. This leak is permitted by declassification, but not by the definition above. Therefore, we define *release knowledge* as the set of traces producing the same outputs, making progress, *and* releasing *the same event.* Our definition for knowledge-based progress-insensitive noninterference with declassification says that, outside of declassification, the attacker should not learn anything by watching the system take a step (except that the system has made progress) and when something is declassified, the attacker should only learn what is declassified. We say $\mathsf{releaseA}(T \Longrightarrow K)$ if $(\mathsf{last}(T) \Longrightarrow K) \downarrow_l^c = \mathsf{rls}(\ldots)$, where $\mathsf{last}(T)$ is the last state in T. That is, $\mathsf{releaseA}(T \Longrightarrow K)$ means something was declassified in the last step.

Definition 1 (PINI with Declassification). A system satisfies progress-insensitive noninterference, outside of what is declassified, against l-observers for $l \in \mathcal{L}_c$ iff given any initial global store Σ_0, security policy $\mathcal{P}$, and declassification policy $\mathcal{R}$, it is the case that for all traces T, actions α, and configurations K s.t. $(T \overset{\alpha}{\Longrightarrow} K) \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{P})$, then, the following holds

- *If* $\mathsf{releaseA}(T \overset{\alpha}{\Longrightarrow} K)$*:* $\mathcal{K}(T \overset{\alpha}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{P}, l) \supseteq_\leq \mathcal{K}_{rp}(T, \Sigma_0, \mathcal{R}, \mathcal{P}, \alpha, l)$
- *Otherwise:* $\mathcal{K}(T \overset{\alpha}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{P}, l) \supseteq_\leq \mathcal{K}_p(T, \Sigma_0, \mathcal{R}, \mathcal{P}, l)$

Example Recall Example 3.2 of leaks *between* executions. The security policy is that keypress events should be Secret, but clicks may be declassified from Secret to Public. Which button is added by the host depends on what the user types. The attacker adds all buttons $b_1, \ldots, b_n$ and registers an onClick event handler to each button which outputs i to a (P, U) channel if registered for b_i. When the user types, the attacker isn't sure which key is pressed. Their knowledge at this point includes all possible keypresses:

$$\mathcal{K}(K, \Sigma_0, \mathcal{R}, \mathcal{P}, P) = \{f.\mathsf{keyPress}(1), \ldots, f.\mathsf{keyPress}(n)\}$$

The keypress triggers the onKeypress event handler which adds button b_i to the user's page if they pressed i. Suppose the attacker also registered a Click event handler to b_{secret} to directly leak the user's keypress through a (P, U) channel. If the output were allowed,

Knowledge	$\mathcal{K}(T, \Sigma_0, \mathcal{R}, \mathcal{P}, l) =$ $\{\tau \mid \exists T' \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{P}), T \approx_l^c T', \tau = \mathsf{in}(T')\}$	All possible inputs producing the same observations
Progress Knowledge	$\mathcal{K}_p(T, \Sigma_0, \mathcal{R}, \mathcal{P}, l) =$ $\{\tau \mid \exists T' \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{P}), T \approx_l^c T', \tau = \mathsf{in}(T'), \mathsf{prog}(T')\}$	All possible inputs producing the same observations *and* accept another input: prog(T') holds if T' can reach the consumer state
Release Knowledge	$\mathcal{K}_{rp}(T \overset{\alpha}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{P}, l) =$ $\{\tau \mid \exists T' \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{P}), T \approx_l^c T',$ $\tau = \mathsf{in}(T'), \mathsf{prog}(T'), \mathsf{releaseT}(T', \alpha))\}$	All possible inputs producing the same observations, accept another input, *and* release the same event: releaseT(T', α) holds if T' can be extended to release the same event α

Figure 11: Knowledge definitions. Knowledge and progress knowledge are for defining a knowledge-based progress-insensitive noninterference. Release knowledge accounts for what is leaked to the attacker through declassification.

$$\boxed{T \downarrow_l^p = \tau}$$

$$\frac{}{\mathcal{P} \vdash K \downarrow_l^p = \cdot}\ \textsc{TP-Base}$$

$$\frac{pc \downarrow^p \sqsubseteq l \qquad \alpha \notin \{id.Ev(v), ch(v)\}}{(\mathcal{P}, \mathcal{D} \vdash K \overset{(\alpha, pc)}{\Longrightarrow} T') \downarrow_l^p = \alpha :: T' \downarrow_l^p}\ \textsc{TP-Out1}$$

$$\frac{pc \downarrow^p \sqsubseteq l \lor \mathcal{P}(ch) \downarrow^p \sqsubseteq l}{(\mathcal{P}, \mathcal{D} \vdash K \overset{(ch(v), pc)}{\Longrightarrow} T') \downarrow_l^p = ch(v) :: T' \downarrow_l^p}\ \textsc{TP-Out2}$$

$$\frac{pc \downarrow^p \not\sqsubseteq l \downarrow^p \qquad \mathcal{P}(ch) \downarrow^p \not\sqsubseteq l}{(\mathcal{P}, \mathcal{D} \vdash K \overset{(ch(v), pc)}{\Longrightarrow} T') \downarrow_l^p = T' \downarrow_l^p}\ \textsc{TP-Out-S1}$$

$$\frac{\alpha \notin \{id.Ev(v), ch(v)\} pc \downarrow^p \not\sqsubseteq l}{(\mathcal{P}, \mathcal{D} \vdash K \overset{(\alpha, pc)}{\Longrightarrow} T') \downarrow_l^p = T' \downarrow_l^p}\ \textsc{TP-Out-S2}$$

$$\frac{\begin{array}{c}\mathcal{P}(id.Ev(v)) = pc' \qquad \Sigma(pc)(id) \not\sqsubseteq pc \downarrow^i \\ \tau = id.Ev(v) \text{ if } pc' \downarrow^p \sqcup pc \downarrow^p \sqsubseteq l \qquad \tau = \cdot \text{ otherwise}\end{array}}{(\mathcal{P}, \mathcal{D} \vdash \_; \Sigma; \_ \overset{(id.Ev(v), pc)}{\Longrightarrow} T') \downarrow_l^p = \tau :: T' \downarrow_l^p}\ \textsc{TP-In}$$

$$\frac{\begin{array}{c}\mathcal{P}(id.Ev(v)) = pc' \qquad \Sigma(pc)(id) \sqsubseteq pc \downarrow^i \\ (\mathcal{R}', E) = \mathsf{declassify}(\mathcal{D}, \mathcal{R}, \Sigma, (id.Ev(v), pc), pc') \\ \tau = \mathsf{rls}(id.Ev(v), \mathcal{R}', E \downarrow_l^p) \text{ if } \mathcal{R} \neq \mathcal{R}' \\ \tau = id.Ev(v) \text{ if } \mathcal{R} = \mathcal{R}' \land pc \downarrow^p \sqcup pc \downarrow^p \sqsubseteq l \\ \tau = \cdot \text{ otherwise}\end{array}}{(\mathcal{P}, \mathcal{D} \vdash \mathcal{R}; \Sigma; \_ \overset{(id.Ev(v), pc)}{\Longrightarrow} T') \downarrow_l^p = \tau :: T' \downarrow_l^p}\ \textsc{TP-In-R}$$

Figure 12: The observation (p=c) or behavior (p=i) of T at l

the attacker would be able to eliminate the traces where the user pressed a different key which refines their knowledge.

$$\mathcal{K}(K \overset{ch(i)}{\Longrightarrow} K', \Sigma_0, \mathcal{R}, \mathcal{P}, P) = \{\cancel{f.\mathsf{keyPress}(1)}, \ldots,$$
$$f.\mathsf{keyPress}(secret) :: b_{\mathsf{secret}}.\mathsf{Click}(\_), \ldots, \cancel{f.\mathsf{keyPress}(n)}\}$$

Our knowledge-based security condition would correctly identify this output as insecure because: $\mathcal{K}(K \overset{ch(i)}{\Longrightarrow} K', \ldots) \not\supseteq \mathcal{K}(K, \ldots)$

In reality, the SME monitor would prevent the output from the (S, U) execution to the (P, U) channel. The user's click would not be able to directly leak their keypress to the attacker, but it could be declassified to the (P, U) execution. Since the attacker added all possible buttons $b_1, \ldots, b_n$, they are guaranteed to trigger the leaky output and learn which key the user pressed. Because the releaseA condition allows the attacker's knowledge to be refined by declassifications, our security condition for confidentiality does not catch this leak. Next, we describe our security condition for integrity and how this condition can be used to describe both progress-insensitive noninterference as well as robust declassification.

6.2 Influence-based security (integrity)

We measure the attacker's ability to change the behavior of the system with a dual condition to knowledge called *influence*, (based on attacker power [5]). At a high level, an attacker's influence is the set of all untrusted inputs which might have produced the same trusted behaviors. The attacks in the influence set have the same relative ability to influence the system's behavior. If the attacker has no influence over the system, then the set should include all possible attacks: all of the attacks are equally powerless. As the system runs, the refinement of attacker's influence indicates that some attacks are more powerful than the others because the ones eliminated couldn't have led to the observed behavior. We define the attacker's influence over behaviors at l (for $l \in \mathcal{L}_i$) below:

$$\mathcal{I}(T, \Sigma_0, \mathcal{R}, \mathcal{P}, l) = \{\tau \mid \exists T' \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{P}),$$
$$T \approx_l^i T' \land \tau = \mathsf{in}(T')\}$$

The influence of an attacker over behaviors at l is the set of all τ which are inputs from execution traces T' ($\tau = \mathsf{in}(T')$) that are *behaviorally equivalent at* l to T ($T \approx_l^i T'$) and start from the same initial state with the same security and declassification policies ($T' \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{P})$). We say that two runs are *behaviorally equivalent at* l if they produce the same l-trusted actions (i.e., they make the same outputs on any l-trusted channel and the l-trusted executions behave the same). $T \downarrow_l^p$ is defined in Figure 12. We summarize our influence definitions in Figure 13.

An influence-based progress-sensitive noninterference says the attacker's influence over a system should never be refined:

$$\mathcal{I}(T, \Sigma_0, \mathcal{R}, \mathcal{P}, l) \subseteq_{\preceq} \mathcal{I}(T \Longrightarrow K, \Sigma_0, \mathcal{R}, \mathcal{P}, l)$$

Similar to progress knowledge, we define *progress influence* as the set of traces producing the same behaviors *and* making enough progress to accept another input. Then, an influence-based progress-insensitive security condition states that when the system takes a step, the attacker's influence should not be refined, outside of what control they have over whether the system makes progress:

$$\mathcal{I}_p(T, \Sigma_0, \mathcal{R}, \mathcal{P}, l) \subseteq_{\preceq} \mathcal{I}(T \Longrightarrow K, \Sigma_0, \mathcal{R}, \mathcal{P}, l)$$

Influence	$\mathcal{I}(T, \Sigma_0, \mathcal{R}, \mathcal{P}, l) =$ $\{\tau \mid \exists T' \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{P}), T \approx_l^i T', \tau = \mathsf{in}(T')\}$	All possible inputs producing the same trusted actions
Progress Influence	$\mathcal{I}_p(T, \Sigma_0, \mathcal{R}, \mathcal{P}, l) =$ $\{\tau \mid \exists T' \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{P}), T \approx_l^i T', \tau = \mathsf{in}(T'), \mathsf{prog}(T')$	All possible inputs producing the same trusted actions *and* accept another input: $\mathsf{prog}(T')$ holds if T' can reach the consumer state
Robust Influence	$\mathcal{I}_{rp}(T \overset{\alpha}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{P}, l) =$ $\{\tau \mid \exists T' \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{P}), T \approx_l^i T',$ $\tau = \mathsf{in}(T'), \mathsf{prog}(T'), \mathsf{robustT}(T', \alpha))\}$	All possible inputs producing the same trusted actions, accept another input, *and* capable of the same robust declassifications: $\mathsf{robustT}(T', \alpha)$ holds if T' can be extended to create the same trusted page event α

Figure 13: Influence definitions. Influence and progress influence are for defining an influence-based progress-insensitive noninterference. Robust influence is for defining robust declassification.

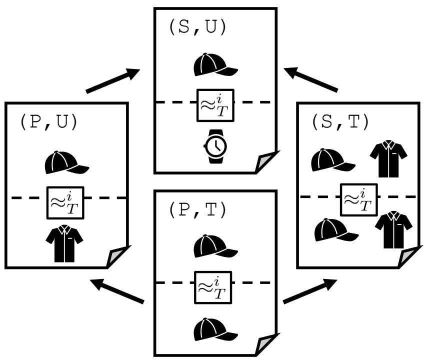


Figure 14: The states above and below the dotted line are behaviorally equivalent at T even there are different products in the (P, U) and (S, U) states.

6.3 Robust declassification

In addition to showing that the attacker doesn't have influence over trusted behaviors, we also want to show that the attacker doesn't influence declassification. We can define robust declassification by extending our influence-based security condition.

A naïve formalization of robust declassification is as follows. We model an *active attacker* by treating the addition of a page element or event handler ($\mathsf{new}(id, pc)$, $\mathsf{new}(id, eh, pc)$) as an input. A system is robust if any of these *attacks* have equivalent power. That is, when a new declassification happens, we will know the attacker's code influenced the declassification if the set of attacks *without* their code could not have led to the same declassification. However, it turns out that this definition is too strong and leads to false positives.

Consider the online shop described in Section 3. The buttons are all loaded by the Trusted host, so they can safely influence declassification: the declassifications in this example are robust. The issue is that behavioral equivalence at T only guarantees that the Trusted executions behave the same. See the example of two equivalent traces in Figure 14. The (S, T) execution has the same products in both traces, as does the (P, T) shop, but even among two equivalent runs, the (S, U) and (P, U) executions may have different products. When the user clicks b_{Hat} in the (S, U) execution, the click is declassified. But it isn't possible to produce the same declassification in the equivalent state because there is no b_{Hat} for the user to click on. This makes it appear as though the attacker had some influence over the declassification, even though the declassification is actually robust against their influence.

$$\frac{\begin{array}{c}\alpha \in \{\mathsf{new}(id, l_{src}), \mathsf{new}(id, eh, l_{id}, l_{src})\} \\ pc \downarrow^i \not\sqsubseteq l \qquad \tau = \mathsf{r}(id, pc) \text{ if } l_{src} \sqsubseteq pc \downarrow^i \\ \tau = \mathsf{r}(id, eh, pc) \text{ if } l_{id} \sqcup l_{src} \sqsubseteq pc \downarrow^i \qquad \tau = \cdot \text{ otherwise}\end{array}}{(\mathcal{P}, \mathcal{D}, \vdash K \overset{(\alpha, pc)}{\Longrightarrow} T') \downarrow_l^i = \tau :: T' \downarrow_l^i} \text{ TP-New}$$

Figure 15: New rule for the behavior of a trace for robust declassification.

To make these benign influence refinements concrete, we introduce *robust influence* for when trusted page elements are created. Robust influence is the set of traces producing the same elements, making progress, *and* capable of producing the same robust declassifications in the untrusted executions. This is similar to release knowledge from Section 6.1. We say $\mathsf{robustA}(T)$ if $(\mathsf{last}(T) \Longrightarrow K) \downarrow^i = \mathsf{r}(\ldots)$, where $\mathsf{last}(T)$ is the last state in T. That is, $\mathsf{robustA}(T \Longrightarrow K)$ means something capable of robust declassification was added to an Untrusted execution.

To model an *active* attacker's ability to add code to the page, we emit an action for dynamically-generated elements and event handlers. $\mathsf{new}(id, pc)$ is a new page element identified, id, added to the pc execution, while $\mathsf{new}(id, eh, pc)$ is a new event handler eh registered to the element identified by id in the execution at security level pc. Sequences of actions include page elements/event handlers *which are capable of robust declassification* $\mathsf{r}(\ldots)$.

$$\begin{array}{llll}\textit{Actions:} & \alpha & ::= & id.Ev(v) \mid ch(v) \\ & & \mid & \mathsf{new}(id, pc) \mid \mathsf{new}(id, eh, pc) \mid \bullet \\ \textit{Sequence of actions}: \tau & & ::= & \cdot \mid \tau :: \alpha \mid \tau :: \mathsf{rls}(id.Ev(v), \mathcal{R}, E) \\ & & & \mid \tau :: \mathsf{r}(id, pc) \mid \tau :: \mathsf{r}(id, eh, pc)\end{array}$$

We modify the behavior of a trace as shown in Figure 15. When a new page element is created or event handler is registered, this is not considered an observable action unless it is capable of a robust declassification (rule TP-New).

Definition 2 (Influence-based PINI with Robust Declassification). A system satisfies progress-insensitive noninterference with robust declassification for behaviors at $l \in \mathcal{L}_i$ iff given any initial global store Σ_0, security policy $\mathcal{P}$, and declassification policy $\mathcal{R}$, it is the case that for all traces T, actions α, and configurations K s.t. $(T \overset{\alpha}{\Longrightarrow} K) \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{P})$, then, the following holds

- *If* $\mathsf{robustA}(T \overset{\alpha}{\Longrightarrow} K)$*:* $\mathcal{I}(T \overset{\alpha}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{P}, l) \supseteq_{\leq} \mathcal{I}_{rp}(T, \Sigma_0, \mathcal{R}, \mathcal{P}, \alpha, l)$
- *Otherwise:* $\mathcal{I}(T \overset{\alpha}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{P}, l) \supseteq_{\leq} \mathcal{I}_p(T, \Sigma_0, \mathcal{R}, \mathcal{P}, l)$

Example: To illustrate how this new definition is sufficient for defining robust declassification, we will walk through examples from Section 3. In Example 3.1 of a leak *within* an execution, the Untrusted attacker registers the event handler onLoad^U and the Trusted host registers onLoad^T to add buttons to the page.

After the page finishes loading, we know that the Trusted *"Agree"* button, b_{Agree}, must have been dynamically loaded because all of the behaviorally-equivalent Trusted executions have run onLoad^T. On the other hand, we aren't sure whether the Untrusted "Click me!" button, was added because the Untrusted pages are equivalent whether or not onLoad^U has run. At this point, the attacks where the *"Click me!"* button has been added are equally as powerful as the attacks without it:[3]

$$\mathcal{I}(K, \Sigma_0, \mathcal{R}, \mathcal{P}, T) = \{\mathsf{new}(b)^T, \mathsf{new}(b)^U :: \mathsf{new}(b)^T, \ldots\}$$

If the system allowed the click on the Untrusted b_{Agree} to be declassified, it would mean there *must* be a *"Click me!"* button on the (S, U) copy of the page. Therefore, the only viable attack leading to this behavior are the ones including the Untrusted b_{Agree} button:

$$\begin{aligned}\mathcal{I}(T \xRightarrow{b^U} K, \mathcal{R}, \mathcal{P}, T) = \{&\cancel{\mathsf{new}(b)^T}, \\ &\mathsf{new}(b)^U :: \mathsf{new}(b)^T :: b^U.\mathsf{Click}(\,), \ldots\}\end{aligned}$$

Because $\mathcal{I}(T \xRightarrow{b^U} K, \mathcal{R}, \mathcal{P}, T) \not\supseteq_{\leq} \mathcal{I}_P(T, \mathcal{R}, \mathcal{P}, T)$, the attacker must have had influence over the declassification, so it isn't robust.

Example 3.2 of leaks *between* executions is similar. The Trusted host adds a different button to the page depending on what the user has typed and the Untrusted attacker adds all possible buttons.

After the user presses a key on their keyboard, we know that there is one button on the (S, T) page (based on the actual *secret* value) and another button on the (U, T) page (based on the default value dv) because all of the behaviorally-equivalent Trusted executions have run the Trusted event handler in response to the user's keypress. We also know that the (S, U) and (P, U) copies of the page must include b_{secret} and b_{dv} (respectively) because those buttons are capable of robust declassification since they were added by the host. On the other hand, we aren't sure whether the attacker has added their buttons, because the Untrusted pages are equivalent with or without those buttons:

$$\begin{aligned}\mathcal{I}(K, \Sigma_0, \mathcal{R}, \mathcal{P}, l) = \{&\mathsf{new}(b_{\text{secret}})^S :: \mathsf{new}(b_{\text{dv}})^P, \\ &\mathsf{new}(b_{\text{secret}})^S :: \mathsf{new}(b_{\text{dv}})^P :: \mathsf{new}(b_1)^U :: \ldots :: \mathsf{new}(b_n)^U, \ldots\}\end{aligned}$$

Now, when the user's click on b_{secret} in the (S, U) page is declassified to the matching button b_i in the (P, U) page, we know there must be a b_i button on the (P, U) copy of the page to capture the event. Then, the only viable attack is the one where b_i has been added to the page:

$$\begin{aligned}\mathcal{I}(K, \Sigma_0, \mathcal{R}, \mathcal{P}, l) = \{&\cancel{\mathsf{new}(b_{\text{secret}})^S :: \mathsf{new}(b_{\text{dv}})^P}, \\ &\mathsf{new}(b_{\text{secret}})^S :: \mathsf{new}(b_{\text{dv}})^P :: \mathsf{new}(b_1)^U :: \ldots :: \mathsf{new}(b_n)^U, \ldots\}\end{aligned}$$

Since the attacker's influence has been refined we know this example is not robust either.

Finally, consider the secure web shop where the host adds products to the page and declassifies click counts so that a (P, U) library can do analytics for them. All of the elements are added by the Trusted host, so they are capable of robust declassification. From the robustA case in Definition 2, the attacker's influence can be refined by the addition of these elements to include only the traces that load the same products on the web store. This means that declassifying a user's click won't refine the attacker's influence and our security condition correctly identifies this as robust.

6.4 Metatheory

We prove that our semantics are sound. Formally:

Theorem 3 (Soundness). $\forall \mathcal{P}, \mathcal{D}, \Sigma_0$, *the SME state* Σ_0 *satisfies knowledge-based progress-insensitive noninterference with declassification at* $l_c \in \mathcal{L}_c$ *and influence-based progress-insensitive noninterference with robust declassification at* $l_i \in \mathcal{L}_i$ *w.r.t. the security policy* $\mathcal{P}$ *and declassification policy* $\mathcal{D}$.

Complete proofs may be found in Appendix H. Robust declassification follows from influence-based progress-insensitive noninterference. If we treat declassifications as trusted and prove that untrusted sources cannot influence trusted behaviors, then it must be the case that the declassifications are robust.

Corollary 4 (Robust Declassification). $\forall \mathcal{P}, \mathcal{D}, \Sigma_0$ *s.t.* Σ_0 *satisfies influence-based progress-insensitive noninterference at* l_i *w.r.t. the security policy* $\mathcal{P}$ *and declassification policy* $\mathcal{D}$*, then an attacker at* $l_i' \in \mathcal{L}_i$ *with* $l_i' \not\sqsubseteq l_i$ *has no influence over whether the user's events at* l_i *are declassified.*

7 IMPLEMENTATION AND EVALUATION

We have prototyped SME^T in OCaml on top of Featherweight Firefox [15], which is a lightweight implementation of the web browser model. The implementation provides a sanity check on the semantics and helps understand the behavior of programs that restrict declassification to certain cases. The original Featherweight Firefox does not include recent browser features, but is expressive enough to enforce all features of our formalization and demonstrate its feasibility in a browser-like setting. We leave enforcement in a real browser to future work.

We modify the model to attach labels to nodes on a web page. The host page has higher integrity than the user and third-party integrity. We leverage the implementation from prior work [13] to label input events and the outputs generated. The labels of the nodes are fixed across all executions of the browser model. To emulate the behavior of adding a new node, we define the handlers of the node up front and insert it into the page along with the handlers, when the event handler is called. We omit the trigger command from the semantics and assume that all events are user-generated.

We implement the release module to perform declassification as per the release policies. When an input event is received, we check the context (label) of the event (which is user in our case) and compare it against the label of the node on which the event was triggered. If the label on the node is not trusted by the source of the event, then the release module is not called. Otherwise, the release module writes the declassified value to a shared channel. For simplicity, we declassify all values in the release module to the confidentiality level P. In the current implementation, we assume that the declassify command reads the last declassified value for that level.

[3]Due to space constraints, we write $\mathsf{new}(b)^T$ instead of $\mathsf{new}(b_{\text{Agree}}, (S, T))$ and $\mathsf{new}(b_{\text{Agree}}, (P, T))$, and likewise for $\mathsf{new}(b)^U$ for the Untrusted executions

Evaluation: To compare against SME models with declassification, we also implement versions of the model with the original stateful declassification approach [40] and the one that prohibits declassification on dynamically created elements [29]; we modify the release module to declassify without checking the node label (for the former) and assigning a special label SD, which never declassifies (for the latter). We observe that the example programs presented earlier leak information with unrestricted declassification while our approach is more permissive compared to the approach where declassification is never allowed for events from dynamic elements. In terms of performance overhead, our monitor performs worse compared to the existing approaches due to the operations involving multiple levels and the additional integrity label. More concretely, the overhead of running our monitor as compared to the prior approaches is around 15% and 9%, respectively, for the example presented earlier.

8 DISCUSSION

Robust declassification and attacker control Prior work on robust declassification that is most similar to our setting involves attacker control [5] which is the set of attacks (i.e., untrusted inputs) with a similar effect on knowledge. They say declassification is robust if the attacker control (which are the possible attacks resulting in the *same declassification*) includes all of the attacks *reaching* the declassification; otherwise the attacker must have influenced the declassification. Our definition is similar. We relate the set of possible attacks before and after declassification and consider the declassification robust if attacks reaching the declassification could also result in the same declassification. The key benefit of our condition over prior work is that robust declassification follows from our influence-based security condition which makes the definitions more uniform and simplifies the proofs.

(Transparent) endorsement and qualified robustness The focus of this work is robust declassification, but like our "influence-based" security condition is the integrity dual of "knowledge-based" security conditions for confidentiality, *(transparent) endorsement* is the integrity dual of *(robust) declassification.* Endorsement allows a program to treat untrusted data as if it were more trusted, and transparent endorsement ensures that the data is sufficiently public before endorsing. The idea being that if the attacker supplies information they do not actually have the privilege to see, we should not trust it. For example, prior work [18] proposing transparent endorsement explains that without restricting endorsement to what data the attacker has the privilege to see, they could cheat in a sealed-bid auction by simply bidding "one more than the other person" (even though they don't know what the other person bid).

In Appendix A, we include (transparent) endorsement by adding an endorsement policy $\mathcal{E}$ and module $\mathcal{S}$, which functions similarly to $\mathcal{D}$ and $\mathcal{R}$. We update the input rules to ensure the source of the event has enough privilege to see the page element ($\Sigma(pc)(id) \downarrow^c \sqsubseteq pc \downarrow^c$). An event may be both declassified and endorsed as long as the *original* event is both robust and transparent (we do not declassify before checking for transparency or vice versa).

The changes to the security conditions are similarly straightforward. We add *sanitized influence* to prove an influence-based progress-insensitive noninterference *with endorsement.* Sanitized influence measures the amount of influence the attacker gains through endorsement and is defined as the set of all possible inputs producing the same trusted actions, accepting another input, and capable of the same endorsements (similar to release knowledge). If we treat endorsements as public, transparent endorsement follows from our knowledge-based security condition if we add *transparent knowledge* which captures the information leaked by adding an element to a secret execution that is capable of transparent endorsement (similar to robust influence). The supporting definitions for these security conditions may be found in Appendix D and G.

Note that because an event associated with an attacker-controlled page element might be endorsed, we are actually proving a *qualified robustness* condition [31] (and *qualified transparency*) which says that the attacker does not have influence over declassifications, outside of what has been endorsed (and we do not endorse what the attacker does not have privilege to see, outside of what has been declassified). This does not change our security conditions because sanitized influence (and release knowledge) already capture this, but it does give the attacker more power over what is declassified since untrusted code could be endorsed and then be permitted to influence declassification.

Alternative DOM models: In our model, each execution has its own copy of the DOM, similar to prior work [13, 17, 29]. Another option would be to have a single DOM [23, 40]. In these models, the security policy would determine which API calls would succeed and which would be replaced with a default value. Yet other work looks at the possibility of using a single DOM with SME by tracking secrets (taint) through the nodes, attributes, and event handlers [28]. It would be challenging to allow similar fine-grained declassifications of events related to dynamically generate elements in the first model, and the second model is susceptible to implicit leak through control flow decisions.

Our "DOM" is a flat structure with few APIs since the structure of the DOM did not contribute directly to the relationship between attacker influence and robust declassification. As future work, it would be interesting to have a more realistic tree-structured DOMs [28, 35] to model more complex DOM features [3, 33] to explore whether attacker influence over event bubbling order and pre-emptive event scheduling (for instance) yields new attacks.

9 CONCLUSION

We developed SME^T, an IFC monitor, which combines SME and taint tracking to prevent attackers from influencing declassification. SME^T permits the benign declassifications involving trusted dynamic features—without sacrificing security. We proved that SME^T satisfies progress-insensitive noninterference for both confidentiality and integrity using knowledge-based and influence-based security conditions, respectively. We showed that robust declassification follows from our novel influence-based security condition.

REFERENCES

[1] Amir A. Ahmadian and Musard Balliu. 2022. Dynamic Policies Revisited. In *Proceedings of the 2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P).*

[2] Maximilian Algehed, Alejandro Russo, and Cormac Flanagan. 2019. Optimising Faceted Secure Multi-Execution. In *Proceedings of the 2019 IEEE Computer Security Foundations Symposium (CSF).*

[3] Ana Gualdina Almeida Matos, José Fragoso Santos, and Tamara Rezk. 2014. An Information Flow Monitor for a Core of DOM: Introducing References and Live Primitives. In *Proceedings of the International Symposium on Trustworthy Global Computing (TGC)*.

[4] Aslan Askarov and Stephen Chong. 2012. Learning is Change in Knowledge: Knowledge-Based Security for Dynamic Policies. In *Proceedings of the 2012 IEEE Computer Security Foundations Symposium (CSF)*.

[5] Aslan Askarov and Andrew Myers. 2011. Attacker Control and Impact for Confidentiality and Integrity. *Logical Methods in Computer Science (LMCS)* 7 (2011).

[6] Aslan Askarov and Andrei Sabelfeld. 2007. Gradual Release: Unifying Declassification, Encryption and Key Release Policies. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy (SP)*.

[7] Thomas H. Austin and Cormac Flanagan. 2009. Efficient purely-dynamic information flow analysis. In *ACM Workshop on Programming Languages and Analysis for Security (PLAS)*.

[8] Thomas H. Austin and Cormac Flanagan. 2012. Multiple facets for dynamic information flow. In *Proceedings of the ACM Principles of Programming Languages (POPL)*.

[9] Musard Balliu. 2013. A logic for information flow analysis of distributed programs. In *Proceedings of the Nordic Conference on Secure IT Systems (NordSec)*.

[10] Anindya Banerjee, David A Naumann, and Stan Rosenberg. 2008. Expressive declassification policies and modular static enforcement. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy (SP)*.

[11] Lujo Bauer, Shaoying Cai, Limin Jia, Timothy Passaro, Michael Stroucken, and Yuan Tian. 2015. Run-time Monitoring and Formal Analysis of Information Flows in Chromium. In *Proceedings of the 2015 Network and Distributed System Security Symposium (NDSS '15)*.

[12] Abhishek Bichhawat, Vineet Rajani, Jinank Jain, Deepak Garg, and Christian Hammer. 2017. WebPol: Fine-grained Information Flow Policies for Web Browsers. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS '17)*.

[13] Nataliia Bielova, Dominique Devriese, Fabio Massacci, and Frank Piessens. 2011. Reactive non-interference for a browser model. In *Proceedings of the International Conference on Network and System Security (NSS)*.

[14] Nataliia Bielova and Tamara Rezk. 2016. Spot the Difference: Secure Multi-execution and Multiple Facets. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*.

[15] Aaron Bohannon and Benjamin C. Pierce. 2010. Featherweight Firefox: Formalizing the Core of a Web Browser. In *USENIX Conference on Web Application Development (WebApps 10)*. USENIX Association. https://www.usenix.org/conference/webapps-10/featherweight-firefox-formalizing-core-web-browser

[16] Aaron Bohannon, Benjamin C. Pierce, Vilhelm Sjöberg, Stephanie Weirich, and Steve Zdancewic. 2009. Reactive noninterference. In *Proceedings of the 2009 ACM Conference on Computer and Communications Security (CCS)*.

[17] Roberto Capizzi, Antonio Longo, V. N. Venkatakrishnan, and A. Prasad Sistla. 2008. Preventing Information Leaks through Shadow Executions. In *Proceedings of the 2008 Annual Computer Security Applications Conference (ACSAC)*.

[18] Ethan Cecchetti, Andrew C. Myers, and Owen Arden. 2017. Nonmalleable Information Flow Control. In *Proceedings of the 2017 ACM Conference on Computer and Communications Security (CCS)*.

[19] Stephen Chong and Andrew C. Myers. 2006. Decentralized Robustness. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW)*.

[20] Stephen Chong, K. Vikram, and Andrew C. Myers. 2007. SIF: Enforcing Confidentiality and Integrity in Web Applications. In *Proceedings of the 16th USENIX Security Symposium*. 1–16.

[21] Andrey Chudnov and David A. Naumann. 2010. Information Flow Monitor Inlining. In *Proceedings of the 2010 23rd IEEE Computer Security Foundations Symposium (CSF '10)*.

[22] Brian J. Corcoran, Nikhil Swamy, and Michael Hicks. 2009. Cross-tier, Label-based Security Enforcement for Web Applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 269–282.

[23] Willem De Groef, Dominique Devriese, Nick Nikiforakis, and Frank Piessens. 2012. FlowFox: a web browser with flexible and precise information flow control. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS)*.

[24] Dominique Devriese and Frank Piessens. 2010. Noninterference through Secure Multi-execution. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy (SP)*.

[25] Dongseok Jang, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. 2010. An Empirical Study of Privacy-violating Information Flows in JavaScript Web Applications. In *Proceedings of the 2010 ACM Conference on Computer and Communications Security (CCS '10)*.

[26] Limin Jia, Jassim Aljuraidan, Elli Fragkaki, Lujo Bauer, Michael Stroucken, Kazuhide Fukushima, Shinsaku Kiyomoto, and Yutaka Miyake. 2013. Run-Time Enforcement of Information-Flow Properties on Android. In *Computer Security—ESORICS 2013: 18th European Symposium on Research in Computer Security*.

[27] Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, and Robert Morris. 2007. Information flow control for standard OS abstractions. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles (SOSP)*.

[28] McKenna McCall, Abhishek Bichhawat, and Limin Jia. 2022. Compositional Information Flow Monitoring for Reactive Programs. In *Proceedings of the 2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*.

[29] McKenna McCall, Hengruo Zhang, and Limin Jia. 2018. Knowledge-based Security of Dynamic Secrets for Reactive Programs. In *Proceedings of the 31st IEEE Computer Security Foundations Symposium (CSF)*.

[30] Scott Moore and Stephen Chong. 2011. Static Analysis for Efficient Hybrid Information-Flow Control. In *Proceedings of the 24TH IEEE Computer Security Foundations Symposium (csf)*.

[31] Andrew C. Myers, Andrei Sabelfeld, and Steve Zdancewic. 2006. Enforcing Robust Declassification and Qualified Robustness. *Journal of Computer Security (JCS)* 14, 2 (2006).

[32] Minh Ngo, Nataliia Bielova, Cormac Flanagan, Tamara Rezk, Alejandro Russo, and Thomas Schmitz. 2018. A Better Facet of Dynamic Information Flow Control. In *Proceedings of The Web Conference (WWW)*.

[33] Vineet Rajani, Abhishek Bichhawat, Deepak Garg, and Christian Hammer. 2015. Information flow control for event handling and the DOM in web browsers. In *Proceedings of the 2015 IEEE Computer Security Foundations Symposium (CSF)*.

[34] Alejandro Russo and Andrei Sabelfeld. 2010. Dynamic vs. Static Flow-Sensitive Security Analysis. In *Proceedings of the 2010 23rd IEEE Computer Security Foundations Symposium*.

[35] Alejandro Russo, Andrei Sabelfeld, and Andrey Chudnov. 2009. Tracking Information Flow in Dynamic Tree Structures. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*.

[36] Thomas Schmitz, Maximilian Algehed, Cormac Flanagan, and Alejandro Russo. 2018. Faceted Secure Multi Execution. In *Proceedings of the 2018 ACM Conference on Computer and Communications Security (CCS)*.

[37] Daniel Schoepe, Musard Balliu, Benjamin C. Pierce, and Andrei Sabelfeld. 2016. Explicit Secrecy: A Policy for Taint Tracking. In *Proceedings of the 2016 IEEE 1st European Symposium on Security and Privacy (EuroS&P)*.

[38] Steven Sprecher, Christoph Kerschbaumer, and Engin Kirda. 2022. SoK: All or Nothing - A Postmortem of Solutions to the Third-Party Script Inclusion Permission Model and a Path Forward. In *Proceedings of the 2022 IEEE 7th European Symposium on Security and Privacy (EuroSP '22)*.

[39] Deian Stefan, Edward Z. Yang, Brad Karp, Petr Marchenko, Alejandro Russo, and David Mazières. 2014. Protecting Users by Confining JavaScript with COWL. In *Proceedings of the 8th Symposium on Operating Systems Design and Implementation (OSDI)*.

[40] Mathy Vanhoef, Willem De Groef, Dominique Devriese, Frank Piessens, and Tamara Rezk. 2014. Stateful declassification policies for event-driven programs. In *Proceedings of the 2014 IEEE Computer Security Foundations Symposium (CSF)*.

[41] MDN web docs. 2023. Event.isTrusted. https://developer.mozilla.org/en-US/docs/Web/API/Event/isTrusted [Online; accessed 9-January-2023].

[42] Steve Zdancewic and Andrew C Myers. 2001. Robust Declassification.. In *Proceedings of Computer Security Foundations Workshop (CSFW)*.

[43] Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières. 2006. Making information flow explicit in HiStar. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*.

A SME SYNTAX AND SEMANTICS

Confidentiality label:	ℓ_c	$\in$	$\mathcal{L}_c$
Integrity label:	ℓ_i	$\in$	$\mathcal{L}_i$
Label set:	$\mathcal{L}$	$::=$	$\mathcal{L}_c \times \mathcal{L}_i$
Program counter:	pc	$\in$	$\mathcal{L}$
Policy context:	$\mathcal{P}$	$::=$	(Γ, m_l)
Downgrade state:	ρ	$::=$	$\cdot \mid \rho, (id_1.Ev_1, n_1)$
Downgrade channels:	d	$::=$	$(\iota_1, v_1), \cdots, (\iota_n, v_n)$
Declassification function:	$\mathcal{D}$		
Endorsement function:	$\mathcal{E}$		
Declassification:	$\mathcal{R}$	$::=$	(ρ_d, d_d)
Endorsement:	$\mathcal{S}$	$::=$	(ρ_e, d_e)
Declassified/endorsed value:	r	$::=$	$\cdot \mid \mathsf{none} \mid \mathsf{some}(\iota, v)$
Event:	Ev	$::=$	$\ldots$
Event handler:	eh	$::=$	$\mathsf{on}Ev(x)\{c\}$
Expression:	e	$::=$	$x \mid id \mid \mathsf{uop}\ e \mid e_1\ \mathsf{bop}\ e_2$
Command:	c	$::=$	$\mathsf{skip} \mid c_1; c_2 \mid x := e \mid id := e \mid \mathsf{while}\ e\ \mathsf{do}\ c \mid \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \mathsf{output}\ ch\ e$
		$\mid$	$\mathsf{trigger}\ id.Ev(e) \mid \mathsf{new}(id, e) \mid \mathsf{addEh}(id, eh) \mid x := \mathsf{declassify}(\iota, e) \mid x := \mathsf{endorse}(\iota, e)$
Event handler map:	M	$::=$	$\cdot \mid M, Ev \mapsto EH$
Event handler set:	EH	$::=$	$\{\ \} \mid EH \cup \{(eh, pc)\}$
Local (variable) state:	σ^v	$::=$	$\cdot \mid \sigma^v, x \mapsto v$
Global (variable) state:	σ^g	$::=$	$\cdot \mid \sigma^g, x \mapsto v$
EH state (DOM):	σ^{EH}	$::=$	$\cdot \mid \sigma^{EH}, id \mapsto (v, M, pc)$
Single global state:	σ^G	$::=$	σ^g, σ^{EH}

Event queue:	E	$::=$	$\cdot \mid E, (id.Ev(v), pc)$
Execution state:	s	$::=$	$P \mid C$
Single configuration:	κ	$::=$	σ^v, c, s, E
Configuration stack:	ks	$::=$	$\cdot \mid (\kappa, pc_{src}, pc) :: \mathsf{ks}$
Actions:	α	$::=$	$id.Ev(v) \mid ch(v) \mid \bullet \mid \mathsf{dcl}(v) \mid \mathsf{end}(v) \mid \mathsf{new}(id, pc_{src}) \mid \mathsf{newEH}(id, eh, pc_{id}, pc_{src})$
Labeled actions:	α_l	$::=$	(α, pc)
Global state:	Σ	$::=$	$\cdot \mid pc \mapsto \sigma^G, \Sigma$
SME Configuration:	K	$::=$	$\mathcal{R}, \mathcal{S}; \Sigma; \mathsf{ks}$
SME Exec. Traces:	T	$::=$	$K \mid T \overset{\alpha_l}{\Longrightarrow} K$

Input rules. These rules process inputs and determine which event handlers to run. All of the executions who trust/have enough privilege to see the event receive the event: $E = ((id.Ev(v), pc'') \mid pc'' \in \mathcal{L}$ s.t. $pc \sqcup pc' \sqsubseteq pc'')$ Here, we consider both the source of the event (pc) and the policy (pc'). The event handler lookup semantics $\Sigma, E \rightsquigarrow \mathsf{ks}$ look up the event handlers for the event in each copy of the DOM that receives the event. First, we look up the matching page element in the DOM for each execution receiving the event. Rule LOOKUP handles the case where there is a matching page element (we use $pc, pc_{id}, v \vdash M(Ev) \rightsquigarrow \mathsf{ks}$ to set up the execution context for the resulting event handlers in $M(Ev)$), while rule LOOKUP-MISSING skips executions where no matching element exists. Rule LOOKUPEH attaches two labels to the event handler. One reflects the trustworthiness of the source of the given event handler (this is the execution context, plus the source of the page element and event handler, $pc \sqcup pc_{id} \sqcup pc_{eh}$). If this event handler were to add a new page element (or register a new event handler), this would be the label attched to that element (resp., event handler). The other label reflects the execution context (pc), which determines which copy of the DOM/other shared storage the event handler interacts with.

The input rules also perform declassification and endorsement, ensuring that the user trusts/has enough privilege to interact with the page element associated with the event. That is, we need to ensure the declassification is robust and the endorsement is transparent. Rule IN handles the case where the declassification would not be robust ($\mathsf{labelOf}(\sigma^{EH}(id)) \downarrow^i \not\sqsubseteq pc \downarrow^i$) and the endorsement would not be transparent ($\mathsf{labelOf}(\sigma^{EH}(id)) \downarrow^c \not\sqsubseteq pc \downarrow^c$). Rules IN-D and IN-E handle the case where the declassfication is robust, but endorsement would not be transparent (resp., endorsement is transparent, but declassification is not robust), and rule IN-DE handles the case where both are true.

If downgrading is deemed safe, we use downgrade to downgrade the event. downgrade uses policies $\mathcal{D}$ (for declassification) and $\mathcal{E}$ (for endorsement) to determine what (if anything) is downgraded. Note that $\mathcal{D}$ and $\mathcal{E}$ also run the downgraded event in the executions which trust (resp. have enough privilege) to see the original event. For example, suppose $\mathcal{D}$ declassifies an event at (l_c, l_i) to confidentiality label l'_c, it will also release the event to (l'_c, l'_i) for all l'_i s.t. $l_i \sqsubseteq l'_i$. That is, declassification will make the event more public, but it will also run this

$$\boxed{\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{(\alpha, pc)}{\Longrightarrow} K'}$$

$$\frac{\begin{array}{c}\mathcal{P}(id.Ev(v)) = pc' \qquad \Sigma(pc) = (\_, \sigma^{EH}) \\ \mathsf{labelOf}(\sigma^{EH}(id)) \downarrow^i \not\sqsubseteq pc \downarrow^i \qquad \mathsf{labelOf}(\sigma^{EH}(id)) \downarrow^c \not\sqsubseteq pc \downarrow^c \\ E = ((id.Ev(v), pc'') \mid pc'' \in \mathcal{L} \text{ s.t. } pc \sqcup pc' \sqsubseteq pc'') \qquad \Sigma, E \rightsquigarrow \mathsf{ks}\end{array}}{\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathcal{R}, \mathcal{S}; \Sigma; \cdot \overset{(id.Ev(v), pc)}{\Longrightarrow} \mathcal{R}, \mathcal{S}; \Sigma; \mathsf{ks}} \text{ IN}$$

$$\frac{\begin{array}{c}\alpha = (id.Ev(v), pc) \qquad \mathcal{P}(id.Ev(v)) = pc' \qquad \Sigma(pc) = (\_, \sigma^{EH}) \\ \mathsf{labelOf}(\sigma^{EH}(id)) \downarrow^i \sqsubseteq pc \downarrow^i \qquad \mathsf{labelOf}(\sigma^{EH}(id)) \downarrow^c \not\sqsubseteq pc \downarrow^c \\ E = ((id.Ev(v), pc'') \mid pc'' \in \mathcal{L} \text{ s.t. } pc \sqcup pc' \sqsubseteq pc'') \qquad \Sigma, E \rightsquigarrow \mathsf{ks} \\ \mathsf{downgrade}_{\mathcal{D}}(\mathcal{R}, \Sigma, \alpha, pc') = (\mathcal{R}', E_d) \qquad pc, \mathsf{r} \vdash \Sigma, E_d \rightsquigarrow \mathsf{ks}_d\end{array}}{\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathcal{R}, \mathcal{S}; \Sigma; \cdot \overset{\alpha}{\Longrightarrow} \mathcal{R}', \mathcal{S}; \Sigma; \mathsf{ks} :: \mathsf{ks}_d} \text{ IN-D}$$

$$\frac{\begin{array}{c}\alpha = (id.Ev(v), pc) \qquad \mathcal{P}(id.Ev(v)) = pc' \qquad \Sigma(pc) = (\_, \sigma^{EH}) \\ \mathsf{labelOf}(\sigma^{EH}(id)) \downarrow^i \not\sqsubseteq pc \downarrow^i \qquad \mathsf{labelOf}(\sigma^{EH}(id)) \downarrow^c \sqsubseteq pc \downarrow^c \\ E = ((id.Ev(v), pc'') \mid pc'' \in \mathcal{L} \text{ s.t. } pc \sqcup pc' \sqsubseteq pc'') \qquad \Sigma, E \rightsquigarrow \mathsf{ks} \\ \mathsf{downgrade}_{\mathcal{E}}(\mathcal{S}, \Sigma, \alpha, pc') = (\mathcal{S}', E_e) \qquad pc, \mathsf{t} \vdash \Sigma, E_e \rightsquigarrow \mathsf{ks}_e\end{array}}{\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathcal{R}, \mathcal{S}; \Sigma; \cdot \overset{\alpha}{\Longrightarrow} \mathcal{R}, \mathcal{S}'; \Sigma; \mathsf{ks} :: \mathsf{ks}_e} \text{ IN-E}$$

$$\frac{\begin{array}{c}\alpha = (id.Ev(v), pc) \qquad \mathcal{P}(id.Ev(v)) = pc' \qquad \Sigma(pc) = (\_, \sigma^{EH}) \\ \mathsf{labelOf}(\sigma^{EH}(id)) \downarrow^i \sqsubseteq pc \downarrow^i \qquad \mathsf{labelOf}(\sigma^{EH}(id)) \downarrow^c \sqsubseteq pc \downarrow^c \\ E = ((id.Ev(v), pc'') \mid pc'' \in \mathcal{L} \text{ s.t. } pc \sqcup pc' \sqsubseteq pc'') \qquad \Sigma, E \rightsquigarrow \mathsf{ks} \\ \mathsf{downgrade}_{\mathcal{D}}(\mathcal{R}, \Sigma, \alpha, pc') = (\mathcal{R}', E_d) \qquad pc, \mathsf{r} \vdash \Sigma, E_d \rightsquigarrow \mathsf{ks}_d \\ \mathsf{downgrade}_{\mathcal{E}}(\mathcal{S}, \Sigma, \alpha, pc') = (\mathcal{S}', E_e) \qquad pc, \mathsf{t} \vdash \Sigma, E_e \rightsquigarrow \mathsf{ks}_e \\ \mathsf{downgrade}_{\mathcal{D},\mathcal{E}}(\mathcal{R}, \mathcal{S}, \Sigma, \alpha, pc') = E_{d,e} \qquad pc, \mathsf{rt} \vdash \Sigma, E_{d,e} \rightsquigarrow \mathsf{ks}_{d,e}\end{array}}{\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathcal{R}, \mathcal{S}; \Sigma; \cdot \overset{\alpha}{\Longrightarrow} \mathcal{R}', \mathcal{S}'; \Sigma; \mathsf{ks} :: \mathsf{ks}_d :: \mathsf{ks}_e :: \mathsf{ks}_{d,e}} \text{ IN-DE}$$

Figure 16: SME Input Rules

event at all of the executions trusting the original event. This is similar to the way that the original event (before downgrading) is received by all of the executions which trust/have enough privilege to see the event.

Next, use the event handler lookup semantics $\rightsquigarrow$ to look up the event handlers for the downgraded event. These rules are similar to the lookup semantics described above, except that we use a flag to indicate whether the event is a result of declassification/endorsement/both so that we can ensure robustness/transparency. When we look up downgraded events, we need to ensure that the source of the event trusts/has enough privilege to interact with the event handler. The lookup rule LOOKUP-R handles the case where there is at least one event handler that is sufficiently trusted ($M(Ev) \downarrow^i_l$ is the set of event handlers trusted by l) to be declassified and the rule LOOKUP-NOTR handles the case where there is no such event handler ($Ev \notin M$ or $M(Ev) \downarrow^i_l = \cdot$). The rules for transparency and both robustness and transparency are similar.

We use both the policy and the source of the event to decide which events are triggered. An event runs in all of the executions as public/untrusted as both the policy and source of the event. The event can also run in executions that are more public/trusted than the source of the event (up to what is given by the policy) if declassification and/or endorsement are safe to do.

Output rules. OUT-SILENT handles all of the actions which are not communications on channels. These may be silent actions, $\bullet$, or tracking downgrades (dcl and end). Declassifications and endorsements are not outputs on channels, but we do track them as explicit actions to make proofs easier. producer(κ) is true if the execution state is producer (i.e., $\kappa = \sigma, c, P, E$) and consumer(κ) is true otherwise (i.e., the execution state is consumer, $\kappa = \sigma, c, C, E$).

We use $pc \sqcup pc_{id} \sqcup pc_{eh}$ for the source of the event handler in the $\rightsquigarrow$ lookup semantics because $pc_{id} \sqcup pc_{eh}$ captures the integrity/secrecy of the code which added id to the page (and registered eh to id), and pc captures the integrity/secrecy of the event triggering this event handler. For instance, pc_{id} would capture an element added by a third-party (i.e., anything added to the page by this element shouldn't be declassified or the third-party will have influenced the declassification) and pc would capture whether the event itself is secret (i.e., anything this event adds shouldn't be endorsed or you risk laundering secrets through the endorsement).

$$\boxed{\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{(\alpha, pc)}{\Longrightarrow} K'}$$

$$\frac{\text{producer}(\kappa) \qquad \mathcal{R} = (\rho_d, d_d) \qquad \mathcal{S} = (\rho_e, d_e) \qquad pc_{src}, d_d, d_e \vdash \Sigma, \kappa \xrightarrow{ch(v)}_{pc} \Sigma', \text{ks}' \qquad \mathcal{P}(ch) = pc}{\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathcal{R}, \mathcal{S}; \Sigma; (\kappa, pc_{src}, pc) :: \text{ks} \overset{(ch(v), pc)}{\Longrightarrow} \mathcal{R}, \mathcal{S}; \Sigma'; \text{ks}' :: \text{ks}} \text{ Out}$$

$$\frac{\text{producer}(\kappa) \qquad \mathcal{R} = (\rho_d, d_d) \qquad \mathcal{S} = (\rho_e, d_e) \qquad pc_{src}, d_d, d_e \vdash \Sigma, \kappa \xrightarrow{ch(v)}_{pc} \Sigma', \text{ks}' \qquad \mathcal{P}(ch) \neq pc}{\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathcal{R}, \mathcal{S}; \Sigma; (\kappa, pc_{src}, pc) :: \text{ks} \overset{(\bullet, pc)}{\Longrightarrow} \mathcal{R}, \mathcal{S}; \Sigma'; \text{ks}' :: \text{ks}} \text{ Out-Skip}$$

$$\frac{\text{producer}(\kappa) \qquad \mathcal{R} = (\rho_d, d_d) \qquad \mathcal{S} = (\rho_e, d_e) \qquad pc_{src}, d_d, d_e \vdash \Sigma, \kappa \xrightarrow{\alpha}_{pc} \Sigma', \text{ks}' \qquad \alpha \neq ch(v)}{\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathcal{R}, \mathcal{S}; \Sigma; (\kappa, pc_{src}, pc) :: \text{ks} \overset{(\alpha, pc)}{\Longrightarrow} \mathcal{R}, \mathcal{S}; \Sigma'; \text{ks}' :: \text{ks}} \text{ Out-Silent}$$

$$\frac{\text{consumer}(\kappa)}{\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathcal{R}, \mathcal{S}; \Sigma; (\kappa, pc_{src}, pc) :: \text{ks} \overset{(\bullet, pc)}{\Longrightarrow} \mathcal{R}, \mathcal{S}; \Sigma; \text{ks}} \text{ Out-Next}$$

Figure 17: SME Output Rules

$$\boxed{\text{downgrade}_{\mathcal{D}}(\mathcal{R}, \Sigma, (id.Ev(v), pc), pc_{Ev}) = (\mathcal{R}', E)}$$

$$\frac{\mathcal{R} = (\rho, d) \qquad E = ((id.Ev(v), (l_c, l_i)) \mid l_c \in \mathcal{L}_c \text{ s.t. } pc \downarrow^c \sqsubseteq l_c \sqsubset pc_{Ev} \downarrow^c \wedge l_i = pc_{Ev} \downarrow^i \sqcup pc \downarrow^i) \qquad \mathcal{D}((id.Ev(v), pc_{Ev}), pc, \rho) = (\rho', v_d, E_d) \qquad d' = \text{update}(d, v_d) \qquad E_r = \text{robust}(\Sigma, E :: E_d, pc_{Ev})}{\text{downgrade}_{\mathcal{D}}(\mathcal{R}, \Sigma, (id.Ev(v), pc_{Ev}), pc) = ((\rho', d'), E_r)} \text{ downgrade}$$

$$\frac{\Sigma(pc) = (\_, \sigma^{EH}) \qquad \text{labelOf}(\sigma^{EH}(id)) \downarrow^i \sqsubseteq pc_{Ev} \downarrow^i}{\text{robust}(\Sigma, ((id.Ev(v), pc) :: E), pc_{Ev}) = (id.Ev(v), pc) :: \text{robust}(\Sigma, E, pc_{Ev})} \text{ robust}$$

$$\frac{\Sigma(pc) = (\_, \sigma^{EH}) \qquad id \notin \sigma^{EH} \qquad \text{or} \qquad \text{labelOf}(\sigma^{EH}(id)) \downarrow^i \not\sqsubseteq pc_{Ev} \downarrow^i}{\text{robust}(\Sigma, ((id.Ev(v), pc) :: E), pc_{Ev}) = \text{robust}(\Sigma, E, pc_{Ev})} \text{ not-robust} \qquad \frac{}{\text{robust}(\Sigma, \cdot, pc_{Ev}) = \cdot} \text{ robust-emp}$$

Figure 18: Downgrade (declassification only) Semantics

We could update the label on the page elemetn for UPDATE, but we don't in favor of simpler semantics and to avoid implicit leaks. For example, if an attacker modifies a trusted page element, updating the label would prevent declassifications associated with that element–meaning the attacker has some (implicit) control over whether a declassification happens. When adding a new element to the DOM or registering a new event handler, we label it as pc_{src} instead of pc because pc_{src} captures the secrecy/integrity of the triggering event as well as the source of the event handler code. If we used pc instead, our monitor would be too rigid. For instance, the code in an untrusted execution may have been added by a trusted *or* untrusted party–but using only the label on the execution, we would treat all code in the untrusted execution as untrusted.

B ADDITONAL SYNTAX/TERMINOLOGY

$\approx_l^p$ represents equivalence for property p for attackers at level $l \in \mathcal{L}_p$. p may be c (confidentiality) or i (integrity).

$$\boxed{\mathsf{downgrade}_{\mathcal{E}}(\mathcal{S}, \Sigma, (id.Ev(v), pc), pc_{Ev}) = (\mathcal{S}', E)}$$

$$\frac{\begin{array}{c}\mathcal{S} = (\rho, d) \\ E = ((id.Ev(v), (l_c, l_i)) \mid l_i \in \mathcal{L}_i \text{ s.t. } pc \downarrow^i \sqsubseteq l_i \sqsubset pc_{Ev} \downarrow^i \wedge l_c = pc_{Ev} \downarrow^c \sqcup pc \downarrow^c) \qquad \mathcal{E}((id.Ev(v), pc_{Ev}), pc, \rho) = (\rho', v_e, E_s) \\ d' = \mathsf{update}(d, v_e) \qquad \mathcal{S}' = \mathcal{S}[pc_{Ev} \mapsto (\rho', d')] \qquad E_t = \mathsf{transparent}(\Sigma, E :: E_s, pc_{Ev})\end{array}}{\mathsf{downgrade}_{\mathcal{E}}(\mathcal{S}, \Sigma, (id.Ev(v), pc_{Ev}), pc) = ((\rho', d'), E_t)} \text{ DOWNGRADE}$$

$$\frac{\Sigma(pc) = (\_, \sigma^{EH}) \qquad \mathsf{labelOf}(\sigma^{EH}(id)) \downarrow^c \sqsubseteq pc_{Ev} \downarrow^c}{\begin{array}{c}\mathsf{transparent}(\Sigma, ((id.Ev(v), pc) :: E), pc_{Ev}) = \\ (id.Ev(v), pc) :: \mathsf{transparent}(\Sigma, E, pc_{Ev})\end{array}} \text{ TRANSPARENT}$$

$$\frac{\Sigma(pc) = (\_, \sigma^{EH}) \qquad id \notin \sigma^{EH} \qquad \text{or} \qquad \mathsf{labelOf}(\sigma^{EH}(id)) \downarrow^c \not\sqsubseteq pc_{Ev} \downarrow^c}{\begin{array}{c}\mathsf{transparent}(\Sigma, ((id.Ev(v), pc) :: E), pc_{Ev}) = \\ \mathsf{transparent}(\Sigma, E, pc_{Ev})\end{array}} \text{ NOT-TRANSPARENT}$$

$$\frac{}{\mathsf{transparent}(\Sigma, \cdot, pc_{Ev}) = \cdot} \text{ TRANSPARENT-EMP}$$

Figure 19: Downgrade (endorsement only) Semantics

$$\frac{\begin{array}{c}\mathcal{R} = (\rho_d, d_d) \qquad \mathcal{S} = (\rho_e, d_e) \qquad E_c = ((id.Ev(v), (l_c, l_i)) \mid l_c \in \mathcal{L}_c \text{ s.t. } pc \downarrow^c \sqsubseteq l_c \sqsubset pc_{Ev} \downarrow^c \wedge l_i = pc_{Ev} \downarrow^i \sqcup pc \downarrow^i) \\ E_i = ((id.Ev(v), (l_c, l_i)) \mid l_i \in \mathcal{L}_i \text{ s.t. } pc \downarrow^i \sqsubseteq l_i \sqsubset pc_{Ev} \downarrow^i \wedge l_c = pc_{Ev} \downarrow^c \sqcup pc \downarrow^c) \\ \mathcal{D}((id.Ev(v), pc_{Ev}), pc, \rho_d) = (\rho_d', v_d, E_d) \qquad \mathcal{E}((id.Ev(v), pc_{Ev}), pc, \rho_e) = (\rho_e', v_e, E_e) \\ E = \mathsf{mergeEvents}(E_c :: E_d, E_i :: E_e) \\ E' = \mathsf{robustTransparent}(\Sigma, E, pc_{Ev})\end{array}}{\mathsf{downgrade}_{\mathcal{D},\mathcal{E}}(\mathcal{R}, \mathcal{S}, \Sigma, (id.Ev(v), pc_{Ev}), pc) = E'} \text{ DOWNGRADE}$$

$$\frac{E = ((id.Ev(v), (pc \downarrow^c, pc' \downarrow^i)) \mid (id.Ev(v), pc') \in E_e)}{\begin{array}{c}\mathsf{mergeEvents}((id.Ev(v), pc) :: E_d, E_e) = \\ E :: \mathsf{mergeEvents}(E_d, E_e)\end{array}} \text{ MERGEEV-SAME} \qquad \frac{\nexists (id.Ev(v), \_) \in E_e}{\begin{array}{c}\mathsf{mergeEvents}((id.Ev(v), pc) :: E_d, E_e) = \\ \mathsf{mergeEvents}(E_d, E_e)\end{array}} \text{ MERGEEV-DIFF}$$

$$\frac{}{\mathsf{mergeEvents}(\cdot, E_e) = E_e} \text{ MERGEEV-EMP}$$

Figure 20: Downgrade (declassification and endorsement) Semantics

$$\frac{\Sigma(pc) = (\_, \sigma^{EH}) \qquad \mathsf{labelOf}(\sigma^{EH}(id)) \downarrow^i \sqsubseteq pc_{Ev} \downarrow^i \qquad \mathsf{labelOf}(\sigma^{EH}(id)) \downarrow^c \sqsubseteq pc_{Ev} \downarrow^c}{\begin{array}{c}\mathsf{robustTransparent}(\Sigma, ((id.Ev(v), pc) :: E), pc_{Ev}) = \\ (id.Ev(v), pc) :: \mathsf{robustTransparent}(\Sigma, E, pc_{Ev})\end{array}} \text{ ROBUSTTRANSPARENT}$$

$$\frac{\begin{array}{c}\Sigma(pc) = (\_, \sigma^{EH}) \\ id \notin \sigma^{EH} \qquad \text{or} \qquad \mathsf{labelOf}(\sigma^{EH}(id)) \downarrow^i \not\sqsubseteq pc_{Ev} \downarrow^i \qquad \text{or} \qquad \mathsf{labelOf}(\sigma^{EH}(id)) \downarrow^c \not\sqsubseteq pc_{Ev} \downarrow^c\end{array}}{\begin{array}{c}\mathsf{robustTransparent}(\Sigma, ((id.Ev(v), pc) :: E), pc_{Ev}) = \\ \mathsf{robustTransparent}(\Sigma, E, pc_{Ev})\end{array}} \text{ NOT-ROBUSTTRANSPARENT}$$

$$\frac{}{\mathsf{robustTransparent}(\Sigma, \cdot, pc_{Ev}) = \cdot} \text{ ROBUSTTRANSPARENT-EMP}$$

$$\boxed{\Sigma, E \rightsquigarrow \mathsf{ks}}$$

$$\frac{\begin{array}{c}\Sigma(pc) = (\_, \sigma^{EH}) \qquad \sigma^{EH}(id) = (\_, M, pc_{id}) \\ pc, pc_{id}, v \vdash M(Ev) \rightsquigarrow \mathsf{ks} \qquad \Sigma, E \rightsquigarrow \mathsf{ks}'\end{array}}{\Sigma, (id.Ev(v), pc) :: E \rightsquigarrow \mathsf{ks} :: \mathsf{ks}'}\ \textsc{lookup}$$

$$\frac{\begin{array}{c}\Sigma(pc) = (\_, \sigma^{EH}) \\ id \notin \sigma^{EH} \qquad \text{or} \qquad \sigma^{EH}(id) = (\_, M, \_) \wedge Ev \notin M \\ \Sigma, E \rightsquigarrow \mathsf{ks}\end{array}}{\Sigma, (id.Ev(v), pc) :: E \rightsquigarrow \mathsf{ks}}\ \textsc{lookup-missing}$$

$$\frac{}{\Sigma, \cdot \rightsquigarrow \cdot}\ \textsc{lookup-empty}$$

$$\frac{\mathsf{ks} = ((\cdot, eh(v), P, \cdot), pc_{id} \sqcup pc_{eh}, pc) \qquad pc, pc_{id} \vdash EH \rightsquigarrow \mathsf{ks}'}{pc, pc_{id}, v \vdash \{(eh, pc_{eh})\} \cup EH \rightsquigarrow \mathsf{ks} :: \mathsf{ks}'}\ \textsc{lookupEH}$$

$$\frac{}{pc, pc_{id}, v \vdash \cdot \rightsquigarrow \cdot}\ \textsc{lookupEH-emp}$$

Figure 21: Event Handler Lookup Rules

$$\boxed{pc, \mathsf{f} \vdash \Sigma, E \rightsquigarrow \mathsf{ks}}$$

$$\frac{\begin{array}{c}\Sigma(pc) = \_, \sigma^{EH} \qquad \sigma^{EH}(id) = (\_, M, pc_{id}) \\ l = pc_{Ev} \downarrow^{i} \qquad M(Ev) \downarrow^{i}_{l} = EH \neq \cdot \qquad pc, pc_{id}, v \vdash EH \rightsquigarrow \mathsf{ks} \qquad pc_{Ev}, \mathsf{r} \vdash \Sigma, E \rightsquigarrow \mathsf{ks}'\end{array}}{pc_{Ev}, \mathsf{r} \vdash \Sigma, (id.Ev(v), pc) :: E \rightsquigarrow \mathsf{ks} :: \mathsf{ks}'}\ \textsc{lookup-R}$$

$$\frac{\begin{array}{c}\Sigma(pc) = \_, \sigma^{EH} \qquad \sigma^{EH}(id) = (\_, M, pc_{id}) \\ l = pc_{Ev} \downarrow^{i} \qquad Ev \notin M \text{ or } M(Ev) \downarrow^{i}_{l} = \cdot \\ pc_{Ev}, \mathsf{r} \vdash \Sigma, E \rightsquigarrow \mathsf{ks}\end{array}}{pc_{Ev}, \mathsf{r} \vdash \Sigma, (id.Ev(v), pc) :: E \rightsquigarrow \mathsf{ks}}\ \textsc{lookup-notR}$$

$$\frac{}{pc_{Ev}, \mathsf{r} \vdash \Sigma, \cdot \rightsquigarrow \cdot}\ \textsc{lookup-R-emp}$$

$$\frac{\begin{array}{c}\Sigma(pc) = \_, \sigma^{EH} \qquad \sigma^{EH}(id) = (\_, M, pc_{id}) \\ l = pc_{Ev} \downarrow^{c} \qquad M(Ev) \downarrow^{c}_{l} = EH \neq \cdot \qquad pc, pc_{id}, v \vdash EH \rightsquigarrow \mathsf{ks} \qquad pc_{Ev}, \mathsf{t} \vdash \Sigma, E \rightsquigarrow \mathsf{ks}'\end{array}}{pc_{Ev}, \mathsf{t} \vdash \Sigma, (id.Ev(v), pc) :: E \rightsquigarrow \mathsf{ks} :: \mathsf{ks}'}\ \textsc{lookup-T}$$

$$\frac{\begin{array}{c}\Sigma(pc) = \_, \sigma^{EH} \qquad \sigma^{EH}(id) = (\_, M, pc_{id}) \\ l = pc_{Ev} \downarrow^{c} \qquad Ev \notin M \text{ or } M(Ev) \downarrow^{c}_{l} = \cdot \\ pc_{Ev}, \mathsf{t} \vdash \Sigma, E \rightsquigarrow \mathsf{ks}\end{array}}{pc_{Ev}, \mathsf{t} \vdash \Sigma, (id.Ev(v), pc) :: E \rightsquigarrow \mathsf{ks}}\ \textsc{lookup-notT}$$

$$\frac{}{pc_{Ev}, \mathsf{t} \vdash \Sigma, \cdot \rightsquigarrow \cdot}\ \textsc{lookup-T-emp}$$

$$\frac{\begin{array}{c}\Sigma(pc) = \_, \sigma^{EH} \qquad \sigma^{EH}(id) = (\_, M, pc_{id}) \\ l_i = pc_{Ev} \downarrow^{i} \qquad l_c = pc_{Ev} \downarrow^{c} \\ M(Ev) \downarrow_{(l_c, l_i)} = EH \neq \cdot \\ pc, pc_{id}, v \vdash EH \rightsquigarrow \mathsf{ks} \qquad pc_{Ev}, \mathsf{rt} \vdash \Sigma, E \rightsquigarrow \mathsf{ks}'\end{array}}{pc_{Ev}, \mathsf{rt} \vdash \Sigma, (id.Ev(v), pc) :: E \rightsquigarrow \mathsf{ks} :: \mathsf{ks}'}\ \textsc{lookup-RT}$$

$$\frac{\begin{array}{c}\Sigma(pc) = \_, \sigma^{EH} \qquad \sigma^{EH}(id) = (\_, M, pc_{id}) \\ l_i = pc_{Ev} \downarrow^{i} \qquad l_c = pc_{Ev} \downarrow^{c} \qquad Ev \notin M \text{ or } M(Ev) \downarrow_{(l_c, l_i)} = \cdot \qquad pc_{Ev}, \mathsf{rt} \vdash \Sigma, E \rightsquigarrow \mathsf{ks}\end{array}}{pc_{Ev}, \mathsf{rt} \vdash \Sigma, (id.Ev(v), pc) :: E \rightsquigarrow \mathsf{ks}}\ \textsc{lookup-notRT}$$

$$\frac{}{pc_{Ev}, \mathsf{rt} \vdash \Sigma, \cdot \rightsquigarrow \cdot}\ \textsc{lookup-RT-emp}$$

$$\boxed{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, c \xrightarrow{\alpha}_{pc} \Sigma', \sigma', c', E}$$

$$\frac{}{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, \mathsf{skip}; c \xrightarrow{\bullet}_{pc} \Sigma, \sigma, c, \cdot}\ \textsc{skip} \qquad \frac{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, c_1 \xrightarrow{\alpha}_{pc} \Sigma', \sigma', c_1', E}{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, c_1; c_2 \xrightarrow{\alpha}_{pc} \Sigma', \sigma', c_1'; c_2, E}\ \textsc{seq}$$

$$\frac{[\![e]\!]^{pc}_{\sigma,\Sigma} = v \qquad \Sigma(pc) = (\sigma^g, \_) \qquad x \notin \sigma^g}{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, x := e \xrightarrow{\bullet}_{pc} \Sigma, \sigma[x \mapsto v], \mathsf{skip}, \cdot}\ \textsc{assign-l}$$

$$\frac{[\![e]\!]^{pc}_{\sigma,\Sigma} = v \qquad \Sigma(pc) = (\sigma^g, \sigma^{EH}) \qquad x \in \sigma^g \qquad \sigma^{g'} = \sigma^g[x \mapsto v] \qquad \Sigma' = \Sigma[pc \mapsto (\sigma^{g'}, \sigma^{EH})]}{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, x := e \xrightarrow{\bullet}_{pc} \Sigma', \sigma, \mathsf{skip}, \cdot}\ \textsc{assign-g}$$

$$\frac{[\![e]\!]^{pc}_{\sigma,\Sigma} = v \qquad \Sigma(pc) = (\sigma^g, \sigma^{EH}) \qquad \sigma^{EH}(id) = (\_, M, pc_{id}) \qquad \sigma^{EH'} = \sigma^{EH}[id \mapsto (v, M, pc_{id})] \qquad \Sigma' = \Sigma[pc \mapsto (\sigma^g, \sigma^{EH'})]}{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, id := e \xrightarrow{\bullet}_{pc} \Sigma', \sigma, \mathsf{skip}, \cdot}\ \textsc{update}$$

$$\frac{[\![e]\!]^{pc}_{\sigma,\Sigma} = \mathsf{true}}{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \xrightarrow{\bullet}_{pc} \Sigma, \sigma, c_1, \cdot}\ \textsc{if-true} \qquad \frac{[\![e]\!]^{pc}_{\sigma,\Sigma} = \mathsf{false}}{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \xrightarrow{\bullet}_{pc} \Sigma, \sigma, c_2, \cdot}\ \textsc{if-false}$$

$$\frac{[\![e]\!]^{pc}_{\sigma,\Sigma} = \mathsf{true}}{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, \mathsf{while}\ e\ \mathsf{do}\ \ c \xrightarrow{\bullet}_{pc} \Sigma, \sigma, c; \mathsf{while}\ e\ \mathsf{do}\ \ c\ , \cdot}\ \textsc{while-true}$$

$$\frac{[\![e]\!]^{pc}_{\sigma,\Sigma} = \mathsf{false}}{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, \mathsf{while}\ e\ \mathsf{do}\ \ c \xrightarrow{\bullet}_{pc} \Sigma, \sigma, \mathsf{skip}, \cdot}\ \textsc{while-false}$$

Figure 22: Operational Semantics for Local Evaluation

τ is a sequence of actions visible at some security level. This includes standard actions α, declassifications and endorsements, and the creation of a new element/event handler which is capable of robust declassification or transparent endorsement.

$$\begin{array}{llll}
\textit{Sequence of actions:} & \tau & ::= & \cdot \mid \tau :: \alpha \mid \tau_{\mathsf{in}} \mid \tau_{\mathsf{rls}} \mid \tau_{\mathsf{sntz}} \mid \tau_{\mathsf{down}} \mid \tau_{\mathsf{nm}} \\
\textit{Input actions:} & \tau_{\mathsf{in}} & ::= & \cdot \mid (id.Ev(v), pc) \\
\textit{Release actions:} & \tau_{\mathsf{rls}} & ::= & \tau_{\mathsf{in}} \mid \mathsf{rls}(id.Ev(v), \rho, v, E, pc) \\
\textit{Sanitize actions:} & \tau_{\mathsf{sntz}} & ::= & \tau_{\mathsf{in}} \mid \mathsf{sntz}(id.Ev(v), \rho, v, E, pc) \\
\textit{Downgraded actions:} & \tau_{\mathsf{down}} & ::= & \mathsf{down}(id.Ev(v), \tau_{\mathsf{rls}}, \tau_{\mathsf{sntz}}, E, pc) \\
\textit{Nonmalleable actions:} & \tau_{\mathsf{nm}} & ::= & \mathsf{r}(id, pc) \mid \mathsf{t}(id, pc) \mid \mid \mathsf{r}(id, eh, pc) \mid \mathsf{t}(id, eh, pc)
\end{array}$$

C CONFIDENTIALITY

Secret inputs should not influence public outputs. A system which ensures that information does not flow down the confidentiality lattice is secure. Equivalent traces T and T', written $T \approx^c_l T'$, have the same l-visible events for $l \in \mathcal{L}_c$.

Knowledge. An l-observer's knowledge is what they believe the inputs might have been after observing the l-visible outputs of a trace.

$$\mathcal{K}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l) = \{\tau \mid \exists T' \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}), T \approx^c_l T' \wedge \tau = \mathsf{in}(T')\}$$

$\mathsf{in}(T)$ is the sequence of input events provided to the system resulting in trace T, which includes both user interactions with the system ($id.Ev(v)$) and dynamically-generated page elements ($\mathsf{new}(id, pc_{src})$). We consider dynamically-generated page inputs in order to model an active attacker, who may control some of the code running on the webpage.

Confidentiality Security. An attacker at $l \in \mathcal{L}_c$ should not be able to refine their knowledge of the secret inputs by watching the system run.

$$\mathcal{K}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l) \subseteq_{\preceq} \mathcal{K}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \Longrightarrow K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$$

$$\boxed{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, c \xrightarrow{\alpha}_{pc} \Sigma', \sigma', c', E}$$

$$\frac{[\![e]\!]^{pc}_{\sigma,\Sigma} = v}{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, \mathsf{output}\ ch\ e \xrightarrow{ch(v)}_{pc} \Sigma, \sigma, \mathsf{skip}, \cdot} \text{ OUTPUT}$$

$$\frac{[\![e]\!]^{pc}_{\sigma,\Sigma} = v}{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, \mathsf{trigger}\ id.Ev(e) \xrightarrow{\bullet}_{pc} \Sigma, \sigma, \mathsf{skip}, (id.Ev(v), pc)} \text{ EVENT-TRIGGER}$$

$$\frac{\begin{array}{c}[\![e]\!]^{pc}_{\sigma,\Sigma} = v \\ \Sigma(pc) = (\sigma^g, \sigma^{EH}) \qquad id \notin \sigma^{EH} \qquad \sigma^{EH'} = \sigma^{EH}[id \mapsto (v, \cdot, pc_{src})] \qquad \Sigma' = \Sigma[pc \mapsto (\sigma^g, \sigma^{EH'})] \qquad \alpha = \mathsf{new}(id, pc_{src})\end{array}}{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, \mathsf{new}(id, e) \xrightarrow{\alpha}_{pc} \Sigma', \sigma, \mathsf{skip}, \cdot} \text{ NEW}$$

$$\frac{\begin{array}{c}\Sigma(pc) = (\sigma^g, \sigma^{EH}) \qquad eh = \mathsf{on}Ev(x)\{c\} \\ \sigma^{EH}(id) = (v, M, pc_{id}) \qquad M(Ev) = EH_{Ev} \\ M' = M[Ev \mapsto EH_{Ev} \cup \{(eh, pc_{src})\}] \\ \sigma^{EH'} = \sigma^{EH}[id \mapsto (v, M', pc_{id})] \qquad \Sigma' = \Sigma[pc \mapsto (\sigma^g, \sigma^{EH'})] \qquad \alpha = \mathsf{newEH}(id, eh, pc_{id}, pc_{src})\end{array}}{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, \mathsf{addEh}(id, eh) \xrightarrow{\alpha}_{pc} \Sigma', \sigma, \mathsf{skip}, \cdot} \text{ ADD-EH}$$

$$\frac{\mathsf{read}(d_d, \iota) = v}{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, x := \mathsf{declassify}(\iota, e) \xrightarrow{\bullet}_{pc} \Sigma, x := v, \mathsf{skip}, \cdot} \text{ DECLASSIFY}$$

$$\frac{\mathsf{read}(d_e, \iota) = v}{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, x := \mathsf{endorse}(\iota, e) \xrightarrow{\bullet}_{pc} \Sigma, x := v, \mathsf{skip}, \cdot} \text{ ENDORSE}$$

Figure 23: Operational Semantics for Local Evaluation Continued

$$\boxed{pc_{src}, d_d, d_e \vdash \Sigma, \kappa \xrightarrow{\alpha}_{pc} \Sigma', \mathsf{ks}}$$

$$\frac{}{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, \mathsf{skip}, P, \cdot \xrightarrow{\bullet}_{pc} \Sigma, ((\sigma, \mathsf{skip}, C, \cdot), pc_{src}, pc)} \text{ PtoC}$$

$$\frac{E = (id.Ev(v), pc) :: E' \qquad \Sigma, E \rightsquigarrow \mathsf{ks}}{\begin{array}{c}pc_{src}, d_d, d_e \vdash \Sigma, \sigma, \mathsf{skip}, P, E \\ \xrightarrow{\bullet}_{pc} \Sigma, ((\sigma, \mathsf{skip}, C, \cdot), pc_{src}, pc) :: \mathsf{ks}\end{array}} \text{ PtoLC}$$

$$\frac{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, c \xrightarrow{\alpha}_{pc} \Sigma', \sigma', c', E'}{pc_{src}, d_d, d_e \vdash \Sigma, \sigma, c, P, E \xrightarrow{(\alpha, pc)}_{pc} \Sigma', ((\sigma', c', P, (E, E')), pc_{src}, pc)} \text{ P}$$

Figure 24: Operational Semantic Rules for Single Execution

Progress-Insensitive (PI) Knowledge. At attacker at $l \in \mathcal{L}_c$ should not be able to refine their knowledge of the secret inputs, besides what is leaked by observing that the system makes progress.

$$\mathcal{K}_p(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l) = \{\tau_i \mid \exists T' \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}), T \approx^c_l T' \wedge \tau_i = \mathsf{in}(T') \wedge \mathsf{prog}(T')\}$$

$\mathsf{prog}(T)$ holds if the trace T eventually returns to the consumer state to process another user event.

$$\mathsf{prog}(T) \text{ iff } T = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_0 \Longrightarrow^* K \wedge \exists K_C \text{ s.t. } \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{\tau}{\Longrightarrow}{}^* K_C \wedge \mathsf{consumer}(K_C) \wedge \forall \alpha_l \in \tau, \mathsf{output}(\alpha_l)$$

$\mathsf{consumer}(K)$ holds if there are no pending event handlers in the event handler queue (i.e., $K = \mathcal{R}, \mathcal{S}; \Sigma; \cdot$)

PI Release Knowledge.

$$\begin{array}{c}\mathcal{K}_{rp}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, \alpha_l, l) = \{\tau_i \mid \exists T' \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}), T \approx^c_l T' \wedge \tau_i = \mathsf{in}(T') \wedge \\ \mathsf{prog}(T') \wedge \tau_r = (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \overset{\alpha_l}{\Longrightarrow} K)) \downarrow^c_l \wedge \mathsf{releaseT}(T', \tau_r, l))\}\end{array}$$

where release(T, τ, l) holds if the trace T will eventually produce the same release event(s), τ, at level l. Note: we use τ here because a downgraded event may appear different to different security levels, so a downgraded event may result in multiple events.

$$\begin{aligned}
&\mathsf{releaseT}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_0 \Longrightarrow^* K, \tau, l) \text{ iff } \exists T, K_C, K' \text{ s.t., } \mathsf{consumer}(K_C) \wedge \\
&\quad T = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \Longrightarrow^* K_C \Longrightarrow K' \text{ with } (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \Longrightarrow^* K_C) \downarrow_l^c = \cdot \\
&\quad \text{and...} \\
&\quad\ T \downarrow_l^c = \begin{cases} \tau & \text{when } \tau = \mathsf{rls}(_) \\ \mathsf{down}(id.Ev(v), \tau_{\mathsf{rls}}, _, E, pc) & \text{when } \tau = \mathsf{down}(id.Ev(v), \tau_{\mathsf{rls}}, \tau_{\mathsf{sntz}}, E, pc) \end{cases}
\end{aligned}$$

PI Transparent Knowledge. Secure downgrading involves both confidentiality and integrity. In order to securely endorse an event, the source should have enough privilege to see the event. When we define equivalent traces, we need to also consdier the confidentiality level of the source of events (even those in executions that the attacker does not have enough privilege to see). We define transparent knowledge to measure the amount of information leaked to an attacker when a transparent endorsement originates in an exeuction they don't have privilege to see (i.e., they learn that there is an element on that copy of the page which the principal generating the event has enough privilege to see).

$$\begin{aligned}
\mathcal{K}_{tp}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, \alpha_l, l) = \{\tau_i \mid \exists T' \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}), T \approx_l^c T' \wedge \tau_i = \mathsf{in}(T') \wedge \\
\mathsf{prog}(T') \wedge \tau = (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \stackrel{\alpha_l}{\Longrightarrow} K)) \downarrow_l^c \wedge \mathsf{transparentT}(T', \tau, l))\}
\end{aligned}$$

where transparentT(T, τ, l) holds if the trace T will eventually produce the same elements capable of transparent endorsement, given by τ. We also need to consider the case where T does *not* produce any elements capable of transparent endorsement. In this case, T has reached a new input event and an equivalent trace should be able to get to a consumer state without producing any visible events (like t(_)). This is why input events are also consdiered transparent actions, in addition to t(_). For a similar reason, we need to consider outputs made in executions that are visible to the attacker. A single event may trigger event handlers in several executions, not all of which are visible to the attacker. If an event handler is running in an execution that is visible to the attacker (i.e., the trace is producing $ch(\_)$ or $\bullet$ events), then we know an equivalent trace running an event handler in an execution that is *not* visible to the attacker should not produce any visible events (like t(_)).

$$\begin{aligned}
&\mathsf{transparentT}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_0 \Longrightarrow^* K, \tau, l) \text{ iff } \exists T, K', \tau \text{ s.t. } T = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \stackrel{\tau}{\Longrightarrow}{}^* K' \\
&\ \text{and...} \\
&T \downarrow_l^c = \tau \qquad \text{when } \tau = \mathsf{t}(_) \\
&\mathsf{consumer}(K') \wedge T \downarrow_l^c = \cdot \qquad \text{when } \tau \in \{(id.Ev(v), _), \mathsf{sntz}(_)\} \\
&\mathsf{lowEH}(K') \wedge \forall (\alpha, pc) \in \tau', \alpha \in \{ch(_), \bullet\} \wedge pc \downarrow^c \not\sqsubseteq l \qquad \text{when } \tau \in \{ch(_), \bullet\}
\end{aligned}$$

where lowEH(K) holds if the current event handler running in K is running with $pc \downarrow^c \sqsubseteq l$

Confidentiality Security (with Declassification).

Definition 5 (Knowledge-based PINI with Transparent Endorsement). A system satisfies progress-insensitive noninterference with transparent endorsement against observers at $l \in \mathcal{L}_c$ iff given any initial global store Σ_0 and downgrade policy $\mathcal{R}, \mathcal{S}, \mathcal{P}$, it is the case that for all traces T, actions α_l, and configurations K s.t. $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \stackrel{\alpha_l}{\Longrightarrow} K) \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P})$, then, the following holds

- *If* $\mathsf{rlsA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \stackrel{\alpha_l}{\Longrightarrow} K, l)$*:*
  $\mathcal{K}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \stackrel{\alpha_l}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l) \supseteq_{\preceq} \mathcal{K}_{rp}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, \alpha_l, l)$
- *If* $\mathsf{trnsprntA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \stackrel{\alpha_l}{\Longrightarrow} K, l)$*:*
  $\mathcal{K}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \stackrel{\alpha_l}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l) \supseteq_{\preceq} \mathcal{K}_{tp}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, \alpha_l, l)$
- *Otherwise:*
  $\mathcal{K}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \stackrel{\alpha_l}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l) \supseteq_{\preceq} \mathcal{K}_p(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$

where $\mathsf{rlsA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \stackrel{\alpha_l}{\Longrightarrow} K, l)$ iff $T \downarrow_l^c \in \{\mathsf{rls}(\_), \mathsf{down}(\_)\}$ $\mathsf{trnsprntA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \Longrightarrow K, l)$ iff $T \downarrow_l^c \in \{\mathsf{t}(\_), (id.Ev(v), \_), \mathsf{sntz}(\_), ch(\_), \bullet\}$

D INTEGRITY

Untrusted inputs should not influence the trusted operations of a system. A system which ensures that information does not flow down the integrity lattice (i.e., in the direction of U to T) is secure. $\approx_l^i$ traces have the same l-trusted actions for $l \in \mathcal{L}_i$.

Influence. We can protect l-trusted components of a system by measuing the possible untrusted inputs which might have produced the given l-trusted trace (for $l \in \mathcal{L}_i$).

$$\mathcal{I}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l) = \{\tau \mid \exists T' \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}), T \approx_l^i T' \wedge \tau = \mathsf{in}(T')\}$$

$\mathsf{in}(T)$ is the sequence of input events provided to the system resulting in trace T, which includes both user interactions with the system ($id.Ev(v)$) and dynamically-generated page elements ($\mathsf{new}(id, pc_{src})$).

Integrity Security. The possible inputs (including new page elements) supplied by an untrusted attacker should not be refined as more l-trusted actions are taken by the system; if they are, it means the attacker must have influence something trusted.

$$\mathcal{I}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l) \subseteq_{\preceq} \mathcal{I}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \Longrightarrow K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$$

Progress-Insensitive (PI) Influence. If a loop condition depends on an untrustred value, untrusted parties would be in control of whether any trusted operations following the loop occur. We permit this influence, so we only consider the traces which return to consumer states (i.e., the ones which make progress).

The possible inputs supplied by an attacker at should not be refined as more l-trusted actions are taken by the system, outside of what influence they have over whether the system makes progress.

$$\mathcal{I}_p(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l) = \{\tau_i \mid \exists T' \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}), T \approx_l^i T' \wedge \tau_i = \mathsf{in}(T') \wedge \mathsf{prog}(T')\}$$

PI Sanitization Influence.

$$\begin{aligned}\mathcal{I}_{ep}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, \alpha_l, l) = \{\tau_i \mid \exists T' \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}), T \approx_l^i T' \wedge \tau_i = \mathsf{in}(T') \wedge \\ \mathsf{prog}(T') \wedge \tau_s = (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \stackrel{\alpha_l}{\Longrightarrow} K)) \downarrow_l^i, \mathsf{sanitizeT}(T', \tau_s, l))\}\end{aligned}$$

where $\mathsf{sanitize}(T, \tau, l)$ holds if the trace T will eventually produce the same endorsement(s), τ, at level l.

$$\begin{aligned}&\mathsf{sanitizeT}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_0 \Longrightarrow^* K, \tau, l) \text{ iff } \exists T, K_C, K' \text{ s.t., } \mathsf{consumer}(K_C) \wedge \\ &\quad T = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \Longrightarrow^* K_C \Longrightarrow K' \text{ with } (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \Longrightarrow^* K_C) \downarrow_l^i = \cdot \\ &\quad \text{and...} \\ &\qquad T \downarrow_l^i = \begin{cases} \tau & \text{when } \tau = \mathsf{sntz}(\_) \\ \mathsf{down}(id.Ev(v), \_, \tau_{\mathsf{sntz}}, E, pc) & \text{when } \tau = \mathsf{down}(id.Ev(v), \tau_{\mathsf{rls}}, \tau_{\mathsf{sntz}}, E, pc) \end{cases}\end{aligned}$$

PI Robust Influence. Secure downgrading involves both confidentiality and integrity. In order to securely declassify an event, the principal triggering the event should trust the source of the event handler. When we define equivalent traces, we need to also consider the integrity level of the source of events (even those in executions that the attacker has direct influence over). We define robust influence to measure the amount of influence the attacker has over robust page elements (i.e., we allow their influence to be refined by the existence of robust page elements that they must not have had influence over).

$$\begin{aligned}\mathcal{I}_{rp}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, \alpha_l, l) = \{\tau_i \mid \exists T' \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}), T \approx_l^i T' \wedge \tau_i = \mathsf{in}(T') \wedge \\ \mathsf{prog}(T') \wedge \tau = (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \stackrel{\alpha_l}{\Longrightarrow} K)) \downarrow_l^c \wedge \mathsf{robustT}(T', \tau, l))\}\end{aligned}$$

where $\mathsf{robustT}(T, \tau, l)$ holds if the trace T will eventually produce the same elements capable of robust declassification, given by τ. We also need to consider the case where T does *not* produce any elements capable of robust declassification. In this case, T has reached a new input event and an equivalent trace should be able to get to a consumer state without producing any visible events (like $\mathsf{r}(\_)$). This is why input events are also consdiered robust actions, in addition to $\mathsf{r}(\_)$. For a similar reason, we need to consider outputs made in executions that are not under the influence of the attacker. A single event may trigger event handlers in several executions, not all of which are under the attacker's influence. If an event handler is running in an execution that is not under the attacker's influence (i.e., the trace is producing $ch(\_)$ or $\bullet$ events), then we know an equivalent trace running an event handler in an execution that *is* under the attacker's influence should not produce any visible events (like $\mathsf{r}(\_)$).

$$\begin{aligned}&\mathsf{robustT}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_0 \Longrightarrow^* K, \tau, l) \text{ iff } \exists T, K', \tau \text{ s.t. } T = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \stackrel{\tau}{\Longrightarrow^*} K' \\ &\quad \text{and...} \\ &T \downarrow_l^i = \tau \qquad \text{when } \tau = \mathsf{r}(\_) \\ &\mathsf{consumer}(K') \wedge T \downarrow_l^i = \cdot \qquad \text{when } \tau \in \{(id.Ev(v), \_), \mathsf{rls}(\_)\} \\ &\mathsf{lowEH}(K') \wedge \forall (\alpha, pc) \in \tau, \alpha \in \{ch(\_), \bullet\} \wedge pc \downarrow^i \not\sqsubseteq l \qquad \text{when } \tau \in \{ch(\_), \bullet\}\end{aligned}$$

where $\mathsf{lowEH}(K)$ holds if the current event handler running in K is running with $pc \downarrow^i \sqsubseteq l$

Integrity Security (with Endorsement).

Definition 6 (Influence-based PINI with Endorsement and Robust Declassification). A system satisfies progress-insensitive noninterference with endorsement and robust declassification for behaviors at $l \in \mathcal{L}_i$ iff given any initial global store Σ_0 and downgrade policy $\mathcal{R}, \mathcal{S}, \mathcal{P}$, it is the case that for all traces T, actions α_l, and configurations K s.t. $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \stackrel{\alpha_l}{\Longrightarrow} K) \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P})$, then, the following holds

- *If* $\mathsf{sntzA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \xRightarrow{\alpha_l} K, l)$*:*
  $\mathcal{I}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \xRightarrow{\alpha_l} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l) \supseteq_{\leq} \mathcal{I}_{ep}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, \alpha_l, l)$
- *If* $\mathsf{rbstA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \xRightarrow{\alpha_l} K, l)$*:*
  $\mathcal{I}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \xRightarrow{\alpha_l} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l) \supseteq_{\leq} \mathcal{I}_{rp}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, \alpha_l, l)$
- *Otherwise:*
  $\mathcal{I}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \xRightarrow{\alpha_l} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l) \supseteq_{\leq} \mathcal{I}_{p}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$

where $\mathsf{sntzA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \xRightarrow{\alpha_l} K, l)$ iff $T \downarrow_l^i \in \{\mathsf{sntz}(\_), \mathsf{down}(\_)\}$ $\mathsf{rbstA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \Longrightarrow K, l)$ iff $T \downarrow_l^i \in \{\mathsf{r}(\_), (id.Ev(v), \_), \mathsf{rls}(\_), ch(\_), \bullet\}$

E ROBUST DECLASSIFICATION

Only trusted data should be declassified. The attacker should not be in control of what is declassified or when declassification happens.

Since declassifications should only be triggered by trusted parties, we can treat declassification as a trusted event itself. Then, we can ensure declassification is transparent (i.e., only depends on trusted values) if the attacker's influence is not refined by the declassification. Robust declassification follows from integrity security.

F TRANSPARENT ENDORSEMENT

Information should be sufficiently public to be endorsed. Attacker inputs should be chosen without knowledge of secret information. Otherwise, the attacker could affect flows of the endorser's secret information into trusted information.

Since endorsements should only involve public information, we can treat endorsement as a public event itself. Then, we can ensure endorsements are transparent (i.e., only depends on public values) if the attacker's knowledge is not refined by the endorsement. Transparent endorsement follows from confidentiality security.

G EQUIVALENCE DEFINITIONS

G.1 Operations on labels

$\boxed{pc \downarrow^p}$

$$\frac{}{(l_c, l_i) \downarrow^c = l_c} \qquad \frac{}{(l_c, l_i) \downarrow^i = l_i}$$

G.2 Configuration equivalence

$\boxed{K \approx_l^p K'}$

$$\frac{\mathcal{R}_1 = \mathcal{R}_2 \qquad \mathcal{S}_1 = \mathcal{S}_2 \qquad \Sigma_1 \approx_l^p \Sigma_2 \qquad \mathsf{ks}_1 \approx_l^p \mathsf{ks}_2}{(\mathcal{R}_1, \mathcal{S}_1, \Sigma_1, \mathsf{ks}_1) \approx_l^p (\mathcal{R}_2, \mathcal{S}_2, \Sigma_2, \mathsf{ks}_2)}$$

Store equivalence.

$\boxed{\Sigma \approx_l^p \Sigma'}$

$$\frac{\Sigma_1 \downarrow_l^p = \Sigma_2 \downarrow_l^p}{\Sigma_1 \approx_l^p \Sigma_2}$$

$\boxed{\Sigma \downarrow_l^p = \Sigma'}$

$$\frac{\Sigma = pc \mapsto \sigma_{pc}^G, \Sigma' \qquad pc \downarrow^p \sqsubseteq l}{\Sigma \downarrow_l^p = pc \mapsto \sigma_{pc}^G, \Sigma' \downarrow_l^p} \qquad \frac{\Sigma = pc \mapsto (\_, \sigma^{EH}) \qquad pc \downarrow^p \not\sqsubseteq l}{\Sigma \downarrow_l^p = pc \mapsto (\cdot, \sigma^{EH} \downarrow_l^p), \Sigma' \downarrow_l^p} \qquad \frac{}{\cdot \downarrow_l^p = \cdot}$$

$\boxed{\sigma^{EH} \downarrow_l^p = \sigma^{EH\prime}}$

$$\frac{pc \downarrow^p \sqsubseteq l}{(id \mapsto (v, M, pc), \sigma^{EH}) \downarrow_l^p = (id \mapsto (\mathsf{dv}, M \downarrow_l^p, pc)), \sigma^{EH} \downarrow_l^p} \qquad \frac{pc \downarrow^p \not\sqsubseteq l}{(id \mapsto (v, M, pc), \sigma^{EH}) \downarrow_l^p = \sigma^{EH} \downarrow_l^p} \qquad \frac{}{\cdot \downarrow_l^p = \cdot}$$

$\boxed{M \downarrow_l^p = M'}$

$$\frac{}{(Ev \mapsto EH, M) \downarrow_l^p = (Ev \mapsto EH \downarrow_l^p), M \downarrow_l^p} \qquad \frac{}{\cdot \downarrow_l^p = \cdot}$$

$\boxed{EH \downarrow^p_l = EH'}$

$$\frac{pc \downarrow^p \sqsubseteq l}{(\{(eh, pc)\} \cup EH) \downarrow^p_l = \{(eh, pc)\} \cup EH \downarrow^p_l} \qquad \frac{pc \downarrow^p \not\sqsubseteq l}{(\{(eh, pc)\} \cup EH) \downarrow^p_l = EH \downarrow^p_l} \qquad \frac{}{\cdot \downarrow^p_l = \cdot}$$

Configuration stack equivalence.

$\boxed{\mathsf{ks} \approx^p_l \mathsf{ks}'}$

$$\frac{\mathsf{ks}_1 \downarrow^p_l = \mathsf{ks}_2 \downarrow^p_l}{\mathsf{ks}_1 \approx^p_l \mathsf{ks}_2}$$

$\boxed{\mathsf{ks} \downarrow^p_l = \mathsf{ks}'}$

$$\frac{\mathsf{ks} = (\kappa, pc_{src}, pc) :: \mathsf{ks}' \qquad pc \downarrow^p \sqsubseteq l}{\mathsf{ks} \downarrow^p_l = (\kappa, pc_{src}, pc) :: \mathsf{ks}' \downarrow^p_l} \qquad \frac{\mathsf{ks} = (\kappa, pc_{src}, pc) :: \mathsf{ks}' \qquad pc \downarrow^p \not\sqsubseteq l}{\mathsf{ks} \downarrow^p_l = \mathsf{ks}' \downarrow^p_l} \qquad \frac{}{\cdot \downarrow^p_l = \cdot}$$

G.3 Trace Equivalence

$\boxed{T \approx^p_l T'}$

$$T \approx^p_l T' \text{ iff } T \downarrow^p_l = T' \downarrow^p_l$$

$$\boxed{T \downarrow_l^p = \tau}$$

$$\frac{}{\mathcal{P} \vdash K \downarrow_l^p = \cdot}\ \textsc{TP-Base}$$

$$\frac{pc \downarrow^p \sqsubseteq l \qquad \alpha \notin \{id.Ev(v), ch(v)\}}{(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{(\alpha, pc)}{\Longrightarrow} T') \downarrow_l^p = \alpha :: T' \downarrow_l^p}\ \textsc{TP-Out1}$$

$$\frac{pc \downarrow^p \sqsubseteq l \vee \mathcal{P}(ch) \downarrow^p \sqsubseteq l}{(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{(ch(v), pc)}{\Longrightarrow} T') \downarrow_l^p = ch(v) :: T' \downarrow_l^p}\ \textsc{TP-Out2}$$

$$\frac{pc \downarrow^p \not\sqsubseteq l \downarrow^p \qquad \mathcal{P}(ch) \downarrow^p \not\sqsubseteq l}{(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{(ch(v), pc)}{\Longrightarrow} T') \downarrow_l^p = T' \downarrow_l^p}\ \textsc{TP-Out-Silent}$$

$$\frac{\alpha \notin \{id.Ev(v), ch(v), \mathsf{new}(\_)\} \qquad pc \downarrow^p \not\sqsubseteq l}{(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{(\alpha, pc)}{\Longrightarrow} T') \downarrow_l^p = T' \downarrow_l^p}\ \textsc{TP-Out-Silent2}$$

$$\frac{\begin{array}{c}\alpha \in \{\mathsf{new}(id, pc_{src}), \mathsf{newEH}(id, eh, pc_{id}, pc_{src})\} \qquad pc \downarrow^c \not\sqsubseteq l \\ \tau = \mathsf{t}(id, pc) \text{ if } \alpha = \mathsf{new}(\dots) \wedge pc_{src} \downarrow^c \sqsubseteq pc \downarrow^c \\ \tau = \mathsf{t}(id, eh, pc) \text{ if } \alpha = \mathsf{newEH}(\dots) \wedge pc_{src} \downarrow^c \sqsubseteq pc \downarrow^c \wedge pc_{id} \downarrow^c \sqsubseteq pc \downarrow^c \qquad \tau = \cdot \text{ otherwise}\end{array}}{(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{(\alpha, pc)}{\Longrightarrow} T') \downarrow_l^c = \tau :: T' \downarrow_l^c}\ \textsc{TP-NewC}$$

$$\frac{\begin{array}{c}\alpha \in \{\mathsf{new}(id, pc_{src}), \mathsf{newEH}(id, eh, pc_{id}, pc_{src})\} \qquad pc \downarrow^i \not\sqsubseteq l \\ \tau = \mathsf{r}(id, pc) \text{ if } \alpha = \mathsf{new}(\dots) \wedge pc_{src} \downarrow^i \sqsubseteq pc \downarrow^i \\ \tau = \mathsf{r}(id, eh, pc) \text{ if } \alpha = \mathsf{newEH}(\dots) \wedge pc_{src} \downarrow^i \sqsubseteq pc \downarrow^i \wedge pc_{id} \downarrow^i \sqsubseteq pc \downarrow^i \qquad \tau = \cdot \text{ otherwise}\end{array}}{(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{(\alpha, pc)}{\Longrightarrow} T') \downarrow_l^i = \tau :: T' \downarrow_l^i}\ \textsc{TP-NewI}$$

$$\frac{\begin{array}{c}\mathcal{P}(id.Ev(v)) = pc' \qquad K = \_, \_; \Sigma; \_ \qquad \Sigma(pc) = (\_, \sigma^{EH}) \\ \sigma^{EH}(id) \downarrow^i \not\sqsubseteq pc \downarrow^i \qquad \sigma^{EH}(id) \downarrow^c \not\sqsubseteq pc \downarrow^c \\ \tau = \mathsf{trInput}(pc', pc, id.Ev(v), l, p)\end{array}}{(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{(id.Ev(v), pc)}{\Longrightarrow} T') \downarrow_l^p = \tau :: T' \downarrow_l^p}\ \textsc{TP-In}$$

$$\frac{\begin{array}{c}\mathcal{P}(id.Ev(v)) = pc' \qquad K = \_, \_; \Sigma; \_ \qquad \Sigma(pc) = (\_, \sigma^{EH}) \\ \sigma^{EH}(id) \downarrow^i \sqsubseteq pc \downarrow^i \qquad \sigma^{EH}(id) \downarrow^c \not\sqsubseteq pc \downarrow^c \\ \tau = \mathsf{trRobust}((\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{(id.Ev(v), pc)}{\Longrightarrow} T'), l, p)\end{array}}{(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{(id.Ev(v), pc)}{\Longrightarrow} T') \downarrow_l^p = \tau :: T' \downarrow_l^p}\ \textsc{TP-In-R}$$

$$\frac{\begin{array}{c}\mathcal{P}(id.Ev(v)) = pc' \qquad K = \_, \_; \Sigma; \_ \qquad \Sigma(pc) = (\_, \sigma^{EH}) \\ \sigma^{EH}(id) \downarrow^i \not\sqsubseteq pc \downarrow^i \qquad \sigma^{EH}(id) \downarrow^c \sqsubseteq pc \downarrow^c \\ \tau = \mathsf{trTransparent}((\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{(id.Ev(v), pc)}{\Longrightarrow} T'), l, p)\end{array}}{(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{(id.Ev(v), pc)}{\Longrightarrow} T') \downarrow_l^p = \tau :: T' \downarrow_l^p}\ \textsc{TP-In-T}$$

$$\frac{\begin{array}{c}\mathcal{P}(id.Ev(v)) = pc' \qquad K = \_, \_; \Sigma; \_ \qquad \Sigma(pc) = (\_, \sigma^{EH}) \\ \sigma^{EH}(id) \downarrow^i \sqsubseteq pc \downarrow^i \qquad \sigma^{EH}(id) \downarrow^c \sqsubseteq pc \downarrow^c \\ \tau = \mathsf{trRobustTransparent}((\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{(id.Ev(v), pc)}{\Longrightarrow} T'), l, p)\end{array}}{(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{(id.ev(v), pc)}{\Longrightarrow} T') \downarrow_l^p = \tau :: T' \downarrow_l^p}\ \textsc{TP-In-RT}$$

Figure 25: Trace projection rules

$\boxed{\mathsf{trInput}(pc, pc', id.Ev(v), l, p) = \tau}$

$$\frac{pc_{src} \downarrow^{p} \sqcup pc_{\mathcal{P}} \downarrow^{p} \sqsubseteq l}{\mathsf{trInput}(pc_{\mathcal{P}}, pc_{src}, id.Ev(v), l, p) = (id.Ev(v), pc_{src})}\ \textsc{input} \qquad \frac{pc_{src} \downarrow^{p} \sqcup pc_{\mathcal{P}} \downarrow^{p} \not\sqsubseteq l}{\mathsf{trInput}(pc', pc, id.Ev(v), l, p) = \cdot}\ \textsc{noInput}$$

$\boxed{\mathsf{trRobust}(T, l, p) = \tau}$

$$\frac{\begin{array}{c} \mathcal{P}(id.Ev(v)) = pc' \\ \mathcal{R} = (\rho, d) \qquad E = ((id.ev(v), (l_c, l_i)) \mid pc' \downarrow^{c} \sqsubseteq l_c \sqsubset pc \downarrow^{c} \wedge l_i = pc \downarrow^{i} \sqcup pc' \downarrow^{i}) \qquad \mathcal{D}((id.ev(v), pc), pc', \rho) = (\rho', v, E_d) \\ E' = \mathsf{robust}(\Sigma, E :: E_d, pc) \qquad pc, \mathsf{r} \vdash \Sigma, E' \rightsquigarrow \mathsf{ks} \\ \tau' = \mathsf{rls}(id.Ev(v), \rho', v, E' \downarrow^{c}_{l}, pc) \text{ if } \rho \neq \rho' \vee v \neq \mathsf{none} \vee \mathsf{ks} \downarrow^{c}_{l} \neq \cdot \qquad \tau' = \mathsf{trInput}(pc', pc, id.Ev(v), l, c) \text{ otherwise} \end{array}}{\mathsf{trRobust}((\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathcal{R}, \_; \Sigma; \_ \xRightarrow{(id.Ev(v), pc)} T'), l, c) = \tau'}\ \textsc{robust-C}$$

$$\frac{\begin{array}{c} \mathcal{P}(id.Ev(v)) = pc' \\ \mathcal{R} = (\rho, d) \qquad E = ((id.ev(v), (l_c, l_i)) \mid pc' \downarrow^{c} \sqsubseteq l_c \sqsubset pc \downarrow^{c} \wedge l_i = pc \downarrow^{i} \sqcup pc' \downarrow^{i}) \qquad \mathcal{D}((id.ev(v), pc), pc', \rho) = (\rho', v, E_d) \\ E' = \mathsf{robust}(\Sigma, E :: E_d, pc) \qquad pc, \mathsf{r} \vdash \Sigma, E' \rightsquigarrow \mathsf{ks} \\ \tau' = \mathsf{rls}(id.Ev(v), \rho', v, E', pc) \text{ if } pc \downarrow^{i} \sqsubseteq l \text{ and } \rho \neq \rho' \vee v \neq \mathsf{none} \vee \mathsf{ks} \neq \cdot \qquad \tau' = \mathsf{trInput}(pc', pc, id.Ev(v), l, i) \text{ otherwise} \end{array}}{\mathsf{trRobust}((\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathcal{R}, \_; \Sigma; \_ \xRightarrow{(id.Ev(v), pc)} T'), l, i) = \tau'}\ \textsc{robust-I}$$

$\boxed{\mathsf{trTransparent}(T, l, p) = \tau}$

$$\frac{\begin{array}{c} \mathcal{P}(id.Ev(v)) = pc' \\ \mathcal{S} = (\rho, d) \qquad E = ((id.Ev(v), (l_c, l_i)) \mid pc' \downarrow^{i} \sqsubseteq l_i \sqsubset pc \downarrow^{i} \wedge l_c = pc \downarrow^{c} \sqcup pc' \downarrow^{c}) \qquad \mathcal{E}((id.ev(v), pc), pc', \rho) = (\rho', v, E_s) \\ E' = \mathsf{transparent}(\Sigma, E :: E_s, pc) \qquad pc, \mathsf{t} \vdash \Sigma, E' \rightsquigarrow \mathsf{ks} \\ \tau' = \mathsf{sntz}(id.Ev(v), \rho', v, E' \downarrow^{i}_{l}, pc) \text{ if } \rho' \neq \rho \vee v \neq \mathsf{none} \vee \mathsf{ks} \downarrow^{i}_{l} \neq \cdot \qquad \tau' = \mathsf{trInput}(pc', pc, id.Ev(v), l, i) \text{ otherwise} \end{array}}{\mathsf{trTransparent}((\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \_, \mathcal{S}; \_; \Sigma; \_ \xRightarrow{(id.Ev(v), pc)} T'), l, i) = \tau'}\ \textsc{transparent-I}$$

$$\frac{\begin{array}{c} \mathcal{P}(id.Ev(v)) = pc' \\ \mathcal{S} = (\rho, d) \qquad E = ((id.ev(v), (l_c, l_i)) \mid pc' \downarrow^{i} \sqsubseteq l_i \sqsubset pc \downarrow^{i} \wedge l_c = pc \downarrow^{c} \sqcup pc' \downarrow^{c}) \qquad \mathcal{E}((id.ev(v), pc), pc', \rho) = (\rho', v, E_s) \\ E' = \mathsf{transparent}(\Sigma, E :: E_s, pc) \\ pc, \mathsf{t} \vdash \Sigma, E' \rightsquigarrow \mathsf{ks} \qquad \tau' = \mathsf{sntz}(id.Ev(v), \rho', v, E', pc) \text{ if } pc \downarrow^{c} \sqsubseteq l \text{ and } \rho' \neq \rho \vee v \neq \mathsf{none} \vee \mathsf{ks} \neq \cdot \\ \tau' = \mathsf{trInput}(pc', pc, id.Ev(v), l, c) \text{ otherwise} \end{array}}{\mathsf{trTransparent}((\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \_, \mathcal{S}; \_; \Sigma; \_ \xRightarrow{(id.Ev(v), pc)} T'), l, c) = \tau'}\ \textsc{transparent-C}$$

$\boxed{\mathsf{trDowngrade}(T, l, p) = \tau}$

$$\frac{\begin{array}{c} K = \mathcal{R}, \mathcal{S}; \Sigma; \_ \qquad \alpha_l = (id.Ev(v), pc) \qquad \mathcal{P}(id.Ev(v)) = pc' \\ \tau_{\mathsf{in}} = \mathsf{trInput}(pc', pc, id.Ev(v), l, p) \\ \tau_d = \mathsf{trRobust}((\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \xRightarrow{\alpha_l} T'), l, p) \qquad \tau_e = \mathsf{trTransparent}((\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \xRightarrow{\alpha_l} T'), l, p) \\ E_{d,e} = \mathsf{downgrade}_{\mathcal{D}, \mathcal{E}}(\mathcal{R}, \mathcal{S}, \Sigma, \alpha_l, pc') \qquad pc, \mathsf{rt} \vdash \Sigma, E_{d,e} \rightsquigarrow \mathsf{ks} \qquad \mathsf{ks} \downarrow^{p}_{l} = \cdot \\ \tau = \tau_{\mathsf{in}} \text{ if } \tau_d \neq \mathsf{rls}(\_) \wedge \tau_e \neq \mathsf{sntz}(\_) \qquad \tau = \tau_d \text{ if } \tau_d = \mathsf{rls}(\_) \wedge \tau_e \neq \mathsf{sntz}(\_) \qquad \tau = \tau_e \text{ if } \tau_e = \mathsf{sntz}(\_) \wedge \tau_d \neq \mathsf{rls}(\_) \end{array}}{\mathsf{trDowngrade}((\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \xRightarrow{\alpha_l} T'), l, p) = \tau}\ \textsc{rls-or-sntz}$$

$$\frac{\begin{array}{c} K = \mathcal{R}, \mathcal{S}; \Sigma; \_ \qquad \alpha_l = (id.Ev(v), pc) \qquad \mathcal{P}(id.Ev(v)) = pc' \\ \tau_d = \mathsf{trRobust}((\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \xRightarrow{\alpha_l} T'), l, p) \qquad \tau_e = \mathsf{trTransparent}((\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \xRightarrow{\alpha_l} T'), l, p) \\ E_{d,e} = \mathsf{downgrade}_{\mathcal{D}, \mathcal{E}}(\mathcal{R}, \mathcal{S}, \Sigma, \alpha_l, pc') \qquad pc, \mathsf{rt} \vdash \Sigma, E_{d,e} \rightsquigarrow \mathsf{ks} \qquad \mathsf{ks} \downarrow^{p}_{l} \neq \cdot \text{ or } \tau_d = \mathsf{rls}(\_) \wedge \tau_e = \mathsf{sntz}(\_) \\ \tau = \mathsf{down}(id.Ev(v), \tau_d, \tau_e, E_{d,e}, pc) \end{array}}{\mathsf{trDowngrade}((\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \xRightarrow{\alpha_l} T'), l, p) = \tau}\ \textsc{down}$$

Figure 26: Helper functions for trace projection

$\boxed{E \downarrow_l^p = E'}$

$$\frac{pc \downarrow^p \sqsubseteq l}{((id.Ev(v), pc) :: E') \downarrow_l^p = (id.Ev(v), pc) :: E' \downarrow_l^p} \qquad \frac{pc \downarrow^p \not\sqsubseteq l}{((id.Ev(v), pc) :: E') \downarrow_l^p = E' \downarrow_l^p}$$

We show the rules for trace projection in Figure 25 with helper functions in Figure 26. Our trace projection definitions for inputs are complex because we need to check whether potential declassifications (or endorsements) are robust (resp. transparent) to determine if they should be included in the observation (resp. behavior) of the trace. Note that there are two rules for each of the helper functions, one for confidentiality, one for integrity. Recall that to prove robust declassification, we want to treat all declassifications as trusted (or endorsements as public). But we also consider an artibtrary lattice, so we also need to check that the declassifications come from trusted sources (we don't care if declassifications from untrusted sources are robust). We do something similar for endorsements.

H SECURITY PROOFS

H.1 Noninterference

Theorem 7 (Soundness - Confidentiality). For any downgrade policy $\mathcal{R}, \mathcal{S}, \mathcal{P}$, SME state Σ_0, and for traces, states, and actions T, K, α_l s.t. $\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \overset{\alpha_l}{\Longrightarrow} K \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P})$, then an attacker's knowledge of events secret to l is not refined:

- *If* $\mathsf{rlsA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \overset{\alpha_l}{\Longrightarrow} K, l)$:
  $\mathcal{K}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \overset{\alpha_l}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l) \supseteq_{\leq} \mathcal{K}_{rp}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, \alpha_l, l)$
- *If* $\mathsf{trnsprntA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \overset{\alpha_l}{\Longrightarrow} K, l)$:
  $\mathcal{K}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \overset{\alpha_l}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l) \supseteq_{\leq} \mathcal{K}_{tp}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, \alpha_l, l)$
- *Otherwise:*
  $\mathcal{K}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \overset{\alpha_l}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l) \supseteq_{\leq} \mathcal{K}_{p}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$

Proof.
The proof is split between three cases depending on the action, shown below.
In each case, we want to show that
$\exists \tau' \in \mathcal{K}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \overset{\alpha_l}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$ s.t. $\tau \leq \tau'$ for τ defined below

Case I: $\mathsf{rlsA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \overset{\alpha_l}{\Longrightarrow} K, l)$
Let
(I.1) $\tau \in \mathcal{K}_{rp}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, \alpha_l, l)$
(I.2) $\tau_r = (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \overset{\alpha_l}{\Longrightarrow} K) \downarrow_l^c$
$\exists T_1, K_0, K_1, K_2$ s.t.
(I.3) $T_1 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_0 \Longrightarrow^* K_1$ and
(I.4) $\tau = \mathsf{in}(T_1)$
(I.5) $T = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_0 \Longrightarrow^* K_2$
From definition $\mathcal{K}_{rp}()$,
(I.6) $T_1 \approx_l^c T$
(I.7) $\mathsf{prog}(T_1)$
(I.8) $\mathsf{releaseT}(T_1, \tau_r, l)$

Subcase i: $\tau_r = \mathsf{rls}(\_)$
By assumption and from (I.8), $\exists K_1'$ s.t.
(i.1) $\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T_1 \Longrightarrow^* K_1'$ with
(i.2) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1') \downarrow_l^c = \tau_r$
From (I.6),
(i.3) $T \downarrow_l^c = T_1 \downarrow_l^c$
From (I.2), (i.2), (i.3), and the definition of $\approx_l^c$,
(i.4) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \overset{\alpha_l}{\Longrightarrow} K) \approx_l^c (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T_1 \Longrightarrow^* K_1')$
From (i.4) and the definition of $\mathcal{K}()$,
$\mathsf{in}(T_1) :: \mathsf{in}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1') \in \mathcal{K}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \overset{\alpha_l}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$
Let

(i.5) $\tau' = \mathsf{in}(T_1) :: \mathsf{in}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1')$
From (i.5), and (I.4),
$\tau \preceq \tau'$

Subcase ii: $\tau_r = \mathsf{down}(\tau_{\mathsf{rls}}, \_)$
From (I.3), (I.5), (I.6), and Lemma 9,
(ii.1) $K_1 \approx_l^c K_2$
By assumption and from (ii.1), (I.2), (I.7), (I.8), and Lemma 30, $\exists K_1'$ s.t.
(ii.2) $\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1'$ with
(ii.3) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow K) \approx_l^c (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1')$
From (I.6) and (ii.3),
(ii.4) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \Longrightarrow K) \approx_l^c (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T_1 \Longrightarrow^* K_1')$
From (ii.4) and the definition of $\mathcal{K}()$,
$\mathsf{in}(T_1) :: \mathsf{in}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1') \in \mathcal{K}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \xRightarrow{\alpha_l} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$
Let
(ii.5) $\tau' = \mathsf{in}(T_1) :: \mathsf{in}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1')$
From (ii.5) and (II.4),
$\tau \preceq \tau'$

Case II: $\mathsf{trnsprntA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \xRightarrow{\alpha_l} K, l)$
Let
(II.1) $\tau \in \mathcal{K}_{tp}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, \alpha_l, l)$
(II.2) $\tau_t = (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \xRightarrow{\alpha_l} K) \downarrow_l^c$
$\exists T_1, K_0, K_1, K_2$ s.t.
(II.3) $T_1 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_0 \Longrightarrow^* K_1$ and
(II.4) $\tau = \mathsf{in}(T_1)$
(II.5) $T = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_0 \Longrightarrow^* K_2$
From definition $\mathcal{K}_{tp}()$,
(II.6) $T_1 \approx_l^c T$
(II.7) $\mathsf{prog}(T_1)$
(II.8) $\mathsf{transparentT}(T_1, \tau_t, l)$

Subcase i: $\tau_t = \mathsf{t}(\_)$
By assumption and from (II.3), (II.8), $\exists K_1'$ s.t.
(i.1) $\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T_1 \Longrightarrow^* K_1'$ with
(i.2) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1') \downarrow_l^c = \tau_t$
From (II.6),
(i.3) $T \downarrow_l^c = T_1 \downarrow_l^c$
From (II.2), (i.2), (i.3), and the definition of $\approx_l^p$ for T,
(i.4) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \xRightarrow{\alpha_l} K) \approx_l^c (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T_1 \Longrightarrow^* K_1')$
From (i.4) and the definition of $\mathcal{K}()$,
$\mathsf{in}(T_1) :: \mathsf{in}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1') \in \mathcal{K}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \xRightarrow{\alpha_l} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$
Let
(i.5) $\tau' = \mathsf{in}(T_1) :: \mathsf{in}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1')$
From (i.5) and (II.4),
$\tau \preceq \tau'$

Subcase ii: $\tau_t = \{(id.Ev(v), \_), \mathsf{sntz}(\_), ch(\_), \bullet\}$
From (II.3), (II.5), (II.6), and Lemma 9,
(ii.1) $K_1 \approx_l^c K_2$
By assumption and form (II.3), (II.5), (ii.1), (II.2), (II.6)-(II.8), and Lemma 22, $\exists K_1'$ s.t.
(ii.2) $\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1'$ with
(ii.3) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow K) \approx_l^c (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1')$

From (II.6) and (ii.3),

(ii.4) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T_1 \Longrightarrow^* K_1') \approx_l^c (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \Longrightarrow K)$

From (ii.4) and the definition of $\mathcal{K}()$,

$\mathsf{in}(T_1) :: \mathsf{in}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1') \in \mathcal{K}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \stackrel{\alpha_l}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$

Let

(ii.5) $\tau' = \mathsf{in}(T_1) :: \mathsf{in}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1')$

From (ii.5) and (I.4),

$\tau \preceq \tau'$

Case III: $\neg\mathsf{rlsA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \stackrel{\alpha_l}{\Longrightarrow} K, l)$
and $\neg\mathsf{trnsprntA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \stackrel{\alpha_l}{\Longrightarrow} K, l)$

Let

(III.1) $\tau \in \mathcal{K}_p(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, \alpha_l, l)$

$\exists T_1, K_0, K_1, K_2$ s.t.

(III.2) $T_1 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_0 \Longrightarrow^* K_1$ and

(III.3) $\tau = \mathsf{in}(T_1)$

(III.4) $T = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_0 \Longrightarrow^* K_2$ and

(III.5) $\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \stackrel{\alpha_l}{\Longrightarrow} K$

From definition $\mathcal{K}_{rp}()$,

(III.6) $T_1 \approx_l^c T$

(III.7) $\mathsf{prog}(T_1)$

From (III.6),

(III.8) $T \downarrow_l^c = T_1 \downarrow_l^c$

By assumption and from the definition for rlsA and trnsprntA,

(III.9) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \Longrightarrow K) \downarrow_l^c \notin \{\mathsf{rls}(\_), \mathsf{down}(\_), \mathsf{sntz}(\_), \mathsf{t}(\_), (id.Ev(v), \_), ch(\_), \bullet\}$

From (III.9) and the definition of $\downarrow_l^c$ for T,

(III.10) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \stackrel{\alpha_l}{\Longrightarrow} K) \downarrow_l^c = \cdot$

From (III.10),

(III.11) $T \approx_l^c (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \stackrel{\alpha_l}{\Longrightarrow} K)$

From (III.4) and (III.11),

(III.12) $T_1 \approx_l^c (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \stackrel{\alpha_l}{\Longrightarrow} K)$

Let

(III.13) $\tau' = \mathsf{in}(T_1)$

From (III.12) and (III.13),

$\tau' \in \mathcal{K}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \stackrel{\alpha_l}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$

From (III.3) and (III.13),

$\tau \preceq \tau'$

□

Theorem 8 (Soundness - Integrity). For any downgrade policy $\mathcal{R}, \mathcal{S}, \mathcal{P}$, SME state Σ_0, and for traces, states, and actions T, K, α_l s.t. $\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \stackrel{\alpha_l}{\Longrightarrow} K \in \mathsf{runs}(\Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P})$, then an attacker does not have influence over trusted behaviors at l:

- *If* $\mathsf{sntzA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \stackrel{\alpha_l}{\Longrightarrow} K, l)$*:*
  $\mathcal{I}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \stackrel{\alpha_l}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l) \supseteq_{\preceq} \mathcal{I}_{ep}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, \alpha_l, l)$
- *If* $\mathsf{rbstA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \stackrel{\alpha_l}{\Longrightarrow} K, l)$*:*
  $\mathcal{I}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \stackrel{\alpha_l}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l) \supseteq_{\preceq} \mathcal{I}_{rp}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, \alpha_l, l)$
- *Otherwise:*
  $\mathcal{I}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \stackrel{\alpha_l}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l) \supseteq_{\preceq} \mathcal{I}_p(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$

Proof.
The proof is split between three cases depending on the action, shown below.
In each case, we want to show that
$\exists \tau' \in \mathcal{I}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \stackrel{\alpha_l}{\Longrightarrow} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$ s.t. $\tau \preceq \tau'$ for τ defined below

Case I: $\mathsf{sntzA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \xRightarrow{\alpha_l} K, l)$

Let

(I.1) $\tau \in \mathcal{I}_{ep}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, \alpha_l, l)$

(I.2) $\tau_s = (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \xRightarrow{\alpha_l} K) \downarrow_l^i$

$\exists T_1, K_0, K_1$ s.t.

(I.3) $T_1 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_0 \Longrightarrow^* K_1$ and

(I.4) $\tau = \mathsf{in}(T_1)$

(I.5) $T = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_0 \Longrightarrow^* K_2$

From definition $\mathcal{I}_{ep}()$,

(I.6) $T_1 \approx_l^i T$

(I.7) $\mathsf{prog}(T_1)$

(I.8) $\mathsf{sanitizeT}(T_1, \alpha', l)$

Subcase i: $\tau_s = \mathsf{sntz}(\_)$

From (I.8), $\exists K_1', \alpha_{l,1}$ s.t.

(i.1) $\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T_1 \Longrightarrow^* K_1'$ with

(i.2) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1') \downarrow_l^i = \tau_r$

From (I.6),

(i.3) $T \downarrow_l^i = T_1 \downarrow_l^i$

From (I.2), (i.2), (i.3), and the definition of $\approx_l^i$,

(i.4) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \xRightarrow{\alpha_l} K) \approx_l^i (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T_1 \Longrightarrow^* K_1')$

From (i.4) and the definition of $\mathcal{I}()$,

$\mathsf{in}(T_1) :: \mathsf{in}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1') \in \mathcal{I}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \xRightarrow{\alpha_l} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$

Let

(i.5) $\tau' = \mathsf{in}(T_1) :: \mathsf{in}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1')$

From (i.5), and (I.4),

$\tau \leq \tau'$

Subcase ii: $\tau_s = \mathsf{down}(\_)$

From (I.3), (I.5), (I.6), and Lemma 9,

(ii.1) $K_1 \approx_l^i K_2$

By assumption and from (ii.1), (I.2), (I.7), (I.8), and Lemma 30, $\exists K_1'$ s.t.

(ii.2) $\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1'$ with

(ii.3) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow K) \approx_l^i (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1')$

From (I.6) and (ii.3),

(ii.4) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \Longrightarrow K) \approx_l^i (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T_1 \Longrightarrow^* K_1')$

From (ii.4) and the definition of $\mathcal{I}()$,

$\mathsf{in}(T_1) :: \mathsf{in}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1') \in \mathcal{I}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \xRightarrow{\alpha_l} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$

Let

(ii.5) $\tau' = \mathsf{in}(T_1) :: \mathsf{in}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1')$

From (ii.5) and (I.4),

$\tau \leq \tau'$

Case II: $\mathsf{rbstA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \xRightarrow{\alpha_l} K, l)$

Let

(II.1) $\tau \in \mathcal{I}_{rp}(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, \alpha_l, l)$

(II.2) $\tau_r = (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \xRightarrow{\alpha_l} K) \downarrow_l^i$

$\exists T_1, K_0, K_1, K_2$ s.t.

(II.3) $T_1 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_0 \Longrightarrow^* K_1$ and

(II.4) $\tau = \mathsf{in}(T_1)$

(II.5) $T = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_0 \Longrightarrow^* K_2$

From definition $\mathcal{I}_{rp}()$,

(II.6) $T_1 \approx_l^i T$

(II.7) $\mathsf{prog}(T_1)$

(II.8) $\mathsf{robustT}(T_1, \tau_r, l)$

Subcase i: $\tau_r = \mathsf{r}$

By assumption and from (II.3), (II.8), $\exists K_1'$ s.t.

(i.1) $\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T_1 \Longrightarrow^* K_1'$ with

(i.2) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1') \downarrow_l^i = \tau_r$

From (II.6),

(i.3) $T \downarrow_l^i = T_1 \downarrow_l^i$

From (II.2), (i.2), (i.3), and the definition of $\approx_l^p$ for T,

(i.4) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \xRightarrow{\alpha_l} K) \approx_l^i (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T_1 \Longrightarrow^* K_1')$

From (i.4) and the definition of $\mathcal{I}()$,

$\mathsf{in}(T_1) :: \mathsf{in}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1') \in \mathcal{I}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \xRightarrow{\alpha_l} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$

Let

(i.5) $\tau' = \mathsf{in}(T_1) :: \mathsf{in}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1')$

From (i.5) and (II.4),

$\tau \preceq \tau'$

Subcase ii: $\tau_r = \{(id.Ev(v), \_), \mathsf{rls}(\_), ch(\_), \bullet\}$

From (II.3), (II.5), (II.6), and Lemma 9,

(ii.1) $K_1 \approx_l^i K_2$

By assumption and from (II.3), (II.5), (ii.1), (II.2), (II.6)-(II.8), and Lemma 22, $\exists K_1'$ s.t.

(ii.2) $\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1'$ with

(ii.3) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow K) \approx_l^i (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1')$

From (II.6) and (ii.3),

(ii.4) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T_1 \Longrightarrow^* K_1') \approx_l^i (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \xRightarrow{K})$

From (ii.4) and the definition of $\mathcal{I}()$,

(ii.5) $\mathsf{in}(T_1) :: \mathsf{in}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1') \in \mathcal{K}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \xRightarrow{\alpha_l} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$

From (ii.5),

$\mathsf{in}(T_1) :: \mathsf{in}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1') \in \mathcal{I}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \xRightarrow{\alpha_l} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$

Let

(ii.6) $\tau' = \mathsf{in}(T_1) :: \mathsf{in}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1')$

From (ii.6) and (II.4),

$\tau \preceq \tau'$

Case III: $\neg\mathsf{sntzA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \xRightarrow{\alpha_l} K, l)$
and $\neg\mathsf{rbstA}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \xRightarrow{\alpha_l} K, l)$

Let

(III.1) $\tau \in \mathcal{I}_p(T, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, \alpha_l, l)$

$\exists T_1, K_0, K_1, K_2$ s.t.

(III.2) $T_1 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_0 \Longrightarrow^* K_1$ and

(III.3) $\tau = \mathsf{in}(T_1)$

(III.4) $T = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_0 \Longrightarrow^* K_2$ and

(III.5) $\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \xRightarrow{\alpha_l} K$

From definition $\mathcal{I}_{ep}()$,

(III.6) $T_1 \approx_l^i T$

(III.7) $\mathsf{prog}(T_1)$

From (III.6),

(III.8) $T \downarrow_l^i = T_1 \downarrow_l^i$

By assumption and from the definition for sntzA and rbstA, (III.9) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \Longrightarrow K) \downarrow_l^i \notin \{\mathsf{rls}(\_), \mathsf{down}(\_), \mathsf{sntz}(\_), \mathsf{t}(\_), (id.Ev(v), \_), ch(\_), \bullet\}$

From (III.9) and the definition of $\downarrow_l^i$ for T,

(III.10) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash \mathsf{last}(T) \xRightarrow{\alpha_l} K) \downarrow_l^i = \cdot$

From (III.10)

(III.11) $T \approx_l^i (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \xRightarrow{\alpha_l} K)$

From (III.4) and (III.11),

(III.12) $T_1 \approx_l^i (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \xRightarrow{\alpha_l} K)$

Let

(III.13) $\tau' = \mathsf{in}(T_1)$

From (III.12) and (III.13),

$\tau' \in \mathcal{I}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash T \xRightarrow{\alpha_l} K, \Sigma_0, \mathcal{R}, \mathcal{S}, \mathcal{P}, l)$

From (III.3) and (III.13),

$\tau \leq \tau'$

□

H.2 Supporting Lemmas

Lemma 9 (Equivalent Trace, Equivalent State). If $T_1 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1'$ and $T_2 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_2'$ with $K_1 \approx_l^p K_2$ and $T_1 \approx_l^p T_2$, then $K_1' \approx_l^p K_2'$

Proof.

By induction on $\mathsf{len}(T_1)$ and $\mathsf{len}(T_2)$

By assumption,

(1) $T_1 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1'$

(2) $T_2 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_2'$

(3) $K_1 \approx_l^p K_2$

(4) $T_1 \approx_l^p T_2$

Base Case I: $\mathsf{len}(T_1) = 0$ and $\mathsf{len}(T_2) = n$

By assumption and from (1),

(I.1) $T_1 = K_1$

(I.2) $K_1 = K_1'$

From (I.1),

(I.3) $T_1 \downarrow_l^p = \cdot$

From (4) and (I.3),

(I.4) $T_2 \downarrow_l^p = \cdot$

From (I.4) and Lemma 10,

(I.5) $K_2 \approx_l^p K_2'$

From (3), (I.2), and (I.5),

$K_1' \approx_l^p K_2'$

Base Case II: $\mathsf{len}(T_1) = n$ and $\mathsf{len}(T_2) = 0$

The proof is similar to **Base Case I**

Inductive Case III: $\mathsf{len}(T_1) = n + 1$ and $\mathsf{len}(T_2) = m + 1$

We assume the conclusion holds for $\mathsf{len}(T_1) \leq n$ and $\mathsf{len}(T_2) \leq m$

By assumption and from (1) and (2),

(III.1) $T_1 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1'' \Longrightarrow K_1'$ with

(III.2) $\mathsf{len}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1'') = n$

(III.3) $T_2 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_2'' \Longrightarrow K_2'$ with

(III.4) $\mathsf{len}(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_2'') = m$

Subcase i: $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1'' \Longrightarrow K_1') \downarrow_l^p = \cdot$

By assumption and from (III.1),

(i.1) $T_1 \downarrow_l^p = (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1'') \downarrow_l^p$

From (i.1),

(i.2) $T_1 \approx_l^p (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1'')$

From (4) and (i.2),

(i.3) $T_2 \approx_l^p (\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1'')$

From (3), (III.2), and (i.3),

The IH may be applied on $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1'')$ and T_2

IH on $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1'')$ and T_2 gives

(i.4) $K_1'' \approx_l^p K_2'$

By assumption and from Lemma 10,

(i.5) $K_1'' \approx_l^p K_1'$

From (i.4) and (i.5),

$K_1' \approx_l^p K_2'$

Subcase ii: $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2'' \Longrightarrow K_2') \downarrow_l^p = \cdot$

The proof is similar to **Subcase i**

Subcase iii: $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1'' \Longrightarrow K_1') \downarrow_l^p \neq \cdot$ and $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2'' \Longrightarrow K_2') \downarrow_l^p \neq \cdot$

By assumption and from (4),

(iii.1) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1'') \approx_l^p (\mathcal{P} \vdash K_2 \Longrightarrow^* K_2'')$ and

(iii.2) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1'' \Longrightarrow K_1') \approx_l^p (\mathcal{P} \vdash K_2'' \Longrightarrow K_2')$

From (3), (III.2), (III.4), and (iii.1),

The IH may be applied to $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1'')$ and $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_2'')$

IH on $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \Longrightarrow^* K_1'')$ and $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_2'')$ gives,

(iii.3) $K_1'' \approx_l^p K_2''$

By assumption and from (iii.2), (iii.3), and Lemma 15,

$K_1' \approx_l^p K_2'$

□

Lemma 10 (Empty Traces, Equivalent States). If $T = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \Longrightarrow^* K'$ and $T \downarrow_l^p = \cdot$, then $K \approx_l^p K'$

Proof.

By induction on the length of T.

By assumption,

(1) $T = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \Longrightarrow^* K'$

(2) $T \downarrow_l^p = \cdot$

Base Case I: $\mathsf{len}(T) = 0$

By assumption and from (1),

(I.1) $T = K$ and

(I.2) $K' = K$

From (I.2),

$K \approx_l^p K'$

Inductive Case II: $\mathsf{len}(T) = n + 1$

By assumption and from (1),

(II.1) $T = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \Longrightarrow^* K_1 \Longrightarrow K_2$

Want to show $K \approx_l^p K_2$

From (2) and (II.1),

(II.2) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \Longrightarrow^* K_1) \downarrow_l^p = \cdot$

From (II.2),

The IH may be applied on $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \Longrightarrow^* K_1)$

IH on $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \Longrightarrow^* K_1)$ gives
(II.3) $K \approx_l^p K_1$
Let $T' = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \overset{\alpha_l}{\Longrightarrow} K_2$
From (1),
(II.4) $T' \downarrow_l^p = \cdot$
Therefore, from (II.3), want to show $K_1 \approx_l^p K_2$

From (II.4) and the definition of $T \downarrow_l^p$,
(II.5) $pc \downarrow^p \not\sqsubseteq l$
From (II.4), when $\alpha_l = \mathsf{new}(id, pc_{src}), pc)$, then
(II.6) $pc \downarrow^p \not\sqsubseteq l$ with $pc_{src} \not\sqsubseteq pc$
From (II.4), when $\alpha_l = \mathsf{newEH}(id, eh, pc_{id}, pc_{src})$, then
(II.7) $pc \downarrow^p \not\sqsubseteq l$ with $pc_{src} \not\sqsubseteq pc$ or $pc_{id} \not\sqsubseteq pc$
From (II.5) - (II.7),
(II.8) $pc \downarrow^p \not\sqsubseteq l$ and $\alpha_l \in \{(\mathsf{new}(id, pc_{src}), pc), \mathsf{newEH}(id, eh, pc_{id}, pc_{eh})\}$ implies $pc_{src} \downarrow^p \not\sqsubseteq l$ and/or $pc_{id} \downarrow^p \not\sqsubseteq l$

Subcase i: T' ends in In
By assumption,
(i.1) $\Sigma_1 = \Sigma_2$
(i.2) $\mathsf{ks}_1 = \cdot$
(i.3) $\mathcal{P}(id.Ev(v)) = pc'$
(i.4) $E = ((id.Ev(v), pc'') \mid pc \sqcup pc' \sqsubseteq pc'')$
(i.5) $\Sigma, E \rightsquigarrow \mathsf{ks}_2$
(i.6) $\mathcal{R}_1 = \mathcal{R}_2$
(i.7) $\mathcal{S}_1 = \mathcal{S}_2$
From (II.4) and the definition of $T \downarrow_l^p$,
(i.8) $pc \downarrow^p \sqcup pc' \downarrow^p \not\sqsubseteq l$
From (i.3), (i.4), and (i.8),
(i.9) $\forall (id.Ev(v), pc'') \in E, pc'' \downarrow^p \not\sqsubseteq l$
From (i.9),
(i.10) $E \downarrow^p = \cdot$
From (i.5), (i.10) and Lemma 13,
(i.11) $\mathsf{ks}_2 \approx_l \cdot$
From (i.2) and (i.11),
(i.12) $\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$
From (i.1), (i.6), (i.7) and (i.12),
$K_1 \approx_l^p K_2$

Subcase ii: T' ends in In-D
By assumption,
(ii.1) $\Sigma_1 = \Sigma_2 = (\_, \sigma^{EH})$
(ii.2) $\mathcal{S}_1 = \mathcal{S}_2$
(ii.3) $\mathsf{ks}_1 = \cdot$
(ii.4) $\mathcal{P}(id.Ev(v)) = pc'$
(ii.5) $E = ((id.Ev(v), pc'') \mid pc \sqcup pc' \sqsubseteq pc'')$
(ii.6) $\mathsf{downgrade}_{\mathcal{D}}(\mathcal{R}_1, \Sigma_1, (id.Ev(v), pc), pc') = (\mathcal{R}_2, E')$
(ii.7) $\Sigma, E \rightsquigarrow \mathsf{ks}$
(ii.8) $pc, \mathsf{r} \vdash \Sigma, E' \rightsquigarrow \mathsf{ks}'$
(ii.9) $\mathsf{ks}_2 = \mathsf{ks} :: \mathsf{ks}'$
(ii.10) $\sigma^{EH}(id) \downarrow^i \sqsubseteq pc \downarrow^i$
(ii.11) $\sigma^{EH}(id) \downarrow^c \not\sqsubseteq pc \downarrow^c$
From (ii.6) and the definition of $\mathsf{downgrade}_{\mathcal{D}}$
(ii.12) $E_d = ((id.Ev(v), (l_c, l_i)) \mid pc' \downarrow^c \sqsubseteq l_c \sqsubset pc \downarrow^c \wedge l_i = pc \downarrow^i \sqcup pc' \downarrow^i)$
(ii.13) $\mathcal{R}_1 = (\rho_1, d_1)$
(ii.14) $\mathcal{D}((id.Ev(v), pc), pc', \rho_1) = (\rho_2, v_d, E_d')$

(ii.15) $d_2 = \mathsf{update}(d_1, v_d)$
(ii.16) $\mathcal{R}_2 = (\rho_2, d_2)$
(ii.17) $E' = \mathsf{robust}(\Sigma_1, E_d :: E'_d, pc)$
From (ii.10) and (ii.11),
(ii.18) $T' \downarrow^p_l = \mathsf{trRobust}(\ldots)$
From (ii.18), (II.4) and the definition of $\mathsf{trInput}$,
(ii.19) $pc \downarrow^p \sqcup pc' \downarrow^p \not\sqsubseteq l$
From (ii.19) and (ii.5),
(ii.20) $E \downarrow^p_l = \cdot$
From (ii.20), (ii.7) and Lemma 13,
(ii.21) $\mathsf{ks} \approx^p_l \cdot$
From (ii.18) and the definition of $\mathsf{trRobust}$,
(ii.22) $\mathcal{D}((id.Ev(v), pc), pc', \rho_1) = (\rho_1, \mathsf{none}, E'_d)$
(ii.23) $\mathsf{ks}' \downarrow^p_l = \cdot$
From (ii.23), (ii.21), (ii.3), and (ii.9),
(ii.24) $\mathsf{ks}_1 \approx^p_l \mathsf{ks}_2$
From (ii.15), (ii.14), and (ii.22),
(ii.25) $d_2 = d_1$
From (ii.16), (ii.14), (ii.22), (ii.25), and (ii.13),
(ii.26) $\mathcal{R}_1 = \mathcal{R}_2$
From (ii.26), (ii.2), (ii.1), and (a.5),
$K_1 \approx^p_l K_2$

Subcase iii: T' ends in In-E or In-DE
The proofs for these cases are similar to **Subcase ii**

Subcase iv: T' ends in Out
By assumption,
(iv.1) $\mathcal{R}_1 = \mathcal{R}_2$
(iv.2) $\mathcal{S}_1 = \mathcal{S}_2$
(iv.3) $\mathsf{ks}_1 = (\kappa, pc_{src}, pc) :: \mathsf{ks}$
(iv.4) $\mathcal{F} :: pc_{src}, d_d, d_e \vdash \Sigma_1, \kappa \longrightarrow_{pc} \Sigma_2, \mathsf{ks}'$
(iv.5) $\mathsf{ks}_2 = \mathsf{ks}' :: \mathsf{ks}$
(iv.6) $\alpha = ch(v)$
By assumption and from (II.4), (iv.6), and the definition of $T \downarrow^p_l$,
(iv.7) $pc \downarrow^p \not\sqsubseteq l$
From (iv.4), (iv.7), (II.8), and Lemma 11,
(iv.8) $\Sigma_1 \approx^p_l \Sigma_2$
(iv.9) $(\kappa, pc_{src}, pc) \approx^p_l \mathsf{ks}'$
From (iv.7) and (iv.9),
(iv.10) $\mathsf{ks}' \downarrow^p_l = \cdot$
(iv.11) $(\kappa, pc_{src}, pc) \downarrow^p_l = \cdot$
From (iv.1), (iv.2), (iv.8), (iv.10), (iv.11), (iv.3), and (iv.5)
$K_1 \approx^p_l K_2$

Subcase v: T' ends in Out-Skip or Out-Silent
The proofs for these cases are similar to **Subcase iv**

Subcase vi: T' ends in Out-Next
By assumption,
(vi.1) $\mathcal{R}_1 = \mathcal{R}_2$
(vi.2) $\mathcal{S}_1 = \mathcal{S}_2$
(vi.3) $\Sigma_1 = \Sigma_2$
(vi.4) $\mathsf{ks}_1 = (\kappa, pc_{src}, pc) :: \mathsf{ks}_2$
By assumption and from (II.4), (iv.6), and the definition of $T \downarrow^p_l$,

(vi.5) $pc \downarrow^p \not\sqsubseteq l$

From (vi.5),

(vi.6) $(\kappa, pc_{src}, pc) \downarrow_l^p = \cdot$

From (vi.4) and (vi.6),

(vi.7) $\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$

From (vi.1)-(vi.3), and (vi.7),

$K_1 \approx_l^p K_2$

□

Lemma 11. If $pc_{src}, d_d, d_e \vdash \Sigma_1, \kappa \xrightarrow{\alpha}_{pc} \Sigma_2, \mathsf{ks}$ *with* $pc \downarrow^p \not\sqsubseteq l$ *and* $\alpha \in \{\mathsf{new}(id, pc_{src}), \mathsf{newEH}(id, eh, pc_{src})\}$ *implies* $pc_{src} \downarrow^p \not\sqsubseteq l$ *and/or* $pc_{id} \downarrow^p \not\sqsubseteq l$*, then* $\Sigma_1 \approx_l^p \Sigma_2$ *and* $(\kappa, pc_{src}, pc) \approx_l^p \mathsf{ks}$

Proof.

We examine each case of $\mathcal{F} :: pc_{src}, d_d, d_e \vdash \Sigma_1, \kappa \longrightarrow_{pc} \Sigma_2, \mathsf{ks}$

By assumption,

(1) $pc \downarrow^p \not\sqsubseteq l$

(2) $\alpha \in \{\mathsf{new}(id, pc_{src}), \mathsf{newEH}(id, eh, pc_{src})\}$ implies $pc_{src} \downarrow^p \not\sqsubseteq l$ and/or $pc_{id} \downarrow^p \not\sqsubseteq l$

Case I: $\mathcal{F}$ ends in PtoC

By assumption,

(I.1) $\mathsf{ks} = (\sigma, \mathsf{skip}, P, \cdot), pc_{src}, pc$

(I.2) $\Sigma_1 = \Sigma_2$

From (2),

(I.3) $(\kappa, pc_{src}, pc) \downarrow_l^p = \cdot$

From (I.1) and (1),

(I.4) $\mathsf{ks} \downarrow_l^p = \cdot$

From (I.3) and (I.4),

$(\kappa, pc_{src}, pc) \approx_l^p \mathsf{ks}$

From (I.2),

$\Sigma_1 \approx_l^p \Sigma_2$

Case II: $\mathcal{F}$ ends in PtoLC

By assumption,

(II.1) $\Sigma_1, E \rightsquigarrow \mathsf{ks}'$

(II.2) $\mathsf{ks} = ((\sigma, \mathsf{skip}, C, \cdot), pc_{src}, pc) :: \mathsf{ks}'$

(II.3) $\Sigma_1 = \Sigma_2$

Since event-trigger is the only rule to add to E,

(II.4) $\forall (id'.Ev'(v'), pc') \in E, pc' = pc$

From (1) and (II.4),

(II.5) $E \downarrow_l^p = \cdot$

From (II.5), (II.1), and Lemma 13,

(II.6) $\mathsf{ks}' \approx_l^p \cdot$

From (1),

(II.7) $(\kappa, pc_{src}, pc) \downarrow_l^p = \cdot$

From (1), (II.6), and (II.2),

(II.8) $\mathsf{ks} \downarrow_l^p = \cdot$

From (II.7) and (II.8),

$(\kappa, pc_{src}, pc) \approx_l^p \mathsf{ks}$

From (II.3),

$\Sigma_1 \approx_l^p \Sigma_2$

Case III: $\mathcal{F}$ ends in P

The proof for this case follows from (1), (2), and Lemma 12

□

Lemma 12. If $pc_{src}, d_d, d_e \vdash \Sigma_1, \sigma_1, c_1 \xrightarrow{\alpha}_{pc} \Sigma_2, \sigma_2, c_2, E$ with $pc \downarrow^p \not\sqsubseteq l$ and $\alpha \in \{\mathsf{new}(id, pc_{src}), \mathsf{newEH}(id, eh, pc_{src})\}$ implies $pc_{src} \downarrow^p \not\sqsubseteq l$ and/or $pc_{id} \downarrow^p \not\sqsubseteq l$, then $\Sigma_1 \approx_l^p \Sigma_2$

Proof.
By induction on the structure of
$\mathcal{F} :: pc_{src}, d_d, d_e \vdash \Sigma_1, \sigma_1, c_1 \longrightarrow_{pc} \Sigma_2, \sigma_2, c_2, E$
By assumption,
(1) $pc \downarrow^p \not\sqsubseteq l$
(2) $\alpha \in \{\mathsf{new}(id, pc_{src}), \mathsf{newEH}(id, eh, pc_{src})\}$ implies $pc_{src} \downarrow^p \not\sqsubseteq l$ and/or $pc_{id} \downarrow^p \not\sqsubseteq l$

Case I: $\mathcal{F}$ ends in a rule which does not modify Σ
In these cases, the conclusion follows from $\Sigma_2 = \Sigma_1$

Case II: $\mathcal{F}$ ends in SEQ
By assumption,
$\exists \mathcal{G} :: pc_s rc, d_d, d_e \vdash \Sigma_1, \sigma, c_1 \longrightarrow_{pc} \Sigma_2, \sigma_2, c_1', E$
The proof follows from the induction hypothesis on $\mathcal{G}$

Case III: $\mathcal{F}$ ends in ASSIGN-G
By assumption,
(III.1) $\Sigma_2 = \Sigma_1[pc \mapsto (\sigma_2^g, \sigma^{EH})]$
From (1) and (III.1),
$\Sigma_1 \approx_l^p \Sigma_2$

Case IV: $\mathcal{F}$ ends in UPDATE
The proof for this cases is similar to **Case III**

Case V: $\mathcal{F}$ ends in NEW
By assumption and from (2) and $\Sigma_1(pc) = (\sigma^g, \sigma^{EH})$
(V.1) $id \notin \sigma^{EH}$
(V.2) $\sigma^{EH'} = \sigma^{EH}[id \mapsto (v, \cdot, pc_{src})]$
(V.3) $\Sigma_2 = \Sigma_1[pc \mapsto (\sigma^g, \sigma^{EH'})]$
(V.4) $\alpha = \mathsf{new}(id, pc_{src})$
From (2), (V.4), (V.1), (V.2), and the definition of $\approx_l^p$ for σ^{EH},
(V.5) $\sigma^{EH} \approx_l^p \sigma^{EH'}$
From (V.3), (V.5), and the definition of $\approx_l^p$ for Σ,
$\Sigma_1 \approx_l^p \Sigma_2$

Case VI: $\mathcal{F}$ ends in ADD-EH
By assumption and from (2) and $\Sigma_1(pc) = (\sigma^g, \sigma^{EH})$
(VI.1) $\sigma^{EH}(id) = (v, M, pc_{id})$ with $M(Ev) = EH$
(VI.2) $M' = M[Ev \mapsto EH \cup \{(eh, pc_{src})\}]$
(VI.3) $\sigma^{EH'} = \sigma^{EH}[id \mapsto (v, M', pc_{id})]$
(VI.4) $\Sigma_2 = \Sigma_1[pc \mapsto (\sigma^g, \sigma^{EH'})]$
(VI.5) $\alpha = \mathsf{newEH}(id, eh, pc_{id}, pc_{src})$
From (2), (VI.5), (VI.1)-(VI.3), and the definition of $\approx_l^p$ for σ^{EH},
(VI.6) $\sigma^{EH} \approx_l^p \sigma^{EH'}$
From (VI.4), (VI.6), and the definition of $\approx_l^p$ for Σ,
$\Sigma_1 \approx_l^p \Sigma_2$

$\square$

Lemma 13 (Secret EH Lookups are Not Observable). If $\Sigma, E \rightsquigarrow \mathsf{ks}$ with $E \downarrow_l^p = \cdot$ then $\mathsf{ks} \approx_l^p \cdot$

Proof.
By induction on the structure of $\mathcal{F} :: \Sigma, E \rightsquigarrow \mathsf{ks}$

By assumption,
(1) $E \downarrow^p = \cdot$

Case I: $\mathcal{F}$ ends in LOOKUP
By assumption,
(I.1) $E = (id.Ev(v), pc) :: E'$
(I.2) $\Sigma(pc) = (\_, \sigma^{EH})$ and $\sigma^{EH}(id) = (\_, M, pc_{id})$
(I.3) $\exists \mathcal{G} :: pc, pc_{id}, v \vdash M(Ev) \rightsquigarrow \mathsf{ks}_1$
(I.4) $\exists \mathcal{G}' :: \Sigma, E' \rightsquigarrow \mathsf{ks}_2$
(I.5) $\mathsf{ks} = \mathsf{ks}_1 :: \mathsf{ks}_2$
From (1) and (I.1),
(I.6) $pc \downarrow^p \not\sqsubseteq l$
(I.7) $E' \downarrow^p_l = \cdot$
From (I.6), (I.3) and Lemma 14,
(I.8) $\mathsf{ks}_1 \approx^p_l \cdot$
From (I.4), (I.7) and IH on $\mathcal{G}$,
(I.9) $\mathsf{ks}_2 \approx^p_l \cdot$
From (I.5), (I.8), and (I.9),
$\mathsf{ks} \approx^p_l \cdot$

Case II: $\mathcal{F}$ ends in LOOKUP-MISSING
By assumption,
(II.1) $E = (id.Ev(v), pc) :: E'$
(II.2) $\exists \mathcal{G} :: \Sigma, E' \rightsquigarrow \mathsf{ks}$
From (1) and (II.1),
(II.3) $E' \downarrow^p_l = \cdot$
From (II.3), (II.2) and IH on $\mathcal{G}$,
$\mathsf{ks} \approx^p_l \cdot$

Case III: $\mathcal{F}$ ends in LOOKUP-EMPTY
By assumption, $\mathsf{ks} = \cdot$

□

Lemma 14. If $pc, pc_{id}, v \vdash EH \rightsquigarrow \mathsf{ks}$ with $pc \downarrow^p \not\sqsubseteq l$, then $\mathsf{ks} \approx^p_l \cdot$

PROOF.
By induction on the structure of $\mathcal{F} :: pc, pc_{id}, v \vdash EH \rightsquigarrow \mathsf{ks}$

By assumption,
(1) $pc \downarrow^p \not\sqsubseteq l$

Case I: $\mathcal{F}$ ends in LOOKUPEH
By assumption,
(I.1) $EH = \{(EH, pc_{eh})\} \cup EH'$
(I.2) $\mathsf{ks}_1 = ((\cdot, eh(v), P, \cdot), pc_{id} \sqcup pc_{eh}, pc)$
(I.3) $\exists \mathcal{G} :: pc, pc_{id}, v \vdash EH' \rightsquigarrow \mathsf{ks}_2$
(I.4) $\mathsf{ks} = \mathsf{ks}_1 :: \mathsf{ks}_2$
From (1) and (I.2),
(I.5) $\mathsf{ks}_1 \approx^p_l \cdot$
From (1), (I.3) and IH on $\mathcal{G}$,
(I.6) $\mathsf{ks}_2 \approx^p_l \cdot$
From (I.4)-(I.6),
$\mathsf{ks} \approx^p_l \cdot$

Case II: $\mathcal{F}$ ends in LOOKUPEH-EMP

By assumption, $\mathrm{ks} = \cdot$

□

Lemma 15 (Weak One-Step). If $T_1 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \stackrel{\alpha_{l,1}}{\Longrightarrow} K_1'$ *and* $T_2 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \stackrel{\alpha_{l,2}}{\Longrightarrow} K_2'$*, with* $T_1 \approx_l^p T_2$, $K_1 \approx_l^p K_2$, $T_1 \downarrow_l^p \neq \cdot$*, and* $T_2 \downarrow_l^p \neq \cdot$*, then* $K_1' \approx_l^p K_2'$

Proof.

We examine each case of $\mathcal{F} :: \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \stackrel{\alpha_{l,1}}{\Longrightarrow} K_1'$
Denote $\mathcal{G} :: \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \stackrel{\alpha_{l,2}}{\Longrightarrow} K_2'$
By assumption,
(1) $K_1 \approx_l^p K_2$
(2) $T_1 \approx_l^p T_2$
(3) $T_1 \downarrow_l^p \neq \cdot$
(4) $T_2 \downarrow_l^p \neq \cdot$
From (1),
(5) $\mathcal{R}_1 = \mathcal{R}_2$
(6) $\mathcal{S}_1 = \mathcal{S}_2$
(7) $\Sigma_1 \approx_l^p \Sigma_2$
(8) $\mathrm{ks}_1 \approx_l^p \mathrm{ks}_2$

Case I: $\mathcal{F}$ ends in In
By assumption and from $\Sigma_1(pc_1) = (\_, \sigma_1^{EH})$,
(I.1) $\mathcal{R}_1 = \mathcal{R}_1'$
(I.2) $\mathcal{S}_1 = \mathcal{S}_1'$
(I.3) $\Sigma_1 = \Sigma_1'$
(I.4) $\alpha_{l,1} = (id.Ev(v), pc_1)$
(I.5) $\mathcal{P}(id.Ev(v)) = pc_1'$
(I.6) $E_1 = ((id.Ev(v), pc_1'') \mid pc_1 \sqcup pc_1' \sqsubseteq pc_1'')$
(I.7) $\sigma_1^{EH}(id) \downarrow^i \not\sqsubseteq pc_1 \downarrow^i$
(I.8) $\sigma_1^{EH}(id) \downarrow^c \not\sqsubseteq pc_1 \downarrow^c$
(I.9) $\Sigma_1, E_1 \rightsquigarrow \mathrm{ks}_1'$
From (3), (I.4), (I.5), (I.5), (I.7), (I.8), and the definition of $T \downarrow_l^p$,
(I.10) $T_1 \downarrow_l^p = (id.Ev(v), pc_1)$
From (I.10) and (3),
(I.11) $pc_1 \downarrow^p \sqcup pc_1' \downarrow^p \sqsubseteq l$
From (I.10) and (2),
(I.12) $T_2 \downarrow_l^p = (id.Ev(v), pc_1)$
From (I.12) and (4),
(I.13) $\alpha_{l,2} = (id.Ev(v), pc_2)$
(I.14) $\mathcal{P}(id.Ev(v)) = pc_2'$
From (I.5) and (I.14),
(I.15) $pc_1' = pc_2'$
From (I.12) and (I.13),
(I.16) $pc_1 = pc_2$

Subcase i: $\mathcal{G}$ ends in In
By assumption,
(i.1) $\mathcal{R}_2 = \mathcal{R}_2'$
(i.2) $\mathcal{S}_2 = \mathcal{S}_2'$
(i.3) $\Sigma_2 = \Sigma_2'$
(i.4) $E_2 = ((id.Ev(v), pc_2'') \mid pc_2 \sqcup pc_2' \sqsubseteq pc_2'')$
(i.5) $\Sigma_2, E_2 \rightsquigarrow \mathrm{ks}_2'$
From (I.15), (I.16), (I.6), and (i.4),
(i.6) $E_1 \approx_l^p E_2$
From (7), (I.9), (i.5), (i.6) and Lemma 19,

(i.7) $\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$

From (5)-(7), (I.1)-(I.3) and (i.1)-(i.3),

(i.8) $\mathcal{R}_1' = \mathcal{R}_2'$

(i.9) $\mathcal{S}_1' = \mathcal{S}_2'$

(i.10) $\Sigma_1' \downarrow^p = \Sigma_2' \downarrow^p$

From (i.7)-(i.10),

$K_1' \approx_l^p K_2'$

Subcase ii: $\mathcal{G}$ ends in IN-D

By assumption and from $\Sigma_2(pc_2) = (\_, \sigma_2^{EH})$,

(ii.1) $\sigma_2^{EH}(id) \downarrow^i \sqsubseteq pc_2 \downarrow^i$

(ii.2) $\sigma_2^{EH}(id) \downarrow^c \not\sqsubseteq pc_2 \downarrow^c$

From (7) and (I.16),

(ii.3) If $pc_1 \downarrow^p \sqsubseteq l$, then $\sigma_1^{EH} = \sigma_2^{EH}$

From (ii.3), (I.7), and (ii.1),

(ii.4) $pc_1 \downarrow^p \not\sqsubseteq l$

But (ii.4) contradicts (I.11), so this case holds vacuously

Subcase iii: $\mathcal{G}$ ends in IN-E or IN-DE

The proofs for these cases are similar to **Subcase ii**

Case II: $\mathcal{F}$ ends in IN-D

By assumption and from $\Sigma_1(pc_1) = (\_, \sigma_1^{EH})$,

(II.1) $\mathcal{S}_1 = \mathcal{S}_1'$

(II.2) $\Sigma_1 = \Sigma_1'$

(II.3) $\mathcal{P}(id.Ev(v)) = pc_1'$

(II.4) $E_1 = ((id.Ev(v), pc_1'') \mid pc_1 \sqcup pc_1' \sqsubseteq pc_1'')$

(II.5) $\sigma_1^{EH}(id) \downarrow^i \sqsubseteq pc_1 \downarrow^i$

(II.6) $\sigma_1^{EH}(id) \downarrow^c \not\sqsubseteq pc_1 \downarrow^c$

(II.7) $(\mathcal{R}_1', E_1') = \mathsf{downgrade}_{\mathcal{D}}(\mathcal{R}_1, \Sigma_1, (id.Ev(v), pc_1), pc_1')$

(II.8) $\Sigma_1, E_1 \rightsquigarrow \mathsf{ks}_1''$

(II.9) $pc_1, \mathsf{r} \vdash \Sigma_1, E_1' \rightsquigarrow \mathsf{ks}_1'''$

(II.10) $\mathsf{ks}_1' = \mathsf{ks}_1'' :: \mathsf{ks}_1'''$

From (II.7) and the definition of $\mathsf{downgrade}_{\mathcal{D}}$,

(II.11) $E_{d,1} = ((id.Ev(v), (l_c, l_i)) \mid pc_1' \downarrow^c \sqsubseteq l_c \sqsubset pc_1 \downarrow^c \wedge l_i = pc_1 \downarrow^i \sqcup pc_1' \downarrow^i)$

(II.12) $\mathcal{R}_1 = (\rho_1, d_1)$

(II.13) $\mathcal{D}((id.Ev(v), pc_1), pc_1', \rho_1) = (\rho_1', v_1, E_{d,1}')$

(II.14) $d_1' = \mathsf{update}(d_1, v_1)$

(II.15) $\mathcal{R}_1' = (\rho_1', d_1')$

(II.16) $E_1' = \mathsf{robust}(\Sigma_1, E_{d,1} :: E_{d,1}', pc_1)$

From (3), (II.5), (II.6), and the definition of $T \downarrow_l^p$,

(II.17) $T_1 \downarrow_l^p = (id.Ev(v), pc_1)$ or

(II.18) $T_1 \downarrow_l^p = \mathsf{rls}(id.Ev(v), \rho_1', v_1, E_1'')$

Subcase i: $T_1 \downarrow_l^p = (id.Ev(v), pc_1)$

From (II.17) and (II.9),

(i.1) $\mathcal{D}((id.Ev(v), pc_1), pc_1', \rho_1) = (\rho_1, \mathsf{none}, E_{d,1}')$

(i.2) $\mathsf{ks}_1''' \downarrow_l^p = \cdot$ if $p = c$ and $\mathsf{ks}_1''' = \cdot$ if $p = i$

From (II.17) and (2),

(i.3) $T_2 \downarrow_l^p = (id.Ev(v), pc_1)$

From (II.17) and (i.3),

(i.4) $pc_1 \downarrow^p \sqcup pc_1' \downarrow^p \not\sqsubseteq l$

(i.5) $pc_2 \downarrow^p \sqcup pc_2' \downarrow^p \not\sqsubseteq l$

(i.6) $pc_1 = pc_2$

From (i.4)-(i.6),

(i.7) $pc_1 \downarrow^p \sqsubseteq l$

(i.8) $pc_2 \downarrow^p \sqsubseteq l$

From (7) and (i.6)-(i.8),

(i.9) $\sigma_1^{EH} = \sigma_2^{EH}$

From (i.9), (II.5), and (II.6),

(i.10) $\sigma_2^{EH}(id) \downarrow^i \sqsubseteq pc_2 \downarrow^i$

(i.11) $\sigma_2^{EH}(id) \downarrow^c \not\sqsubseteq pc_2 \downarrow^c$

From (i.11) and (i.12),

(i.12) $\mathcal{G}$ must end in In-D

From (i.12),

(i.13) $\mathcal{S}_2 = \mathcal{S}_2'$

(i.14) $\Sigma_2 = \Sigma_2'$

(i.15) $\mathcal{P}(id.Ev(v)) = pc_2'$

(i.16) $E_2 = ((id.Ev(v), pc_2'') \mid pc_2 \sqcup pc_2' \sqsubseteq pc_2'')$

(i.17) $\sigma_2^{EH}(id) \downarrow^i \sqsubseteq pc_2 \downarrow^i$

(i.18) $\sigma_2^{EH}(id) \downarrow^c \not\sqsubseteq pc_2 \downarrow^c$

(i.19) $(\mathcal{R}_2', E_2') = \mathsf{downgrade}_{\mathcal{D}}(\mathcal{R}_2, \Sigma_2, (id.Ev(v), pc_2), pc_2')$

(i.20) $\Sigma_2, E_2 \rightsquigarrow \mathsf{ks}_2''$

(i.21) $pc_2, \mathsf{r} \vdash \Sigma_2, E_2' \rightsquigarrow \mathsf{ks}_2'''$

(i.22) $\mathsf{ks}_2' = \mathsf{ks}_2'' :: \mathsf{ks}_2'''$

From (i.19) and the definition of $\mathsf{downgrade}_{\mathcal{D}}$,

(i.23) $E_{d,2} = ((id.Ev(v), (l_c, l_i)) \mid pc_2' \downarrow^c \sqsubseteq l_c \sqsubset pc_2 \downarrow^c \wedge l_i = pc_2 \downarrow^i \sqcup pc_2' \downarrow^i)$

(i.24) $\mathcal{R}_2 = (\rho_2, d_2)$

(i.25) $\mathcal{D}((id.Ev(v), pc_2), pc_2', \rho_2) = (\rho_2', v_2, E_{d,2}')$

(i.26) $d_2' = \mathsf{update}(d_2, v_2)$

(i.27) $\mathcal{R}_2' = (\rho_2', d_2')$

(i.28) $E_2' = \mathsf{robust}(\Sigma_2, E_{d,2} :: E_{d,2}', pc_1)$

From (i.3) and (i.21),

(i.29) $\mathcal{D}((id.Ev(v), pc_2), pc_2', \rho_2) = (\rho_2, \mathsf{none}, E_{d,2}')$

(i.30) $\mathsf{ks}_2''' \downarrow_l^p = \cdot$ if $p = c$ or $\mathsf{ks}_2''' = \cdot$ if $p = i$

From (II.13), (i.1), and (II.14),

(i.31) $d_1' = d_1$

From (II.13), (i.1), (i.31), (II.12), and (II.15),

(i.32) $\mathcal{R}_1' = \mathcal{R}_1$

From (i.25), (i.29), and (i.26),

(i.33) $d_2' = d_2$

From (i.25), (i.29), (i.33), (i.24), and (i.27),

(i.34) $\mathcal{R}_2' = \mathcal{R}_2$

From (5), (i.32), and (i.34),

(i.35) $\mathcal{R}_1' = \mathcal{R}_2'$

From (6), (II.1), and (i.14),

(i.36) $\mathcal{S}_1' = \mathcal{S}_2'$

From (7), (II.2), and (i.14),

(i.37) $\Sigma_1' \approx_l^p \Sigma_2'$

From (II.3) and (i.15),

(i.38) $pc_1' = pc_2'$

From (i.6), (i.38), (II.4), and (i.16),

(i.39) $E_1 = E_2$

From (i.39), (7), and from Lemma 19,

(i.40) $\mathsf{ks}_1'' \approx_l^p \mathsf{ks}_2''$

From (II.10), (i.22), (i.40), (i.2), and (i.30),

(i.41) $\mathsf{ks}_1' \approx_l^p \mathsf{ks}_2'$

From (i.35)-(i.37) and (i.41),

$K_1' \approx_l^p K_2'$

Subcase ii: $T_1 \downarrow_l^p = \mathsf{rls}(id.Ev(v), \rho_1', v_1, E_1'', pc_1)$

From (II.18) and (2),

(ii.1) $T_2 \downarrow_l^p = \mathsf{rls}(id.Ev(v), \rho_1', v_1, E_1'', pc_1)$ where $E_1'' = E_1' \downarrow_l^p$ if $p = c$ and $E_1'' = E_1'$ if $p = i$

From (ii.1),

(ii.2) $pc_1 = pc_2$

From (ii.1) and the definition of $T \downarrow_l^p$,

(ii.3) $\mathcal{G}$ ends in In-D or

(ii.4) $\mathcal{G}$ ends in In-DE

Subsubcase a: $\mathcal{G}$ ends in In-D

From (ii.3),

(a.1) $\mathcal{S}_2 = \mathcal{S}_2'$

(a.2) $\Sigma_2 = \Sigma_2'$

(a.3) $\mathcal{P}(id.Ev(v)) = pc_2'$

(a.4) $E_2 = ((id.Ev(v), pc_2'') \mid pc_2 \sqcup pc_2' \sqsubseteq pc_2'')$

(a.5) $(\mathcal{R}_2', E_2') = \mathsf{downgrade}_{\mathcal{D}}(\mathcal{R}_2, \Sigma_2, (id.Ev(v), pc_2), pc_2')$

(a.6) $\Sigma_2, E_2 \rightsquigarrow \mathsf{ks}_2''$

(a.7) $pc_2, \mathsf{r} \vdash \Sigma_2, E_2' \rightsquigarrow \mathsf{ks}_2'''$

(a.8) $\mathsf{ks}_2' = \mathsf{ks}_2'' :: \mathsf{ks}_2'''$

From (a.5) and the definition of $\mathsf{downgrade}_{\mathcal{D}}$,

(a.9) $E_{d,2} = ((id.Ev(v), (l_c, l_i)) \mid pc_2' \downarrow^c \sqsubseteq l_c \sqsubset pc_2 \downarrow^c \wedge l_i = pc_2 \downarrow^i \sqcup pc_2' \downarrow^i)$

(a.10) $\mathcal{R}_2 = (\rho_2, d_2)$

(a.11) $\mathcal{D}((id.Ev(v), pc_2), pc_2', \rho_2) = (\rho_2', v_2, E_{d,2}')$

(a.12) $d_2' = \mathsf{update}(d_2, v_2)$

(a.13) $\mathcal{R}_2' = (\rho_2', d_2')$

(a.14) $E_2' = \mathsf{robust}(\Sigma_2, E_{d,2} :: E_{d,2}', pc_2)$

From (ii.1),

(a.15) $\mathcal{D}((id.Ev(v), pc_2), pc_2', \rho_2) = (\rho_1', v_1, E_{d,2}')$

(a.16) $E_1' \downarrow_l^p = E_2' \downarrow_l^p$ if $p = c$ or $E_1' = E_2'$ if $p = i$

From (a.11) and (a.15),

(a.17) $\rho_1' = \rho_2'$

(a.18) $v_1 = v_2$

From (II.14), (a.12), and (a.18),

(a.19) $d_1' = d_2'$

From (5), (II.15), (a.13), (a.11), (a.15), and (a.19),

(a.20) $\mathcal{R}_1' = \mathcal{R}_2'$

From (6), (II.1), and (a.1),

(a.21) $\mathcal{S}_1' = \mathcal{S}_2'$

From (7), (II.2), and (a.2),

(a.22) $\Sigma_1' = \Sigma_2'$

From (II.3) and (a.3),

(a.23) $pc_1' = pc_2'$

From (ii.2), (a.23), (II.4), and (a.4),

(a.24) $E_1 = E_2$

From (7), (a.24), (II.8), (a.6), and Lemma 19,

(a.25) $\mathsf{ks}_1'' \approx_l^p \mathsf{ks}_2''$

From (7), (ii.4), (a.16), (II.9), (a.7), and Lemma 20,

(a.26) $\mathsf{ks}_1''' \approx_l^p \mathsf{ks}_2'''$

From (II.10), (a.8), (a.25), and (a.26),

(a.27) $\mathsf{ks}_1' \approx_l^p \mathsf{ks}_2'$

From (a.20)-(a.22) and (a.27),

$K_1' \approx_l^p K_2'$

Subsubcase b: $\mathcal{G}$ ends in In-DE

From (ii.4),

(b.1) $\Sigma_2 = \Sigma'_2$
(b.2) $\mathcal{P}(id.Ev(v)) = pc'_2$
(b.3) $E_2 = ((id.Ev(v), pc''_2) \mid pc_2 \sqcup pc'_2 \sqsubseteq pc''_2)$
(b.4) $(\mathcal{R}'_2, E''_{d,2}) = \mathsf{downgrade}_{\mathcal{D}}(\mathcal{R}_2, \Sigma_2, (id.Ev(v), pc_2), pc'_2)$
(b.5) $(\mathcal{S}'_2, E''_{e,2}) = \mathsf{downgrade}_{\mathcal{E}}(\mathcal{S}_2, \Sigma_2, (id.Ev(v), pc_2), pc'_2)$
(b.6) $E'_2 = \mathsf{downgrade}_{\mathcal{D},\mathcal{E}}(\mathcal{R}_2, \mathcal{S}_2, \Sigma_2, (id.Ev(v), pc_2), pc'_2)$
(b.7) $\Sigma_2, E_2 \rightsquigarrow \mathsf{ks}''_2$
(b.8) $pc_2, \mathsf{r} \vdash \Sigma_2, E''_{d,2} \rightsquigarrow \mathsf{ks}_{d,2}$
(b.9) $pc_2, \mathsf{t} \vdash \Sigma_2, E''_{e,2} \rightsquigarrow \mathsf{ks}_{e,2}$
(b.10) $pc_2, \mathsf{rt} \vdash \Sigma_2, E'_2 \rightsquigarrow \mathsf{ks}'''_2$
(b.11) $\mathsf{ks}'_2 = \mathsf{ks}''_2 :: \mathsf{ks}_{d,2} :: \mathsf{ks}_{e,2} :: \mathsf{ks}'''_2$

From (b.4) and the definition of $\mathsf{downgrade}_{\mathcal{D}}$,
(b.12) $E_{d,2} = ((id.Ev(v), (l_c, l_i)) \mid pc'_2 \downarrow^c \sqsubseteq l_c \sqsubset pc_2 \downarrow^c \wedge l_i = pc_2 \downarrow^i \sqcup pc'_2 \downarrow^i)$
(b.13) $\mathcal{R}_2 = (\rho_{d,2}, d_{d,2})$
(b.14) $\mathcal{D}((id.Ev(v), pc_2), pc'_2, \rho_{d,2}) = (\rho'_{d,2}, v_{d,2}, E'_{d,2})$
(b.15) $d'_{d,2} = \mathsf{update}(d_{d,2}, v_{d,2})$
(b.16) $\mathcal{R}'_2 = (\rho'_{d,2}, d'_{d,2})$
(b.17) $E''_{d,2} = \mathsf{robust}(\Sigma_2, E_{d,2} :: E'_{d,2}, pc_2)$

From (b.5) and the definition of $\mathsf{downgrade}_{\mathcal{E}}$,
(b.18) $E_{e,2} = ((id.Ev(v), (l_c, l_i)) \mid l_c = pc_2 \downarrow^c \sqcup pc'_2 \downarrow^c \wedge pc'_2 \downarrow^i \sqsubseteq l_i \sqsubset pc_2 \downarrow^i)$
(b.19) $\mathcal{S}_2 = (\rho_{e,2}, d_{e,2})$
(b.20) $\mathcal{D}((id.Ev(v), pc_2), pc'_2, \rho_{e,2}) = (\rho'_{e,2}, v_{e,2}, E'_{e,2})$
(b.21) $d'_{e,2} = \mathsf{update}(d_{e,2}, v_{e,2})$
(b.22) $\mathcal{S}'_2 = (\rho'_{e,2}, d'_{e,2})$
(b.23) $E''_{e,2} = \mathsf{transparent}(\Sigma_2, E_{e,2} :: E'_{e,2}, pc_2)$

From (b.6) and the definition of $\mathsf{downgrade}_{\mathcal{D},\mathcal{E}}$
(b.24) $E_{d,e} = \mathsf{mergeEvents}(E_{d,2} :: E'_{d,2}, E_{e,2} :: E'_{e,2})$
(b.25) $E'_2 = \mathsf{robustTransparent}(\Sigma_2, E_{d,e}, pc_2)$

From (ii.1),
(b.26) $\mathcal{D}((id.Ev(v), pc_2), pc'_2, \rho_{d,2}) = (\rho'_1, v_1, E'_{d,2})$
(b.27) $E'_1 \downarrow^p_l = E''_{d,2} \downarrow^p_l$ if $p = c$ and $E'_1 = E''_{d,2}$ if $p = i$
(b.28) $\rho'_{e,2} = \rho_{e,2}$
(b.29) $v_{e,2} = \mathsf{none}$
(b.30) $\mathsf{ks}_{e,2} = \cdot$ if $p = c$ and $\mathsf{ks}_{e,2} \downarrow^p_l = \cdot$ if $p = i$
(b.31) $\mathsf{ks}'''_2 \downarrow^p = \cdot$

From (b.14) and (b.26),
(b.32) $\rho'_1 = \rho'_{d,2}$
(b.33) $v_1 = v_{d,2}$

From (II.14), (b.15), and (b.33),
(b.34) $d'_1 = d'_{d,2}$

From (5), (II.15), (b.16), (b.14), (b.26), and (b.34),
(b.35) $\mathcal{R}'_1 = \mathcal{R}'_2$

From (b.21) and (b.29),
(b.36) $d_{e,2} = d'_{e,2}$

From (b.19), (b.22), (b.28), and (b.36),
(b.37) $\mathcal{S}'_1 = \mathcal{S}'_2$

From (7), (II.2), and (b.1),
(b.38) $\Sigma'_1 = \Sigma'_2$

From (II.3) and (b.2),
(b.39) $pc'_1 = pc'_2$

From (ii.2), (b.39), (II.4), and (b.3),
(b.40) $E_1 = E_2$

From (7), (b.49), (II.8), (b.7), and Lemma 19,
(b.41) $\mathsf{ks}''_1 \approx^p_l \mathsf{ks}''_2$

From (7), (ii.4), (b.27), (II.9), (b.8), and Lemma 20,

(b.42) $\mathsf{ks}_1''' \approx_l^p \mathsf{ks}_{d,2}$

From (II.10), (b.11), (b.41), (b.42), (b.30), and (b.31),

(b.43) $\mathsf{ks}_1' \approx_l^p \mathsf{ks}_2'$

From (b.35), (b.37), (b.38), and (b.43)

$K_1' \approx_l^p K_2'$

Case III: $\mathcal{F}$ ends in IN-E

The proofs for these cases are similar to the one for **Case II**

Case IV: $\mathcal{F}$ ends in IN-DE

By assumption and from $\Sigma_1(pc_1) = (\_, \sigma_1^{EH})$,

(IV.1) $\Sigma_1 = \Sigma_1'$

(IV.2) $\mathcal{P}(id.Ev(v)) = pc_1'$

(IV.3) $E_1 = ((id.Ev(v), pc_1'') \mid pc_1 \sqcup pc_1' \sqsubseteq pc_1'')$

(IV.4) $\sigma_1^{EH}(id) \downarrow^i \sqsubseteq pc_1 \downarrow^i$

(IV.5) $\sigma_1^{EH}(id) \downarrow^c \sqsubseteq pc_1 \downarrow^c$

(IV.6) $(\mathcal{R}_1', E_1') = \mathsf{downgrade}_{\mathcal{D}}(\mathcal{R}_1, \Sigma_1, (id.Ev(v), pc_1), pc_1')$

(IV.7) $(\mathcal{S}_1', E_1'') = \mathsf{downgrade}_{\mathcal{E}}(\mathcal{S}_1, \Sigma_1, (id.Ev(v), pc_1), pc_1')$

(IV.8) $E_1''' = \mathsf{downgrade}_{\mathcal{D},\mathcal{E}}(\mathcal{R}_1, \mathcal{S}_1, \Sigma_1, (id.Ev(v), pc_1), pc_1')$

(IV.9) $\Sigma_1, E_1 \rightsquigarrow \mathsf{ks}_1''$

(IV.10) $pc_1, \mathsf{r} \vdash \Sigma_1, E_1' \rightsquigarrow \mathsf{ks}_{d,1}$

(IV.11) $pc_1, \mathsf{t} \vdash \Sigma_1, E_1'' \rightsquigarrow \mathsf{ks}_{e,1}$

(IV.12) $pc_1, \mathsf{rt} \vdash \Sigma_1, E_1''' \rightsquigarrow \mathsf{ks}_{m,1}$

(IV.13) $\mathsf{ks}_1' = \mathsf{ks}_1'' :: \mathsf{ks}_{d,1} :: \mathsf{ks}_{e,1} :: \mathsf{ks}_{m,1}$

From (IV.6) and the definition of $\mathsf{downgrade}_{\mathcal{D}}$,

(IV.14) $E_{d,1} = ((id.Ev(v), (l_c, l_i)) \mid pc_1' \downarrow^c \sqsubseteq l_c \sqsubset pc_1 \downarrow^c \wedge l_i = pc_1 \downarrow^i \sqcup pc_1' \downarrow^i)$

(IV.15) $\mathcal{R}_1 = (\rho_{d,1}, d_{d,1})$

(IV.16) $\mathcal{D}((id.Ev(v), pc_1), pc_1', \rho_{d,1}) = (\rho_{d,1}', v_{d,1}, E_{d,1}')$

(IV.17) $d_{d,1}' = \mathsf{update}(d_{d,1}, v_{d,1})$

(IV.18) $\mathcal{R}_1' = (\rho_{d,1}', d_{d,1}')$

(IV.19) $E_1' = \mathsf{robust}(\Sigma_1, E_{d,1} :: E_{d,1}', pc_1)$

From (IV.7) and the definition of $\mathsf{downgrade}_{\mathcal{E}}$,

(IV.20) $E_{e,1} = ((id.Ev(v), (l_c, l_i)) \mid pc_1' \downarrow^i \sqsubseteq l_i \sqsubset pc_1 \downarrow^i \wedge l_c = pc_1 \downarrow^c \sqcup pc_1' \downarrow^c)$

(IV.21) $\mathcal{S}_1 = (\rho_{e,1}, d_{e,1})$

(IV.22) $\mathcal{E}((id.Ev(v), pc_1), pc_1', \rho_{e,1}) = (\rho_{e,1}', v_{e,1}, E_{e,1}')$

(IV.23) $d_{e,1}' = \mathsf{update}(d_{e,1}, v_{e,1})$

(IV.24) $\mathcal{S}_1' = (\rho_{e,1}', d_{e,1}')$

(IV.25) $E_1'' = \mathsf{transparent}(\Sigma_1, E_{e,1} :: E_{e,1}', pc_1)$

From (IV.8) and the definition of $\mathsf{downgrade}_{\mathcal{D},\mathcal{E}}$,

(IV.26) $E_{m,1} = \mathsf{mergeEvents}(E_{d,1} :: E_{d,1}', E_{e,1} :: E_{e,1}')$

(IV.27) $E_1''' = \mathsf{robustTransparent}(\Sigma_1, E_{m,1}, pc_1)$

Subcase i: $T_1 \downarrow_l^p = (id.Ev(v), pc_1)$

By assumption, $pc_1 = pc_2$ and $pc_1 \downarrow_l^p \sqsubseteq l$

Then, for $\Sigma_2(pc_2) = (\_, \sigma_2^{EH})$, $\sigma_1^{EH}(pc_1) = \sigma_2^{EH}(pc_2)$

From this, the rest of the proof is straightforward.

Subcase ii: $T_1 \downarrow_l^p = \mathsf{rls}(\ldots)$

The proof for this case is similar to **Subsubcase II.ii.b**

Subcase iii: $T_1 \downarrow_l^p = \mathsf{sntz}(\ldots)$

The proof for this case is similar to **Subcase ii**

Subcase iv: $T_1 \downarrow_l^p = \mathsf{down}(id.Ev(v), \tau_d, \tau_e, E_1''', pc_1)$

By assumption and from (2),

(iv.1) $T_2 \downarrow_l^p = \mathsf{down}(id.Ev(v), \tau_d, \tau_e, E_1''', pc_1)$

From (iv.1),

(iv.2) $pc_1 = pc_2$

From (iv.1) and the definition of $T \downarrow_l^p$,

(iv.3) $\mathcal{G}$ must end in In-DE

(iv.4) $\Sigma_2 = \Sigma_2'$

(iv.5) $\mathcal{P}(id.Ev(v)) = pc_2'$

(iv.6) $E_2 = ((id.Ev(v), pc_2'') \mid pc_2 \sqcup pc_2' \sqsubseteq pc_2'')$

(iv.7) $(\mathcal{R}_2', E_2') = \mathsf{downgrade}_{\mathcal{D}}(\mathcal{R}_2, \Sigma_2, (id.Ev(v), pc_2), pc_2')$

(iv.8) $(\mathcal{S}_2', E_2'') = \mathsf{downgrade}_{\mathcal{E}}(\mathcal{S}_2, \Sigma_2, (id.Ev(v), pc_2), pc_2')$

(iv.9) $E_2''' = \mathsf{downgrade}_{\mathcal{D},\mathcal{E}}(\mathcal{R}_2, \mathcal{S}_2, \Sigma_2, (id.Ev(v), pc_2), pc_2')$

(iv.10) $\Sigma_2, E_2 \rightsquigarrow \mathsf{ks}_2''$

(iv.11) $pc_2, \mathsf{r} \vdash \Sigma_2, E_2' \rightsquigarrow \mathsf{ks}_{d,2}$

(iv.12) $pc_2, \mathsf{t} \vdash \Sigma_2, E_2'' \rightsquigarrow \mathsf{ks}_{e,2}$

(iv.13) $pc_2, \mathsf{rt} \vdash \Sigma_2, E_2''' \rightsquigarrow \mathsf{ks}_{m,2}$

(iv.14) $\mathsf{ks}_2' = \mathsf{ks}_2'' :: \mathsf{ks}_{d,2} :: \mathsf{ks}_{e,2} :: \mathsf{ks}_{m,2}$

From (iv.7) and the definition of $\mathsf{downgrade}_{\mathcal{D}}$,

(iv.15) $E_{d,2} = ((id.Ev(v), (l_c, l_i)) \mid pc_2' \downarrow^c \sqsubseteq l_c \sqsubset pc_2 \downarrow^c \wedge l_i = pc_2 \downarrow^i \sqcup pc_2' \downarrow^i)$

(iv.16) $\mathcal{R}_2 = (\rho_{d,2}, d_{d,2})$

(iv.17) $\mathcal{D}((id.Ev(v), pc_2), pc_2', \rho_{d,2}) = (\rho_{d,2}', v_{d,2}, E_{d,2}')$

(iv.18) $d_{d,2}' = \mathsf{update}(d_{d,2}, v_{d,2})$

(iv.19) $\mathcal{R}_2' = (\rho_{d,2}', d_{d,2}')$

(iv.20) $E_2' = \mathsf{robust}(\Sigma_2, E_{d,2} :: E_{d,2}', pc_2)$

From (iv.8) and the definition of $\mathsf{downgrade}_{\mathcal{E}}$,

(iv.21) $E_{e,2} = ((id.Ev(v), (l_c, l_i)) \mid pc_2' \downarrow^i \sqsubseteq l_i \sqsubset pc_2 \downarrow^i \wedge l_c = pc_2 \downarrow^c \sqcup pc_2' \downarrow^c)$

(iv.22) $\mathcal{S}_2 = (\rho_{e,2}, d_{e,2})$

(iv.23) $\mathcal{E}((id.Ev(v), pc_2), pc_2', \rho_{e,2}) = (\rho_{e,2}', v_{e,2}, E_{e,2}')$

(iv.24) $d_{e,2}' = \mathsf{update}(d_{e,2}, v_{e,2})$

(iv.25) $\mathcal{S}_2' = (\rho_{e,2}', d_{e,2}')$

(iv.26) $E_2'' = \mathsf{transparent}(\Sigma_2, E_{e,2} :: E_{e,2}', pc_2)$

From (iv.9) and the definition of $\mathsf{downgrade}_{\mathcal{D},\mathcal{E}}$,

(iv.27) $E_{m,2} = \mathsf{mergeEvents}(E_{d,2} :: E_{d,2}', E_{e,2} :: E_{e,2}')$

(iv.28) $E_2''' = \mathsf{robustTransparent}(\Sigma_2, E_{m,2}, pc_2)$

From (IV.2) and (iv.5),

(iv.29) $pc_1' = pc_2'$

From (5), (6), (IV.15), (IV.21), (iv.16), and (iv.22),

(iv.30) $\rho_{d,1} = \rho_{d,2}$

(iv.31) $d_{d,1} = d_{d,2}$

(iv.32) $\rho_{e,1} = \rho_{e,2}$

(iv.33) $d_{e,1} = d_{e,2}$

From (IV.16), (IV.22), (iv.17), (iv.23), (iv.2), (iv.29), (iv.30), and (iv.32),

(iv.34) $\rho_{d,1}' = \rho_{d,2}'$

(iv.35) $v_{d,1} = v_{d,2}$

(iv.36) $E_{d,1}' = E_{d,2}'$

(iv.37) $\rho_{e,1}' = \rho_{e,2}'$

(iv.38) $v_{e,1} = v_{e,2}$

(iv.39) $E_{e,1}' = E_{e,2}'$

From (IV.17), (IV.23), (iv.18), (iv.24), (iv.31), (iv.33), (iv.35), and (iv.38),

(iv.40) $d_{d,1}' = d_{d,2}'$

(iv.41) $d_{e,1}' = d_{e,2}'$

From (IV.18), (IV.24), (iv.19), (iv.25), (iv.34), (iv.37), (iv.40), and (iv.41),

(iv.42) $\mathcal{R}_1' = \mathcal{R}_2'$

(iv.43) $\mathcal{S}_1' = \mathcal{S}_2'$
From (7), (IV.1), and (iv.4),
(iv.44) $\Sigma_1' \approx_l^p \Sigma_2'$
From (iv.2), (iv.29), (IV.3), and (iv.6),
(iv.45) $E_1 = E_2$
From (7), (iv.45), (IV.9), (iv.10), and Lemma 19,
(iv.46) $\mathrm{ks}_1'' \approx_l^p \mathrm{ks}_2''$
From (iv.2), (iv.29), (IV.14), (IV.20), (iv.15), and (iv.21),
(iv.47) $E_{d,1} = E_{d,2}$
(iv.48) $E_{e,1} = E_{e,2}$
From (7), (iv.47), (iv.36), (iv.48), (iv.39), (IV.10), (iv.11), (IV.11), (iv.12), and Lemma 20,
(iv.49) $\mathrm{ks}_{d,1} \approx_l^p \mathrm{ks}_{d,2}$
(iv.50) $\mathrm{ks}_{e,1} \approx_l^p \mathrm{ks}_{e,2}$
From (iv.1), (IV.27), (iv.28), and the definition of trDowngrade,
(iv.51) $E_1''' = E_2'''$
From (5), (iv.51), (IV.12), (iv.13), and Lemma 20,
(iv.52) $\mathrm{ks}_{m,1} \approx_l^p \mathrm{ks}_{m,2}$
From (IV.13), (iv.14), (iv.46), (iv.49), (iv.50), and (iv.52),
(iv.53) $\mathrm{ks}_1' \approx_l^p \mathrm{ks}_2'$
From (iv.42)-(iv.44) and (iv.53),
$K_1' \approx_l^p K_2'$

Case V: $\mathcal{F}$ ends in Out
By assumption,
(V.1) $\alpha_{l,1} = (ch(v), pc_1)$
(V.2) $\mathrm{ks}_1 = (\kappa_1, pc_{src,1}, pc_1) :: \mathrm{ks}_1''$
(V.3) $\mathcal{P}(ch) = pc_1$
(V.4) $\exists \mathcal{F}' :: pc_{src,1}, d_{d,1}, d_{e,1} \vdash \Sigma_1, \kappa_1 \xrightarrow{ch(v)} \Sigma_1', \kappa_1'''$
(V.5) $\mathrm{ks}_1' = \mathrm{ks}_1''' :: \mathrm{ks}_1''$
(V.6) $\mathcal{R}_1' = \mathcal{R}_1$
(V.7) $\mathcal{S}_1' = \mathcal{S}_1$
From (3), (V.1), and (V.3),
(V.8) $T_1 \downarrow_l^p = ch(v)$ with
(V.9) $pc_1 \downarrow^p \sqsubseteq l \vee \mathcal{P}(ch) \downarrow^p \sqsubseteq l$
From (V.3) and (V.9),
(V.10) $pc_1 \downarrow^p \sqsubseteq l$
From (2) and (V.8),
(V.11) $T_2 \downarrow_l^p = ch(v)$
From (V.11), and the definition of $T \downarrow_l^p$,
(V.12) $\alpha_{l,2} = (ch(v), pc_2)$ with
(V.13) $pc_2 \downarrow^p \sqsubseteq l \vee \mathcal{P}(ch) \downarrow^p \sqsubseteq l$
From (V.12) and the output rules,
(V.14) $\mathcal{G}$ ends in Out
From (V.14),
(V.15) $\mathcal{P}(ch) = pc_2$
(V.16) $\mathrm{ks}_2 = (\kappa_2, pc_{src,2}, pc_2) :: \mathrm{ks}_2''$
(V.17) $\exists \mathcal{G}' :: pc_{src,2}, d_{d,2}, d_{e,2}, \vdash \Sigma_2, \kappa_2, \xrightarrow{ch(v)} \Sigma_2', \kappa_2'''$
(V.18) $\mathrm{ks}_2' = \mathrm{ks}_2''' :: \mathrm{ks}_2''$
(V.19) $\mathcal{R}_2' = \mathcal{R}_2$
(V.20) $\mathcal{S}_2' = \mathcal{S}_2$
From (V.15) and (V.3),
(V.21) $pc_1 = pc_2$
From (V.10), (V.21), (V.2), (V.16), and (8),
(V.22) $(\kappa_1, pc_{src,1}, pc_1) = (\kappa_2, pc_{src,2}, pc_2)$

(V.23) $\mathrm{ks}_1'' \approx_l^p \mathrm{ks}_2''$
From (5) and (6),
(V.24) $d_{d,1} = d_{d,2}$
(V.25) $d_{e,1} = d_{e,2}$
From (V.4), (V.17), (V.22), (V.24), (V.25), (7), (V.10), (V.21), and Lemma 16
(V.26) $\Sigma_1' \approx_l^p \Sigma_2'$
(V.27) $\mathrm{ks}_1''' \approx_l^p \mathrm{ks}_2'''$
From (V.23), (V.27), (V.5), and (V.18),
(V.28) $\mathrm{ks}_1' \approx_l^p \mathrm{ks}_2'$
From (5), (6), (V.6), (V.7), (V.19), (V.20), (V.26), and (V.28),
$K_1' \approx_l^p K_2'$

Case VI: $\mathcal{F}$ ends in Out-Skip
By assumption,
(VI.1) $\alpha_{l,1} = (\bullet, pc_1)$
(VI.2) $\mathrm{ks}_1 = (\kappa_1, pc_{src,1}, pc_1) :: \mathrm{ks}_1''$
(VI.3) producer(κ_1)
(VI.4) $\mathcal{P}(ch) \neq pc_1$
(VI.5) $\exists \mathcal{F}' :: pc_{src,1}, d_{d,1}, d_{e,1} \vdash \Sigma_1, \kappa_1 \xrightarrow{ch(v)} \Sigma_1', \kappa_1'''$
(VI.6) $\mathrm{ks}_1' = \mathrm{ks}_1''' :: \mathrm{ks}_1''$
(VI.7) $\mathcal{R}_1' = \mathcal{R}_1$
(VI.8) $\mathcal{S}_1' = \mathcal{S}_1$
From (3), (VI.1), and (VI.3),
(VI.9) $T_1 \downarrow_l^p = \bullet$ with
(VI.10) $pc_1 \downarrow^p \sqsubseteq l$
From (2) and (VI.9),
(VI.11) $T_2 \downarrow_l^p = \bullet$
From (VI.11), and the definition of $T \downarrow_l^p$,
(VI.12) $\alpha_{l,2} = (\bullet, pc_2)$ with
(VI.13) $pc_2 \downarrow^p \sqsubseteq l$
From (8), (VI.10), and (VI.13),
(VI.14) $\mathrm{ks}_2 = (\kappa_2, pc_{src,2}, pc_2) :: \mathrm{ks}_2''$ with
(VI.15) $(\kappa_1, pc_{src,1}, pc_1) = (\kappa_2, pc_{src,2}, pc_2)$ with
(VI.16) $pc_1 = pc_2$
(VI.17) $\mathrm{ks}_1'' \approx_l^p \mathrm{ks}_2''$
From (VI.15) and (VI.3),
(VI.18) producer(κ_2)
From (VI.12), (VI.18), and the output rules,
(VI.19) $\mathcal{G}$ ends in Out-Skip or Out-Silent
From (VI.19),
(VI.20) $\exists \mathcal{G}' :: pc_{src,2}, d_{d,2}, d_{e,2}, \vdash \Sigma_2, \kappa_2, \xrightarrow{\alpha} \Sigma_2', \kappa_2'''$
(VI.21) $\mathrm{ks}_2' = \mathrm{ks}_2''' :: \mathrm{ks}_2''$
(VI.22) $\mathcal{R}_2' = \mathcal{R}_2$
(VI.23) $\mathcal{S}_2' = \mathcal{S}_2$
From (5) and (6),
(VI.24) $d_{d,1} = d_{d,2}$
(VI.25) $d_{e,1} = d_{e,2}$
From (VI.5), (VI.20), (VI.15), (VI.24), (VI.25), (7), (VI.10), (VI.16), and Lemma 16,
(VI.26) $\Sigma_1' \approx_l^p \Sigma_2'$
(VI.27) $\mathrm{ks}_1''' \approx_l^p \mathrm{ks}_2'''$
From (VI.17), (VI.27), (VI.6), and (VI.21),
(VI.28) $\mathrm{ks}_1' \approx_l^p \mathrm{ks}_2'$
From (5), (6), (VI.7), (VI.8), (VI.22), (VI.23), (VI.26), and (VI.28),

$K_1' \approx_l^p K_2'$

Case VII: $\mathcal{F}$ ends in OUT-SILENT and $T_1 \downarrow_l^p \notin \{\mathsf{t}(\_), \mathsf{r}(\_)\}$
The proof for this case is similar to **Case V**

Case VIII: $\mathcal{F}$ ends in OUT-SILENT and $T_1 \downarrow_l^p = \in \{\mathsf{t}(\_), \mathsf{r}(\_)\}$
Without loss of generality, assume $T_1 \downarrow_l^p = \mathsf{r}(id, pc_1)$. The proof for the other cases are similar. The most important difference is that when $T_1 \downarrow_l^p = \mathsf{r}(id, eh, pc_1)$, then we also have $pc_{id,1} \downarrow^i \sqsubseteq pc_1 \downarrow^i$ and $pc_{id,2} \downarrow^i \sqsubseteq pc_1 \downarrow^i$
In general:
$pc_{src,1} \downarrow^p \sqsubseteq pc_1 \downarrow^p$
$pc_{src,2} \downarrow^p \sqsubseteq pc_2 \downarrow^p$ and
$pc_{id,1} \downarrow^p \sqsubseteq pc_1 \downarrow^p$
$pc_{id,2} \downarrow^p \sqsubseteq pc_2 \downarrow^p$
(which is the premise for Lemma 18)

By assumption,
(VIII.1) $\alpha_{l,1} = (\mathsf{new}(id, pc_{src,1}), pc_1)$
(VIII.2) $\mathsf{ks}_1 = (\kappa_1, pc_{src,1}, pc_1) :: \mathsf{ks}_1''$
(VIII.3) $\mathsf{producer}(\kappa_1)$
(VIII.4) $\exists \mathcal{F}' :: pc_{src,1}, d_{d,1}, d_{e,1} \vdash \Sigma_1, \kappa_1 \xrightarrow{\mathsf{new}(id, pc_{src,1})} \Sigma_1', \kappa_1'''$
(VIII.5) $\mathsf{ks}_1' = \mathsf{ks}_1''' :: \mathsf{ks}_1''$
(VIII.6) $\mathcal{R}_1' = \mathcal{R}_1$
(VI II.7) $\mathcal{S}_1' = \mathcal{S}_1$
By assumption and from (3) and (VIII.1),
(VIII.8) $pc_1 \downarrow^c \not\sqsubseteq l$
(VIII.9) $pc_{src,1} \downarrow^i \sqsubseteq pc_1 \downarrow^i$
By assumption and from (2),
(VIII.10) $T_2 \downarrow^c = \mathsf{r}(id, pc_2)$
(VIII.11) $pc_1 = pc_2$
From (VIII.10), (VIII.11), and the definition of $T \downarrow^p$,
(VIII.12) $\alpha_{l,2} = (\mathsf{new}(id, pc_{src,2}), pc_2)$ with
(VIII.13) $pc_2 \downarrow^c \not\sqsubseteq l$
(VIII.14) $pc_{src,2} \downarrow^i \sqsubseteq pc_2 \downarrow^i$
Since NEW is the only rule to produce $\alpha = \mathsf{new}(\_)$,
(VIII.15) $\mathsf{ks}_2 = (\kappa_2, pc_{src,2}, pc_2) :: \mathsf{ks}_2''$
(VIII.16) $\kappa_1 = (\_, \mathsf{new}(id, e_1), \_, \_)$
(VIII.17) $\kappa_2 = (\_, \mathsf{new}(id, e_2), \_, \_)$
(VIII.18) $\mathsf{ks}_1'' \approx_l^p \mathsf{ks}_2''$
From (VIII.17),
(VIII.19) $\mathsf{producer}(\kappa_2)$
From (VIII.12), (VIII.19), and the output rules,
(VIII.20) $\mathcal{G}$ ends in OUT-SILENT
From (VIII.20),
(VIII.21) $\exists \mathcal{G}' :: pc_{src,2}, d_{d,2}, d_{e,2}, \vdash \Sigma_2, \kappa_2, \xrightarrow{\mathsf{new}(id, pc_{src,2})} \Sigma_2', \kappa_2'''$
(VIII.22) $\mathsf{ks}_2' = \mathsf{ks}_2''' :: \mathsf{ks}_2''$
(VIII.23) $\mathcal{R}_2' = \mathcal{R}_2$
(VIII.24) $\mathcal{S}_2' = \mathcal{S}_2$
From (VIII.4), (VIII.21), (VIII.16), (VIII.17), (7), (VIII.8), (VIII.9), (VIII.13), (VIII.14), (VIII.11), and Lemma 18,
(VIII.25) $\Sigma_1' \approx_l^p \Sigma_2'$
(VIII.26) $\mathsf{ks}_1''' \approx_l^p \mathsf{ks}_2'''$
From (VIII.18), (VIII.26), (VIII.5), and (VIII.22),
(VIII.27) $\mathsf{ks}_1' \approx_l^p \mathsf{ks}_2'$
From (5), (6), (VIII.6), (VIII.7), (VIII.23)-(VIII.25), and (VIII.27)

$K_1' \approx_l^p K_2'$

Case IX: $\mathcal{F}$ ends in Out-Next

By assumption,

(IX.1) $\alpha_{l,1} = (\bullet, pc_1)$

(IX.2) $\mathsf{ks}_1 = (\kappa_1, pc_{src,1}, pc_1) :: \mathsf{ks}_1'$

(IX.3) $\mathsf{consumer}(\kappa_1)$

(IX.4) $\mathcal{R}_1' = \mathcal{R}_1$

(IX.5) $\mathcal{S}_1' = \mathcal{S}_1$

(IX.6) $\Sigma_1' = \Sigma_1$

From (3) and (IX.1),

(IX.7) $T_1 \downarrow_l^p = \bullet$ with

(IX.8) $pc_1 \downarrow^p \sqsubseteq l$

From (2) and (IX.7),

(IX.9) $T_2 \downarrow_l^p = \bullet$

From (IX.9), and the definition of $T \downarrow_l^p$,

(IX.10) $\alpha_{l,2} = (\bullet, pc_2)$ with

(IX.11) $pc_2 \downarrow^p \sqsubseteq l$

From (8), (IX.8), and (IX.11),

(IX.12) $\mathsf{ks}_2 = (\kappa_2, pc_{src,2}, pc_2) :: \mathsf{ks}_2''$ with

(IX.13) $(\kappa_1, pc_{src,1}, pc_1) = (\kappa_2, pc_{src,2}, pc_2)$ with

(IX.14) $pc_1 = pc_2$

(IX.15) $\mathsf{ks}_1' \approx_l^p \mathsf{ks}_2''$

From (IX.13) and (IX.3),

(IX.16) $\mathsf{consumer}(\kappa_2)$

From (IX.12), (IX.16), and the output rules,

(IX.17) $\mathcal{G}$ ends in Out-Next

From (IX.17),

(IX.18) $\mathcal{R}_2' = \mathcal{R}_2$

(IX.19) $\mathcal{S}_2' = \mathcal{S}_2$

(IX.20) $\Sigma_2' = \Sigma_2$

(IX.21) $\mathsf{ks}_2' = \mathsf{ks}_2''$

From (IX.15) and (IX.21),

(IX.22) $\mathsf{ks}_1' \approx_l^p \mathsf{ks}_2'$

From (5)-(7), (IX.4)-(IX.6), (IX.18)-(IX.20), and (IX.22),

$K_1' \approx_l^p K_2'$

□

Lemma 16. If $pc_{src}, d_d, d_e \vdash \Sigma_1, \kappa \xrightarrow{\alpha_1}_{pc} \Sigma_1', \mathsf{ks}_1$ and $pc_{src}, d_d, d_e \vdash \Sigma_2, \kappa \xrightarrow{\alpha_2}_{pc} \Sigma_2', \mathsf{ks}_2$, with $\Sigma_1 \approx_l^p \Sigma_2$ and $pc \downarrow^p \sqsubseteq l$, then $\Sigma_1' \approx_l^p \Sigma_2'$ and $\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$

Proof.

We examine each case of $\mathcal{F} :: pc_{src}, d_d, d_e \vdash \Sigma_1, \kappa \xrightarrow{\alpha_1}_{pc} \Sigma_1', \mathsf{ks}_1$

Denote $\mathcal{G} :: pc_{src}, d_d, d_e \vdash \Sigma_2, \kappa \xrightarrow{\alpha_2}_{pc} \Sigma_2', \mathsf{ks}_2$

By assumption,

(1) $\Sigma_1 \approx_l^p \Sigma_2$

(2) $pc \downarrow^p \not\sqsubseteq l$

Case I: $\mathcal{F}$ ends in PtoC

By assumption,

(I.1) $\kappa = \sigma, \mathsf{skip}, P, \cdot$

(I.2) $\mathsf{ks}_1 = ((\sigma, \mathsf{skip}, C, \cdot), pc_{src}, pc)$

(I.3) $\Sigma_1' = \Sigma_1$

From (I.1),

(I.4) $\mathcal{G}$ ends in PtoC

From (I.4),
(I.5) $\mathsf{ks}_2 = ((\sigma, \mathsf{skip}, C, \cdot), pc_{src}, pc)$
(I.6) $\Sigma_2' = \Sigma_2$
From (I.2) and (I.5),
$\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$
From (1), (I.3), and (I.6),
$\Sigma_1' \approx_l^p \Sigma_2'$

Case II: $\mathcal{F}$ ends in PtoLC
By assumption,
(II.1) $\kappa = \sigma, \mathsf{skip}, P, E$ with
(II.2) $E \neq \cdot$
(II.3) $\Sigma_1, E \rightsquigarrow \mathsf{ks}_1'$
(II.4) $\mathsf{ks}_1 = ((\sigma, \mathsf{skip}, C, \cdot), pc_{src}, pc) :: \mathsf{ks}_1'$
(II.5) $\Sigma_1' = \Sigma_1$
From (II.1) and (II.2),
(II.6) $\mathcal{G}$ ends in PtoC
From (II.6),
(II.7) $\Sigma_2, E \rightsquigarrow \mathsf{ks}_2'$
(II.8) $\mathsf{ks}_2 = ((\sigma, \mathsf{skip}, C, \cdot), pc_{src}, pc) :: \mathsf{ks}_2'$
(II.9) $\Sigma_2' = \Sigma_2$
From (1), (II.3), (II.7), and Lemma 19,
(II.10) $\mathsf{ks}_1' \approx_l^p \mathsf{ks}_2'$
From (II.4), (II.5), and (II.10),
$\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$
From (1), (II.5), and (II.9),
$\Sigma_1' \approx_l^p \Sigma_2'$

Case III: $\mathcal{F}$ ends in P
By assumption,
(III.1) $\kappa = \sigma, c, P, E$
(III.2) $\exists \mathcal{F}' :: pc_{src}, d_d, d_e \vdash \Sigma_1, \sigma, c \xrightarrow{\alpha_1}_{pc} \Sigma_1', \sigma_1, c_1, E_1$
(III.3) $\mathsf{ks}_1 = ((\sigma_1, c_1, P, E :: E_1), pc_{src}, pc)$
From (III.2) and our operational semantics for commands,
(III.4) $c \neq \mathsf{skip}$
From (III.4),
(III.5) $\exists \mathcal{F}' :: pc_{src}, d_d, d_e \vdash \Sigma_2, \sigma, c \xrightarrow{\alpha_2}_{pc} \Sigma_2', \sigma_2, c_2, E_2$
From (III.5),
(III.6) $\mathcal{G}$ ends in P
From (III.6),
(III.7) $\mathsf{ks}_2 = ((\sigma_2, c_2, P, E :: E_2), pc_{src}, pc)$
From (1), (2), (III.2), (III.5), and Lemma 17
(III.8) $\sigma_1 = \sigma_2$
(III.9) $c_1 = c_2$
(III.10) $E_1 = E_2$
$\Sigma_1' \approx_l^p \Sigma_2'$
From (III.3), (III.7), and (III.8)-(III.10),
$\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$

□

Lemma 17. If $pc_{src}, d_d, d_e \vdash \Sigma_1, \sigma, c \xrightarrow{\alpha_1}_{pc} \Sigma_1', \sigma_1, c_1, E_1$ and $pc_{src}, d_d, d_e \vdash \Sigma_2, \sigma, c \xrightarrow{\alpha_2}_{pc} \Sigma_2', \sigma_2, c_2, E_2$, with $\Sigma_1 \approx_l^p \Sigma_2$ and $pc \downarrow^p \sqsubseteq l$, then $\Sigma_1' \approx_l^p \Sigma_2'$, $\sigma_1 = \sigma_2$, $c_1 = c_2$, and $E_1 = E_2$

Proof.
By induction on the structure of $\mathcal{F} :: pc_{src}, d_d, d_e \vdash \Sigma_1, \sigma, c \xrightarrow{\alpha_1}_{pc} \Sigma_1', \sigma_1, c_1, E_1$

and $\mathcal{G} :: pc_{src}, d_d, d_e \vdash \Sigma_2, \sigma, c \xrightarrow{\alpha_2}_{pc} \Sigma_2', \sigma_2, c_2, E_2$
By assumption,
(1) $\Sigma_1 \approx_l^p \Sigma_2$
(2) $pc \downarrow^p \not\sqsubseteq l$

Case I: $\mathcal{F}$ ends in SKIP
By assumption,
(I.1) $c = \text{skip}; c'$
(I.2) $c_1 = c'$
(I.3) $\Sigma_1' = \Sigma_1$
(I.4) $\sigma_1 = \sigma$
(I.5) $E_1 = \cdot$
From (I.1),
(I.6) $\mathcal{G}$ ends in SKIP
From (I.6),
(I.7) $c_2 = c'$
(I.8) $\Sigma_2' = \Sigma_2$
(I.9) $\sigma_2 = \sigma$
(I.10) $E_2 = \cdot$
From (1), (I.2)-(I.5), and (I.7)-(I.10),
$c_1 = c_2$, $\Sigma_1' \approx_l^p \Sigma_2'$, $\sigma_1 = \sigma_2$, and $E_1 = E_2$

Case II: $\mathcal{F}$ ends in SEQ
By assumption,
(II.1) $c = c_1'; c_2'$
(II.2) $\exists \mathcal{F}' :: pc_{src}, d_d, d_e \vdash \Sigma_1, \sigma, c_1' \xrightarrow{\alpha_1}_{pc} \Sigma_1', \sigma_1, c_1'', E_1$
(II.3) $c_1 = c_1''; c_2'$
From (II.1),
(II.4) $\exists \mathcal{G}' :: pc_{src}, d_d, d_e \vdash \Sigma_2, \sigma, c_1' \xrightarrow{\alpha_2}_{pc} \Sigma_2', \sigma_2, c_2'', E_2$
(II.5) $c_2 = c_2''; c_2'$
By IH on $\mathcal{F}'$ and $\mathcal{G}'$,
(II.6) $c_1'' = c_2''$
$\Sigma_1' \approx_l^p \Sigma_2'$, $\sigma_1 = \sigma_2$, and $E_1 = E_2$
From (II.3), (II.5), and (II.6),
$c_1 = c_2$

Case III: $\mathcal{F}$ ends in ASSIGN-L or ASSIGN-G
We assume $\mathcal{F}$ ends in ASSIGN-G; the proof for the other case is similar
By assumption and for $\Sigma_1(pc) = (\sigma_1^g, \_)$,
(III.1) $c = x := e$
(III.2) $x \notin \sigma_1^g$
(III.3) $[\![e]\!]_{\sigma, \Sigma_1}^{pc} = v_1$
(III.4) $c_1 = \text{skip}$
(III.5) $\Sigma_1' = \Sigma_1$
(III.6) $\sigma_1 = \sigma[x \mapsto v_1]$
(III.7) $E_1 = \cdot$
From (1) and (2),
(III.8) $\Sigma_1(pc) = \Sigma_2(pc)$
From (III.2), (III.8), and for $\Sigma_2(pc) = (\sigma_2^g, \_)$,
(III.9) $x \notin \sigma_2^g$
From (III.1) and (III.9),
(III.10) $\mathcal{G}$ ends in ASSIGN-L
From (III.10),
(III.11) $c_2 = \text{skip}$
(III.12) $[\![e]\!]_{\sigma, \Sigma_2}^{pc} = v_2$

(III.13) $\Sigma_2' = \Sigma_2$
(III.14) $\sigma_2 = \sigma[x \mapsto v_2]$
(III.15) $E_2 = \cdot$
From (III.8), (III.3), and (III.12),
(III.16) $v_1 = v_2$
From (1), (III.5), and (III.13),
$\Sigma_1' \approx_l^p \Sigma_2'$
From (III.6), (III.14), and (III.16),
$\sigma_1 = \sigma_2$
From (III.4), (III.7), (III.11), and (III.15),
$c_1 = c_2$ and $E_1 = E_2$

Case IV: $\mathcal{F}$ ends in UPDATE
By assumption,
(IV.1) $c = id := e$
(IV.2) $[\![e]\!]^{pc}_{\sigma,\Sigma_1} = v_1$
(IV.3) $c_1 = \mathsf{skip}$
(IV.4) $\sigma_1 = \sigma$
(IV.5) $\Sigma_1(pc) = (\sigma^g, \sigma^{EH})$
(IV.6) $\sigma^{EH}(id) = (v, M, pc_{id})$
(IV.7) $\sigma_1^{EH} = \sigma^{EH}[id \mapsto (v_1, M, pc_{id})]$
(IV.8) $\Sigma_1' = \Sigma_1[pc \mapsto (\sigma^g, \sigma_1^{EH})]$
(IV.9) $E_1 = \cdot$
From (IV.1),
(IV.10) $\mathcal{G}$ ends in UPDATE
From (1) and (2),
(IV.11) $\Sigma_1(pc) = \Sigma_2(pc)$
From (IV.11) and (IV.5),
(IV.12) $\Sigma_2(pc) = (\sigma^g, \sigma^{EH})$
From (IV.10) and (IV.12),
(IV.13) $[\![e]\!]^{pc}_{\sigma,\Sigma_2} = v_2$
(IV.14) $c_2 = \mathsf{skip}$
(IV.15) $\sigma_2 = \sigma$
(IV.16) $\sigma_2^{EH} = \sigma^{EH}[id \mapsto (v_2, M, pc_{id})]$
(IV.17) $\Sigma_2' = \Sigma_2[pc \mapsto (\sigma^g, \sigma_2^{EH})]$
(IV.18) $E_2 = \cdot$
From (IV.11), (IV.2), and (IV.13),
(IV.19) $v_1 = v_2$
From (IV.19), (IV.7), and (IV.16),
(IV.20) $\sigma_1^{EH} = \sigma_2^{EH}$
From (IV.8), (IV.17), and (IV.20),
$\Sigma_1' \approx_l^p \Sigma_2'$
From (IV.4), (IV.15), (IV.3), (IV.14), (IV.9), and (IV.18),
$\sigma_1 = \sigma_2$, $c_1 = c_2$, and $E_1 = E_2$

Case V: $\mathcal{F}$ ends in IF-TRUE, IF-FALSE, WHILE-TRUE, or WHILE-FALSE
We assume $\mathcal{F}$ ends in IF-TRUE; the proofs for the other cases are similar
By assumption,
(V.1) $c = \mathsf{if}\ e\ \mathsf{then} c_1'\ \mathsf{else}\ c_2'$
(V.2) $[\![e]\!]^{pc}_{\sigma,\Sigma_1} = \mathsf{true}$
(V.3) $c_1 = c_1'$
(V.4) $\Sigma_1' = \Sigma_1$
(V.5) $\sigma_1 = \sigma$
(V.6) $E_1 = \cdot$
From (1) and (2),

(V.7) $\Sigma_1(pc) = \Sigma_2(pc)$
From (V.2) and (V.7),
(V.8) $[\![e]\!]^{pc}_{\sigma,\Sigma_2} = \mathsf{true}$
From (V.1) and (V.8),
(V.9) $\mathcal{G}$ ends in IF-TRUE
From (V.9),
(V.10) $c_2 = c_1'$
(V.11) $\Sigma_2' = \Sigma_2$
(V.12) $\sigma_2 = \sigma$
(V.13) $E_2 = \cdot$
From (1), (V.4), (V.11), (V.3), (V.10), (V.5), (V.12), (V.6), and (V.13),
$\Sigma_1' \approx^p_l \Sigma_2'$, $\sigma_1 = \sigma_2$, $c_1 = c_2$, and $E_1 = E_2$

Case VI: $\mathcal{F}$ ends in EVENT-TRIGGER
By assumption,
(VI.1) $c = \mathsf{triggger}(id.Ev(e))$
(VI.2) $[\![e]\!]^{pc}_{\sigma,\Sigma_1} = v_1$
(VI.3) $c_1 = \mathsf{skip}$
(VI.4) $\Sigma_1' = \Sigma_1$
(VI.5) $\sigma_1 = \sigma$
(VI.6) $E_1 = (id.Ev(v_1), pc)$
From (VI.1),
(VI.7) $\mathcal{G}$ ends in EVENT-TRIGGER
From (VI.7),
(VI.8) $[\![e]\!]^{pc}_{\sigma,\Sigma_2} = v_2$
(VI.9) $c_2 = \mathsf{skip}$
(VI.10) $\Sigma_2' = \Sigma_2$
(VI.11) $\sigma_2 = \sigma$
(VI.12) $E_2 = (id.Ev(v_2), pc)$
From (1) and (2),
(VI.13) $\Sigma_1(pc) = \Sigma_2(pc)$
From (VI.2), (VI.8), and (VI.13),
(VI.14) $v_1 = v_2$
From (1), (VI.4), (VI.10), (VI.3), (VI.9), (VI.5), (VI.11), (VI.6), (VI.12), and (VI.14),
$\Sigma_1' \approx^p_l \Sigma_2'$, $\sigma_1 = \sigma_2$, $c_1 = c_2$, and $E_1 = E_2$

Case VII: $\mathcal{F}$ ends in NEW or ADD-EH
We assume $\mathcal{F}$ ends in NEW; the proof for the other case is similar
By assumption,
(VII.1) $c = \mathsf{new}(id, e)$
(VII.2) $[\![e]\!]^{pc}_{\sigma,\Sigma_1} = v_1$
(VII.3) $c_1 = \mathsf{skip}$
(VII.4) $\sigma_1 = \sigma$
(VII.5) $\Sigma_1(pc) = (\sigma^g, \sigma^{EH})$
(VII.6) $\sigma_1^{EH} = \sigma^{EH}[id \mapsto (v, \cdot, pc_{src})]$
(VII.7) $\Sigma_1' = \Sigma_1[pc \mapsto (\sigma^g, \sigma_1^{EH})]$
(VII.8) $E_1 = \cdot$
From (VII.1),
(VII.9) $\mathcal{G}$ ends in NEW
From (1) and (2),
(VII.10) $\Sigma_1(pc) = \Sigma_2(pc)$
From (VII.10) and (VII.5),
(VII.11) $\Sigma_2(pc) = (\sigma^g, \sigma^{EH})$
From (VII.9) and (VII.11),
(VII.12) $[\![e]\!]^{pc}_{\sigma,\Sigma_2} = v_2$

(VII.13) $c_2 = \mathsf{skip}$
(VII.14) $\sigma_2 = \sigma$
(VII.15) $\sigma_2^{EH} = \sigma^{EH}[id \mapsto (v_2, \cdot, pc_{src})]$
(VII.16) $\Sigma_2' = \Sigma_2[pc \mapsto (\sigma^g, \sigma_2^{EH})]$
(VII.17) $E_2 = \cdot$
From (VII.10), (VII.2), and (VII.12),
(VII.18) $v_1 = v_2$
From (VII.18), (VII.6), and (VII.15),
(VII.19) $\sigma_1^{EH} = \sigma_2^{EH}$
From (VII.7), (VII.16), and (VII.19),
$\Sigma_1' \approx_l^p \Sigma_2'$
From (VII.4), (VII.14), (VII.3), (VII.13), (VII.8), and (VII.17),
$\sigma_1 = \sigma_2$, $c_1 = c_2$, and $E_1 = E_2$

Case VIII: $\mathcal{F}$ ends in DECLASSIFY or ENDORSE
We assume $\mathcal{F}$ ends in DECLASSIFY; the proof for the other case is similar
By assumption,
(VIII.1) $c = x := \mathsf{declassify}(\iota, e)$
(VIII.2) $\mathsf{read}(d, \iota) = v$
(VIII.3) $c_1 = x := v$
(VIII.4) $\sigma_1 = \sigma$
(VIII.5) $\Sigma_1' = \Sigma_1$
(VIII.6) $E_1 = \cdot$
From (VIII.1),
(VIII.7) $\mathcal{G}$ ends in NEW
From (VIII.7) and (VIII.2),
(VIII.8) $c_2 = x := v$
(VIII.9) $\sigma_2 = \sigma$
(VIII.10) $\Sigma_2' = \Sigma_2$
(VIII.11) $E_2 = \cdot$
From (1), (VIII.5), (VIII.10), (VIII.4), (VIII.9), (VIII.3), (VIII.8), (VIII.6), and (VIII.11),
$\Sigma_1' \approx_l^p \Sigma_2'$, $\sigma_1 = \sigma_2$, $c_1 = c_2$, and $E_1 = E_2$

□

Lemma 18. If $pc_{src,1}, d_{d,1}, d_{e,1} \vdash \Sigma_1, \kappa_1 \xrightarrow{\alpha_1}_{pc} \Sigma_1', \mathsf{ks}_1'$ and $pc_{src,2}, d_{d,2}, d_{e,2} \vdash \Sigma_2, \kappa_2 \xrightarrow{\alpha_2}_{pc} \Sigma_2', \mathsf{ks}_2'$, with $\alpha_1 = \mathsf{new}(id, pc_{src,1})$ and $\alpha_2 = \mathsf{new}(id, pc_{src,2})$, or $\alpha_1 = \mathsf{newEH}(id, eh, pc_{id,1}, pc_{src,1})$ and $\alpha_2 = \mathsf{newEH}(id, eh, pc_{id,2}, pc_{src,2})$ $\Sigma_1 \approx_l^p \Sigma_2$, $pc \downarrow^p \not\sqsubseteq l$, and $pc_{src,1} \downarrow^p \sqsubseteq pc_1 \downarrow^p$ $pc_{src,2} \downarrow^p \sqsubseteq pc_2 \downarrow^p$ and $pc_{id,1} \downarrow^p \sqsubseteq pc_1 \downarrow^p$ $pc_{id,2} \downarrow^p \sqsubseteq pc_2 \downarrow^p$, then $\Sigma_1' \approx_l^p \Sigma_2'$ and $\mathsf{ks}_1' \approx_l^p \mathsf{ks}_2'$

PROOF.
We examine each case of $\mathcal{F} :: pc_{src,1}, d_{d,1}, d_{e,1} \vdash \Sigma_1, \kappa_1 \xrightarrow{\alpha_1}_{pc} \Sigma_1', \mathsf{ks}_1'$
Denote $\mathcal{G} :: pc_{src,2}, d_{d,2}, d_{e,2} \vdash \Sigma_2, \kappa_2 \xrightarrow{\alpha_2}_{pc} \Sigma_2', \mathsf{ks}_2'$
By assumption,
(1) $\Sigma_1 \approx_l^p \Sigma_2$
(2) $pc \downarrow^p \not\sqsubseteq l$
(3) $\alpha_1 = \mathsf{new}(id, pc_{src,1})$ and $\alpha_2 = \mathsf{new}(id, pc_{src,2})$ or
(4) $\alpha_1 = \mathsf{newEH}(id, eh, pc_{id,1}, pc_{src,1})$ and $\alpha_2 = \mathsf{newEH}(id, eh, pc_{id,2}, pc_{src,2})$
(5) $pc_{src,1} \downarrow^p \sqsubseteq pc_1 \downarrow^p$ and $pc_{src,2} \downarrow^p \sqsubseteq pc_2 \downarrow^p$
(6) $pc_{id,1} \downarrow^p \sqsubseteq pc_1 \downarrow^p$ and $pc_{id,2} \downarrow^p \sqsubseteq pc_2 \downarrow^p$
From (3) and (4) and since only P could produce new(_) or newEH(_),
(7) $\mathcal{F}$ and $\mathcal{G}$ must end in P
From (7),
(8) $\exists \mathcal{F}' :: pc_{src,1}, d_{d,1}, d_{e,1} \vdash \Sigma_1, \sigma_1, c_1 \xrightarrow{\alpha_1}_{pc} \Sigma_1', \sigma_1', c_1', E_1$
(9) $c_1 \in \{\mathsf{new}(id, e_1), \mathsf{addEH}(id, eh)\}$
(10) $\mathsf{ks}_1' = ((\sigma_1', c_1', P, E :: E_1), pc_{src,1}, pc)$
(11) $\exists \mathcal{G}' :: pc_{src,2}, d_{d,2}, d_{e,2} \vdash \Sigma_2, \sigma_2, c_2 \xrightarrow{\alpha_2}_{pc} \Sigma_2', \sigma_2', c_2', E_2$

(12) $c_2 \in \{\mathsf{new}(id, e_2), \mathsf{addEH}(id, eh)\}$
(13) $\mathsf{ks}_2' = ((\sigma_2', c_2, P, E :: E_2), pc_{src,2}, pc)$
From (2), (10), and (13),
$\mathsf{ks}_1' \approx_l^p \mathsf{ks}_2'$
From (8), (9), (11), (12),
(14) $\mathcal{F}'$ and $\mathcal{G}'$ end in NEW or ADD-EH

Case I: $\mathcal{F}'$ and $\mathcal{G}'$ end in NEW
By assumption and from $\Sigma_1(pc) = (\_, \sigma_1^{EH})$ and $\Sigma_2(pc) = (\_, \sigma_2^{EH})$,
(I.1) $\sigma_1^{EH'} = \sigma_1^{EH}[id \mapsto (\_, \cdot, pc_{src,1})]$
(I.2) $\sigma_2^{EH'} = \sigma_2^{EH}[id \mapsto (\_, \cdot, pc_{src,2})]$
(I.3) $\Sigma_1' = \Sigma_1[pc \mapsto (\_, \sigma_1^{EH'})]$
(I.4) $\Sigma_2' = \Sigma_2[pc \mapsto (\_, \sigma_2^{EH'})]$
From (1) and (2),
(I.5) $\Sigma_1 \approx_l^p \Sigma_2$
(I.6) $\sigma_1^{EH} \approx_l^p \sigma_2^{EH}$
From (I.6), (I.1), (I.2), and (5),
(I.7) $\sigma_1^{EH'} \approx_l^p \sigma_2^{EH'}$
From (I.5), (I.3), (I.4), (2), and (I.7),
$\Sigma_1' \approx_l^p \Sigma_2'$

Case II: $\mathcal{F}'$ and $\mathcal{G}'$ end in ADD-EH
By assumption and from $\Sigma_1(pc) = (\_, \sigma_1^{EH})$, $\Sigma_2(pc) = (\_, \sigma_2^{EH})$, and $eh = \mathsf{on}Ev(x)\{c\}$
(II.1) $\sigma_1^{EH}(id) = (\_, M_1, pc_{id,1})$
(II.2) $M_1(Ev) = EH_1$
(II.3) $M_1' = M_1[Ev \mapsto EH_1 \cup \{eh, pc_{src,1}\}]$
(II.4) $\sigma_1^{EH'} = \sigma_1^{EH}[id \mapsto (\_, M_1', pc_{id,1})]$
(II.5) $\Sigma_1' = \Sigma_1[pc \mapsto (\_, \sigma_1^{EH'})]$
(II.6) $\sigma_2^{EH}(id) = (\_, M_2, pc_{id,2})$
(II.7) $M_2(Ev) = EH_2$
(II.8) $M_2' = M_2[Ev \mapsto EH_2 \cup \{eh, pc_{src,2}\}]$
(II.9) $\sigma_2^{EH'} = \sigma_2^{EH}[id \mapsto (\_, M_2', pc_{id,2})]$
(II.10) $\Sigma_2' = \Sigma_2[pc \mapsto (\_, \sigma_2^{EH'})]$
From (1) and (2),
(II.11) $\Sigma_1 \approx_l^p \Sigma_2$
(II.12) $\sigma_1^{EH} \approx_l^p \sigma_2^{EH}$
From (II.12), (II.1), (II.2), (II.6), (II.7), and (6),
(II.13) $M_1 \downarrow_l^p = M_2 \downarrow_l^p$
(II.14) $EH_1 \downarrow_l^p = EH_2 \downarrow_l^p$
From (II.13), (II.14), (II.3), (III.8), and (5),
(II.15) $M_1' \downarrow_l^p = M_2' \downarrow_l^p$
From (II.15), (II.4), (II.9), and (6),
(II.16) $\sigma_1^{EH'} \approx_l^p \sigma_2^{EH'}$
From (II.11), (II.5), (II.10), (2), and (II.16),
$\Sigma_1' \approx_l^p \Sigma_2'$

□

Lemma 19 (Equivalent State, Equivalent Event Handlers). If $\Sigma_1 \approx_l^p \Sigma_2$ and $E_1 \approx_l^p E_2$, with $\Sigma_1, E_1 \rightsquigarrow \mathsf{ks}_1$, $\Sigma_2, E_2 \rightsquigarrow \mathsf{ks}_2$ then $\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$

PROOF.
By induction on $\mathcal{F} :: \Sigma_1, E_1 \rightsquigarrow \mathsf{ks}_1$ and $\mathcal{G} :: \Sigma_2, E_2 \rightsquigarrow \mathsf{ks}_2$
By assumption,
(1) $\Sigma_1 \approx_l^p \Sigma_2$
(2) $E_1 \approx_l^p E_2$

Case I: $\mathcal{F}$ ends in LOOKUP

By assumption,

(I.1) $E_1 = (id.Ev(v), pc) :: E_1'$

(I.2) $\Sigma_1(pc) = \_, \sigma^{EH}$

(I.3) $\sigma^{EH}(id) = (\_, M, pc_{id})$

(I.4) $pc, pc_{id}, v \vdash M(Ev) \rightsquigarrow \mathsf{ks}_1'$

(I.5) $\exists \mathcal{F}' :: \Sigma_1, E_1' \rightsquigarrow \mathsf{ks}_1''$

(I.6) $\mathsf{ks}_1 = \mathsf{ks}_1' :: \mathsf{ks}_1''$

Subcase i: $pc \downarrow^p \sqsubseteq l$

By assumption and from (2) and the definition of $\approx_l^p$ for E,

(i.1) $E_2 = (id.Ev(v), pc) :: E_2'$ with

(i.2) $E_1' \approx_l^p E_2'$

By assumption and from (1),

(i.3) $\Sigma_1(pc) = \Sigma_2(pc)$

By assumption and from (i.3), (I.3), and (I.4),

(i.4) $\mathcal{G}$ ends in LOOKUP

From (I.2) and (i.3),

(i.5) $\Sigma_2(pc) = \_, \sigma^{EH}$

From (i.4), (i.5), and (I.3),

(i.6) $pc, pc_{id}, v \vdash M(Ev) \rightsquigarrow \mathsf{ks}_2'$

(i.7) $\exists \mathcal{G}' :: \Sigma_2, E_2' \rightsquigarrow \mathsf{ks}_2''$

(i.8) $\mathsf{ks}_2 = \mathsf{ks}_2' :: \mathsf{ks}_2''$

From (I.4) and (i.6),

(i.9) $\mathsf{ks}_1' = \mathsf{ks}_2'$

From (i.2), (I.5), (i.7), and IH on $\mathcal{F}'$ and $\mathcal{G}'$,

(i.10) $\mathsf{ks}_1'' \approx_l^p \mathsf{ks}_2''$

From (I.6) and (i.8)-(i.10),

$\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$

Subcase ii: $pc \not\sqsubseteq l$

By assumption,

(ii.1) $E_1 \approx_l^p E_1'$

By assumption and from (I.4) and Lemma 14,

(ii.2) $\mathsf{ks}_1' \approx_l^p \cdot$

From (2) and (ii.1),

(ii.3) $E_1' \approx_l^p E_2$

IH on $\mathcal{F}'$ and $\mathcal{G}$ gives

(ii.4) $\mathsf{ks}_1'' \approx_l^p \mathsf{ks}_2$

From (ii.2), (ii.4), and (I.6),

$\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$

Case II: $\mathcal{F}$ ends in LOOKUP-MISSING

By assumption,

(II.1) $E_1 = (id.Ev(v), pc) :: E_1'$

(II.2) $\Sigma_1(pc) = \_, \sigma^{EH}$

(II.3) $id \notin \sigma^{EH}$ or $\sigma^{EH}(id) = (\_, M, \_)$ with $Ev \notin M$

(II.4) $\exists \mathcal{F}' :: \Sigma_1, E_1' \rightsquigarrow \mathsf{ks}_1$

Subcase i: $pc \downarrow^p \sqsubseteq l$

By assumption and from (2) and the definition of $\approx_l^p$ for E,

(i.1) $E_2 = (id.Ev(v), pc) :: E_2'$ with

(i.2) $E_1' \approx_l^p E_2'$

By assumption and from (1),

(i.3) $\Sigma_1(pc) = \Sigma_2(pc)$

By assumption and from (II.3) and (i.3),

(i.4) $\mathcal{G}$ ends in LOOKUP-MISSING

From (i.4),

(i.5) $\exists \mathcal{G}' :: \Sigma_2, E_2' \rightsquigarrow \mathsf{ks}_2$

From (i.2) and IH on $\mathcal{F}'$ and $\mathcal{G}'$,

$\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$

Subcase ii: $pc \downarrow^p \not\sqsubseteq l$

By assumption,

(ii.1) $E_1 \approx_l^p E_1'$

From (2) and (ii.1),

(ii.2) $E_1' \approx_l^p E_2$

IH on $\mathcal{F}'$ and $\mathcal{G}$ gives

$\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$

Case II: $\mathcal{F}$ ends in LOOKUP-EMPTY

By assumption,

(II.1) $E_1 = \cdot$

(II.2) $\mathsf{ks}_1 = \cdot$

From (II.1) and (2),

(II.3) $E_2 \downarrow_l^p = \cdot$

From (II.3) and Lemma 13,

(II.4) $\mathsf{ks}_2 \approx_l^p \cdot$

From (II.2) and (II.4),

$\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$

□

Lemma 20. If $\Sigma_1 \approx_l^p \Sigma_2$ *and* $E_1 \approx_l^p E_2$*, with* $pc_{Ev}, \mathsf{f} \vdash \Sigma_1, E_1 \rightsquigarrow \mathsf{ks}_1$*,* $pc_{Ev}, \mathsf{f} \vdash \Sigma_2, E_2 \rightsquigarrow \mathsf{ks}_2$ *and* $\mathsf{f} \in \{\mathsf{r}, \mathsf{t}, \mathsf{rt}\}$ *then* $\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$

PROOF.

By induction on $\mathcal{F} :: pc_{Ev}, \mathsf{f} \vdash \Sigma_1, E_1 \rightsquigarrow \mathsf{ks}_1$ and $\mathcal{G} :: pc_{Ev}, \mathsf{f} \vdash \Sigma_2, E_2 \rightsquigarrow \mathsf{ks}_2$ for $\mathsf{f} \in \{\mathsf{r}, \mathsf{t}, \mathsf{rt}\}$

By assumption,

(1) $\Sigma_1 \approx_l^p \Sigma_2$

(2) $E_1 \approx_l^p E_2$

Case I: $\mathcal{F}$ ends in LOOKUP-R

The proofs for LOOKUP-T and LOOKUP-RT are similar.

By assumption,

(I.1) $E_1 = (id.Ev(v), pc) :: E_1'$

(I.2) $\Sigma_1(pc) = \_, \sigma^{EH}$

(I.3) $\sigma^{EH}(id) = (\_, M, pc_{id})$

(I.4) $M(Ev) \downarrow_{l'}^i = EH \neq \cdot$ for $l' = pc_{Ev} \downarrow^i$

(I.5) $pc, pc_{id}, v \vdash EH \rightsquigarrow \mathsf{ks}_1'$

(I.6) $\exists \mathcal{F}' :: pc_{Ev}, \mathsf{r} \vdash \Sigma_1, E_1' \rightsquigarrow \mathsf{ks}_1''$

(I.7) $\mathsf{ks}_1 = \mathsf{ks}_1' :: \mathsf{ks}_1''$

Subcase i: $pc \downarrow^p \sqsubseteq l$

By assumption and from (2) and the definition of $\approx_l^p$ for E,

(i.1) $E_2 = (id.Ev(v), pc) :: E_2'$ with

(i.2) $E_1' \approx_l^p E_2'$

By assumption and from (1),

(i.3) $\Sigma_1(pc) = \Sigma_2(pc)$

By assumption and from (i.3), (I.3), and (I.4),

(i.4) $\mathcal{G}$ ends in LOOKUP

From (I.2) and (i.3),

(i.5) $\Sigma_2(pc) = \_, \sigma^{EH}$
From (i.4), (i.5), (I.3), and (I.4),
(i.6) $pc, pc_{id}, v \vdash EH \rightsquigarrow \mathsf{ks}_2'$
(i.7) $\exists \mathcal{G}' :: pc_{Ev}, \mathsf{r} \vdash \Sigma_2, E_2' \rightsquigarrow \mathsf{ks}_2''$
(i.8) $\mathsf{ks}_2 = \mathsf{ks}_2' :: \mathsf{ks}_2''$
From (I.5) and (i.6),
(i.9) $\mathsf{ks}_1' = \mathsf{ks}_2'$
From (i.2), (I.6), (i.7), and IH on $\mathcal{F}'$ and $\mathcal{G}'$,
(i.10) $\mathsf{ks}_1'' \approx_l^p \mathsf{ks}_2''$
From (I.7) and (i.8)-(i.10),
$\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$

Subcase ii: $pc \not\sqsubseteq l$
By assumption,
(ii.1) $E_1 \approx_l^p E_1'$
By assumption and from (I.5) and Lemma 14,
(ii.2) $\mathsf{ks}_1' \approx_l^p \cdot$
From (2) and (ii.1),
(ii.3) $E_1' \approx_l^p E_2$
IH on $\mathcal{F}'$ and $\mathcal{G}$ gives
(ii.4) $\mathsf{ks}_1'' \approx_l^p \mathsf{ks}_2$
From (ii.2), (ii.4), and (I.7),
$\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$

Case II: $\mathcal{F}$ ends in LOOKUP-NOTR
By assumption,
(II.1) $E_1 = (id.Ev(v), pc) :: E_1'$
(II.2) $\Sigma_1(pc) = \_, \sigma^{EH}$ and $\sigma^{EH}(id) = (\_, M, \_)$
(II.3) $Ev \notin M$ or $M(Ev) \downarrow_{l'}^{i} = \cdot$ for $l' = pc_{Ev} \downarrow^{i}$
(II.4) $\exists \mathcal{F}' :: pc_{Ev}, \mathsf{r} \vdash \Sigma_1, E_1' \rightsquigarrow \mathsf{ks}_1$

Subcase i: $pc \downarrow^{p} \sqsubseteq l$
By assumption and from (2) and the definition of $\approx_l^p$ for E,
(i.1) $E_2 = (id.Ev(v), pc) :: E_2'$ with
(i.2) $E_1' \approx_l^p E_2'$
By assumption and from (1),
(i.3) $\Sigma_1(pc) = \Sigma_2(pc)$ and $\sigma^{EH}(id) = (\_, M, \_)$
By assumption and from (II.3) and (i.3),
(i.4) $\mathcal{G}$ ends in LOOKUP-NOTR
From (i.4),
(i.5) $\exists \mathcal{G}' :: pc_{Ev}, \mathsf{r} \vdash \Sigma_2, E_2' \rightsquigarrow \mathsf{ks}_2$
From (i.2) and IH on $\mathcal{F}'$ and $\mathcal{G}'$,
$\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$

Subcase ii: $pc \downarrow^{p} \not\sqsubseteq l$
By assumption,
(ii.1) $E_1 \approx_l^p E_1'$
From (2) and (ii.1),
(ii.2) $E_1' \approx_l^p E_2$
IH on $\mathcal{F}'$ and $\mathcal{G}$ gives
$\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$

Case II: $\mathcal{F}$ ends in LOOKUP-R-EMP
By assumption,

(II.1) $E_1 = \cdot$
(II.2) $\mathsf{ks}_1 = \cdot$
From (II.1) and (2),
(II.3) $E_2 \downarrow^p_l = \cdot$
From (II.3) and Lemma 21,
(II.4) $\mathsf{ks}_2 \approx^p_l \cdot$
From (II.2) and (II.4),
$\mathsf{ks}_1 \approx^p_l \mathsf{ks}_2$

□

Lemma 21. If $pc, \mathsf{f} \vdash \Sigma, E \rightsquigarrow \mathsf{ks}$ with $E \downarrow^p_l = \cdot$ and $\mathsf{f} \in \{\mathsf{r}, \mathsf{t}, \mathsf{rt}\}$, then $\mathsf{ks} \approx^p_l \cdot$

Proof.
By induction on the structure of $\mathcal{F} :: pc, \mathsf{f} \vdash \Sigma, E \rightsquigarrow \mathsf{ks}$ for $\mathsf{f} \in \{\mathsf{r}, \mathsf{t}, \mathsf{rt}\}$

By assumption,
(1) $E \downarrow^p = \cdot$

Case I: $\mathcal{F}$ ends in lookup-R
The proofs for lookup-T and lookup-RT are similar to this case
By assumption,
(I.1) $E = (id.Ev(v), pc) :: E'$
(I.2) $\Sigma(pc) = (\_, \sigma^{EH})$ and $\sigma^{EH}(id) = (\_, M, pc_{id})$
(I.3) $l_{Ev} = pc_{Ev} \downarrow^i$ and $M(Ev) \downarrow^i_{l_{Ev}} = EH \neq \cdot$
(I.4) $\exists \mathcal{G} :: pc, pc_{id}, v \vdash EH \rightsquigarrow \mathsf{ks}_1$
(I.5) $\exists \mathcal{G}' :: pc_{Ev}, \mathsf{r} \vdash \Sigma, E' \rightsquigarrow \mathsf{ks}_2$
(I.6) $\mathsf{ks} = \mathsf{ks}_1 :: \mathsf{ks}_2$
From (1) and (I.1),
(I.6) $pc \downarrow^p \not\sqsubseteq l$
(I.7) $E' \downarrow^p_l = \cdot$
From (I.6), (I.3) and Lemma 14,
(I.8) $\mathsf{ks}_1 \approx^p_l \cdot$
From (I.4), (I.7) and IH on $\mathcal{G}$,
(I.9) $\mathsf{ks}_2 \approx^p_l \cdot$
From (I.5), (I.8), and (I.9),
$\mathsf{ks} \approx^p_l \cdot$

Case II: $\mathcal{F}$ ends in lookup-notR
The proofs for lookup-notR and lookup-notRT are similar to this case
By assumption,
(II.1) $E = (id.Ev(v), pc) :: E'$
(II.2) $\exists \mathcal{G} :: pc_{Ev}, \mathsf{r} \vdash \Sigma, E' \rightsquigarrow \mathsf{ks}$
From (1) and (II.1),
(II.3) $E' \downarrow^p_l = \cdot$
From (II.3), (II.2) and IH on $\mathcal{G}$,
$\mathsf{ks} \approx^p_l \cdot$

Case III: $\mathcal{F}$ ends in lookup-R-emp
The proofs for lookup-T-emp and lookup-RT-emp are similar to this case
By assumption, $\mathsf{ks} = \cdot$

□

Lemma 22 (Strong One-step). If $K_1 \approx^p_l K_2$, $T_1 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \overset{\alpha_{l,1}}{\Longrightarrow} K_1'$ with $T_1 \downarrow^p_l = \tau \neq \cdot$ and $\mathsf{prog}(K_2)$, with $\neg\mathsf{rlsA}(T_1), \mathsf{trnsprntA}(T_1, l), \mathsf{transparentT}(K_2, \tau, l)$ if $p = c$, $\neg\mathsf{sntzA}(T_1), \mathsf{rbstA}(T_1, l), \mathsf{robustT}(K_2, \tau, l)$ if $p = i$, and $T_1 \downarrow^p_l \notin \{\mathsf{t}(\_), \mathsf{r}(\_)\}$, then $\exists K_2', T_2$ s.t. $T_2 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_2'$ with $T_1 \approx^p_l T_2$ and $K_1' \approx^p_l K_2'$

Proof.

We examine each case of $\mathcal{F} :: T_1 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \overset{\alpha_{I,1}}{\Longrightarrow} K_1'$

By assumption,

(1) $T_1 \downarrow_l^p = \tau \neq \cdot$

(2) $K_1 \approx_l^p K_2$

(3) $\mathsf{prog}(K_2)$

(4) $\neg\mathsf{rlsA}(T_1), \mathsf{trnsprntA}(T_1, l), \mathsf{transparentT}(K_2, \tau, l)$ if $p = c$

(5) $\neg\mathsf{sntzA}(T_1), \mathsf{rbstA}(T_1, l), \mathsf{robustT}(K_2, \tau, l)$ if $p = i$

(6) $T_1 \downarrow_l^p \notin \{\mathsf{t}(\_), \mathsf{r}(\_)\}$

Case I: $\mathcal{F}$ ends in In

By assumption and from (4) and (5),

(I.1) $\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_C$ with

(I.2) $\mathsf{consumer}(K_C)$

(I.3) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_C) \downarrow_l^p = \cdot$

From (I.2),

(I.4) $\mathsf{ks}_C = \cdot$

From (I.1), (I.4), and Lemma 10,

(I.5) $K_2 \approx_l^p K_C$

From (2) and (I.5),

(I.6) $K_1 \approx_l^p K_C$

From (I.6),

(I.7) $\mathcal{R}_1 = \mathcal{R}_C$

(I.8) $\mathcal{S}_1 = \mathcal{S}_C$

(I.9) $\Sigma_1 \approx_l^p \Sigma_C$

By assumption,

(I.10) $\mathcal{R}_1' = \mathcal{R}_1$

(I.11) $\mathcal{S}_1' = \mathcal{S}_1$

(I.12) $\Sigma_1' = \Sigma_1$

(I.13) $\alpha_1 = (id.Ev(v), pc)$

(I.14) $\mathcal{P}(id.Ev(v)) = pc'$

(I.15) $\Sigma_1(pc) = (\_, \sigma_1^{EH})$

(I.16) $\sigma_1^{EH}(id) \downarrow^i \not\sqsubseteq pc \downarrow^i$

(I.17) $\sigma_1^{EH}(id) \downarrow^c \not\sqsubseteq pc \downarrow^c$

(I.18) $E_1 = ((id.Ev(v), pc'') \mid (pc \sqcup pc' \sqsubseteq pc''))$

(I.19) $\Sigma_1, E_1 \rightsquigarrow \mathsf{ks}_1'$

From (1), (I.13)-(I.17), and the definition of $T \downarrow_l^p$,

(I.20) $T_1 \downarrow_l^p = (id.Ev(v), pc)$ and

(I.21) $pc \downarrow^p \sqcup pc' \downarrow^p \sqsubseteq l$

From (I.21),

(I.22) $pc \downarrow^p \sqsubseteq l$

(I.23) $pc' \downarrow^p \sqsubseteq l$

From (I.9), (I.22), and (I.15)-(I.17),

(I.24) $\Sigma_C(pc) = (\_, \sigma_C^{EH})$

(I.25) $\sigma_C^{EH}(id) \downarrow^i \not\sqsubseteq pc \downarrow^i$

(I.26) $\sigma_C^{EH}(id) \downarrow^c \not\sqsubseteq pc \downarrow^c$

From (I.4) and (I.24)-(I.26),

(I.27) In may be applied to $\mathcal{R}_C, \mathcal{S}_C; \Sigma_C; \mathsf{ks}_C$ with input $(id.Ev(v), pc)$

From (I.27),

(I.28) $\exists K_2'$ s.t. $\mathcal{G} :: T_2 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_C \overset{(id.Ev(v), pc)}{\Longrightarrow} K_2'$

(I.29) $\mathcal{R}_2' = \mathcal{R}_C$

(I.30) $\mathcal{S}_2' = \mathcal{S}_C$

(I.31) $\Sigma_2' = \Sigma_C$

(I.32) $E_2 = ((id.Ev(v), pc'') \mid (pc \sqcup pc' \sqsubseteq pc''))$

(I.33) $\Sigma_C, E_2 \rightsquigarrow \mathsf{ks}'_2$
From (I.18), and (I.32),
(I.34) $E_1 = E_2$
From (I.9), (I.34), (I.19), (I.33), and Lemma 19,
(I.35) $\mathsf{ks}'_1 \approx^p_l \mathsf{ks}'_2$
From (I.13), (I.28), (I.20), (I.21), and the definition of $T \downarrow^p$,
(I.36) $T_2 \downarrow^p_l = (id.Ev(v), pc)$
From (I.20) and (I.36),
$T_1 \approx^p_l T_2$
From (I.7)-(I.9), (I.10)-(I.12), and (I.29)-(I.31),
(I.37) $\mathcal{R}'_1 = \mathcal{R}'_2$, $\mathcal{S}'_1 = \mathcal{S}'_2$, and $\Sigma'_1 \approx^p_l \Sigma'_2$
From (I.37) and (I.35),
$K'_1 \approx^p_l K'_2$

Case II: $\mathcal{F}$ ends in In-D
By assumption and from (4) and (5),
(II.1) $\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_C$ with
(II.2) $\mathsf{consumer}(K_C)$
(II.3) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_C) \downarrow^p_l = \cdot$
From (II.2),
(II.4) $\mathsf{ks}_C = \cdot$
From (II.1), (II.4), and Lemma 10,
(II.5) $K_2 \approx^p_l K_C$
From (2) and (II.5),
(II.6) $K_1 \approx^p_l K_C$
From (II.6),
(II.7) $\mathcal{R}_1 = \mathcal{R}_C$
(II.8) $\mathcal{S}_1 = \mathcal{S}_C$
(II.9) $\Sigma_1 \approx^p_l \Sigma_C$
By assumption,
(II.10) $\mathcal{S}'_1 = \mathcal{S}_1$
(II.11) $\Sigma'_1 = \Sigma_1$
(II.12) $\alpha_1 = (id.Ev(v), pc)$
(II.13) $\mathcal{P}(id.Ev(v)) = pc'$
(II.14) $\Sigma_1(pc) = (\_, \sigma^{EH}_1)$
(II.15) $\sigma^{EH}_1(id) \downarrow^i \sqsubseteq pc \downarrow^i$
(II.16) $\sigma^{EH}_1(id) \downarrow^c \not\sqsubseteq pc \downarrow^c$
(II.17) $E_1 = ((id.Ev(v), pc'') \mid (pc \sqcup pc' \sqsubseteq pc''))$
(II.18) $\mathsf{downgrade}_{\mathcal{D}}(\mathcal{R}_1, \Sigma_1, (id.Ev(v), pc), pc') = (\mathcal{R}'_1, E'_1)$
(II.19) $\Sigma_1, E_1 \rightsquigarrow \mathsf{ks}''_1$
(II.20) $pc, \mathsf{r} \vdash \Sigma_1, E'_1 \rightsquigarrow \mathsf{ks}'''_1$
(II.21) $\mathsf{ks}'_1 = \mathsf{ks}''_1 :: \mathsf{ks}'''_1$
From (II.18) and the definition of $\mathsf{downgrade}_{\mathcal{D}}$,
(II.22) $\mathcal{R}_1 = (\rho_1, d_1)$
(II.23) $\mathcal{D}(id.Ev(v), pc, \rho_1) = (\rho'_1, v_1, E'_{d,1})$
(II.24) $d'_1 = \mathsf{update}(d_1, v_1)$
(II.25) $\mathcal{R}'_1 = (\rho'_1, d'_1)$
(II.26) $E_{d,1} = ((id.Ev(v), (l_c, l_i)) \mid pc' \downarrow^c \sqsubseteq l_c \sqsubset pc \downarrow^c \wedge l_i = pc \downarrow^i \sqcup pc' \downarrow^i)$
(II.27) $E'_1 = \mathsf{robust}(\Sigma_1, E_{d,1} :: E'_{d,1}, pc)$
From (1), (II.13)-(II.16), and the definition of $T \downarrow^p_l$,
(II.28) $T_1 \downarrow^p_l = (id.Ev(v), pc)$ or
(II.29) $T_1 \downarrow^p_l = \mathsf{rls}(id.Ev(v), \rho'_1, v_1, E''_1, pc)$

Subcase i: $T_1 \downarrow^p_l = (id.Ev(v), pc)$

From (II.28), (II.20), and (II.27),

(i.1) $pc \downarrow_l^p \sqsubseteq l$

(i.2) $\mathcal{D}(id.Ev(v), pc', \rho_1) = (\rho_1, \mathsf{none}, E'_{d,1})$

(i.3) $\mathsf{ks}_1''' \downarrow_l^p = \cdot$ if $p = c$ and $\mathsf{ks}_1''' = \cdot$ if $p = i$

From (II.24) and (i.2),

(i.4), $d_1' = d_1$

From (II.23), (i.2), (i.6), and (II.25),

(i.5) $\mathcal{R}_1' = \mathcal{R}_1$

From (II.9), (i.1), and (II.14)-(II.16),

(i.6) $\Sigma_C(pc) = (\_, \sigma_C^{EH})$

(i.7) $\sigma_C^{EH}(id) \downarrow^i \sqsubseteq pc \downarrow^i$

(i.8) $\sigma_C^{EH}(id) \downarrow^c \not\sqsubseteq pc \downarrow^c$

From (II.4) and (i.6)-(i.8),

(i.9) In-D may be applied to $\mathcal{R}_C, \mathcal{S}_C; \Sigma_C; \mathsf{ks}_C$ with input $(id.Ev(v), pc)$

From (i.9),

(i.10) $\exists K_2'$ s.t. $\mathcal{G} :: T_2 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_C \overset{(id.Ev(v), pc)}{\Longrightarrow} K_2'$

(i.11) $\mathcal{S}_2' = \mathcal{S}_C$

(i.12) $\Sigma_2' = \Sigma_C$

(i.13) $\mathcal{P}(id.Ev(v)) = pc_2'$

(i.14) $E_2 = ((id.Ev(v), pc'') \mid (pc \sqcup pc' \sqsubseteq pc''))$

(i.15) $\mathsf{downgrade}_{\mathcal{D}}(\mathcal{R}_2, \Sigma_2, (id.Ev(v), pc), pc') = (\mathcal{R}_2', E_{d,2})$

(i.16) $\Sigma_C, E_2 \rightsquigarrow \mathsf{ks}_2''$

(i.17) $pc, \mathsf{r} \vdash \Sigma_2, E_2' \rightsquigarrow \mathsf{ks}_2'''$

(i.18) $\mathsf{ks}_2' = \mathsf{ks}_2'' :: \mathsf{ks}_2'''$

From (i.16) and the definition of $\mathsf{downgrade}_{\mathcal{D}}$,

(i.19) $\mathcal{R}_C = (\rho_C, d_C)$

(i.20) $\mathcal{D}(id.Ev(v), pc, \rho_C) = (\rho_2', v_2, E'_{d,2})$

(i.21) $d_2' = \mathsf{update}(d_C, v_2)$

(i.22) $\mathcal{R}_2' = (\rho_2', d_2')$

(i.23) $E_{d,2} = ((id.Ev(v), (l_c, l_i)) \mid pc' \downarrow^c \sqsubseteq l_c \sqsubset pc \downarrow^c \wedge l_i = pc \downarrow^i \sqcup pc' \downarrow^i)$

(i.24) $E_2' = \mathsf{robust}(\Sigma_C, E_{d,2} :: E'_{d,2}, pc)$

From (II.7), (i.1), (II.22), and (i.19),

(i.25) $\mathcal{R}_1 = \mathcal{R}_C = (\rho_1, d_1) = (\rho_C, d_C)$

From (i.25), (i.2), and (i.20),

(i.26) $\mathcal{D}(id.Ev(v), pc, \rho_C) = (\rho_1, \mathsf{none}, E'_{d,1})$

From (i.25) and (i.26),

(i.27) $\mathcal{D}(id.Ev(v), pc, \rho_C) = (\rho_C, \mathsf{none}, E'_{d,1})$

From (i.21), (i.27), and (i.20),

(i.28) $d_2' = d_C$

From (II.26) and (i.23),

(i.29) $E_{d,1} = E_{d,2}$

From (i.27) and (i.20),

(i.30) $E'_{d,1} = E'_{d,2}$

From (i.1), (i.29), (i.30), (II.9), and Lemma 26,

(i.31) $E_1' \approx_l^p E_2'$ if $p = c$ and $E_1' = E_2'$ if $p = i$

From (II.9), (I.4), (i.31), (II.20), (i.17), and Lemma 20,

(i.32) $\mathsf{ks}_1''' \approx_l^p \mathsf{ks}_2'''$ if $p = c$

From (II.9), (i.1), (i.31), (II.20), (i.17), (II.27), (i.24), and Lemma 28,

(i.33) $\mathsf{ks}_1''' = \mathsf{ks}_2'''$ if $p = i$

From (i.27), (i.28), (i.1), (i.3), (i.32), (i.33), (II.13), (i.10), and (i.6)-(i.8), and the definition of $T \downarrow_l^p$,

(i.34) $T_2 \downarrow_l^p = (id.Ev(v), pc)$

From (II.28) and (i.34),

$T_1 \approx_l^p T_2$

From (i.22), (i.19), (i.27), and (i.28), (i.35) $\mathcal{R}_2' = \mathcal{R}_C$

From (II.7), (i.5), and (i.35),

(i.36) $\mathcal{R}_1' = \mathcal{R}_2'$

From (II.8), (II.10), and (i.11),

(i.37) $\mathcal{S}_1' = \mathcal{S}_2'$

From (II.9), (II.11), and (i.12),

(i.38) $\Sigma_1' \approx_l^p \Sigma_2'$

From (II.17) and (i.14),

(i.39) $E_1 = E_2$

From (II.9), (i.39), (II.19), (i.16), and Lemma 19,

(i.40) $\mathsf{ks}_1'' \approx_l^p \mathsf{ks}_2''$

From (II.21), (i.18), (i.40), (i.32), and (i.33),

(i.41) $\mathsf{ks}_1' \approx_l^p \mathsf{ks}_2'$

From (i.35)-(i.38) and (i.41),

$K_1' \approx_l^p K_2'$

Subcase ii: $T_1 \downarrow_l^p = \mathsf{rls}(id.Ev(v), \rho_1', v_1, E_1'', pc)$

From (II.29) and (4),

(ii.1) $p = i$

From (II.29) and (ii.1),

(ii.2) $pc \downarrow^p \sqsubseteq l$

From (2), (ii.2), and $\Sigma_2 = (\_, \sigma_2^{EH})$,

(ii.3) $\sigma_1^{EH} = \sigma_2^{EH}$

The rest of the proof for this case is similar to **Subcase i**

Case III: $\mathcal{F}$ ends in In-E

The proof is similar to **Case II**. It uses Lemma 27 instead of Lemma 26 and Lemma 29 instead of Lemma 28.

Case IV: $\mathcal{F}$ ends in In-DE

By assumption and from (4) and (5),

(IV.1) $\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_C$ with

(IV.2) $\mathsf{consumer}(K_C)$

(IV.3) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_C) \downarrow_l^p = \cdot$

From (IV.2),

(IV.4) $\mathsf{ks}_C = \cdot$

From (IV.1), (IV.4), and Lemma 10,

(IV.5) $K_2 \approx_l^p K_C$

From (2) and (IV.5),

(IV.6) $K_1 \approx_l^p K_C$

From (IV.6),

(IV.7) $\mathcal{R}_1 = \mathcal{R}_C$

(IV.8) $\mathcal{S}_1 = \mathcal{S}_C$

(IV.9) $\Sigma_1 \approx_l^p \Sigma_C$

By assumption,

(IV.10) $\Sigma_1' = \Sigma_1$

(IV.11) $\alpha_1 = (id.Ev(v), pc)$

(IV.12) $\mathcal{P}(id.Ev(v)) = pc'$

(IV.13) $\Sigma_1(pc) = (\_, \sigma_1^{EH})$

(IV.14) $\sigma_1^{EH}(id) \downarrow^i \sqsubseteq pc \downarrow^i$

(IV.15) $\sigma_1^{EH}(id) \downarrow^c \sqsubseteq pc \downarrow^c$

(IV.16) $E_1 = ((id.Ev(v), pc'') \mid (pc \sqcup pc' \sqsubseteq pc''))$

(IV.17) $\mathsf{downgrade}_{\mathcal{D}}(\mathcal{R}_1, \Sigma_1, (id.Ev(v), pc), pc') = (\mathcal{R}_1', E_1')$

(IV.18) $\mathsf{downgrade}_{\mathcal{E}}(\mathcal{S}_1, \Sigma_1, (id.Ev(v), pc), pc') = (\mathcal{S}_1', E_1'')$

(IV.19) $\mathsf{downgrade}_{\mathcal{D},\mathcal{E}}(\mathcal{R}_1, \mathcal{S}_1, \Sigma_1, (id.Ev(v), pc), pc') = E_1'''$

(IV.20) $\Sigma_1, E_1 \rightsquigarrow \mathsf{ks}_1''$

(IV.21) $pc, \mathsf{r} \vdash \Sigma_1, E_1' \rightsquigarrow \mathsf{ks}_{d,1}$
(IV.22) $pc, \mathsf{t} \vdash \Sigma_1, E_1'' \rightsquigarrow \mathsf{ks}_{e,1}$
(IV.23) $pc, \mathsf{rt} \vdash \Sigma_1, E_1''' \rightsquigarrow \mathsf{ks}_{m,1}$
(IV.24) $\mathsf{ks}_1' = \mathsf{ks}_1'' :: \mathsf{ks}_{d,1} :: \mathsf{ks}_{e,1} :: \mathsf{ks}_{m,1}$

From (IV.17) and the definition of $\mathsf{downgrade}_{\mathcal{D}}$,
(IV.25) $\mathcal{R}_1 = (\rho_{d,1}, d_{d,1})$
(IV.26) $\mathcal{D}(id.Ev(v), pc, \rho_{d,1}) = (\rho_{d,1}', v_{d,1}, E_{d,1}')$
(IV.27) $d_{d,1}' = \mathsf{update}(d_{d,1}, v_{d,1})$
(IV.28) $\mathcal{R}_{d,1}' = (\rho_{d,1}', d_{d,1}')$
(IV.29) $E_{d,1} = ((id.Ev(v), (l_c, l_i)) \mid pc' \downarrow^c \sqsubseteq l_c \sqsubset pc \downarrow^c \wedge l_i = pc \downarrow^i \sqcup pc' \downarrow^i)$
(IV.30) $E_1' = \mathsf{robust}(\Sigma_1, E_{d,1} :: E_{d,1}', pc)$

From (IV.18) and the definition of $\mathsf{downgrade}_{\mathcal{E}}$,
(IV.31) $\mathcal{S}_1 = (\rho_{e,1}, d_{e,1})$
(IV.32) $\mathcal{E}(id.Ev(v), pc, \rho_{e,1}) = (\rho_{e,1}', v_{e,1}, E_{e,1}')$
(IV.33) $d_{e,1}' = \mathsf{update}(d_{e,1}, v_{e,1})$
(IV.34) $\mathcal{S}_1' = (\rho_{e,1}', d_{e,1}')$
(IV.35) $E_{e,1} = ((id.Ev(v), (l_c, l_i)) \mid pc' \downarrow^i \sqsubseteq l_i \sqsubset pc \downarrow^i \wedge l_c = pc \downarrow^c \sqcup pc' \downarrow^c)$
(IV.36) $E_1' = \mathsf{transparent}(\Sigma_1, E_{e,1} :: E_{e,1}', pc)$

From (IV.19) and the definition of $\mathsf{downgrade}_{\mathcal{D},\mathcal{E}}$,
(IV.37) $E_{m,1} = \mathsf{mergeEvents}(E_{d,1} :: E_{d,1}', E_{e,1} :: E_{e,1}')$
(IV.38) $E_1''' = \mathsf{robustTransparent}(\Sigma_1, E_{m,1}, pc)$

Subcase i: $T_1 \downarrow_l^p = (id.Ev(v), pc)$
By assumption and from the definition of $\mathsf{trInput}$,
(i.1) $pc \downarrow_l^p \sqcup pc' \downarrow_l^p \sqsubseteq l$
From (i.1),
(i.2) $pc \downarrow^p \sqsubseteq l$
(i.3) $pc' \downarrow^p \sqsubseteq l$
From (2), (i.2), (i.3), and $\Sigma_2 = (\_, \sigma_2^{EH})$,
(i.4) $\sigma_1^{EH} = \sigma_2^{EH}$
The rest of the proof for this case is similar to **Subcase II.i**

Subcase ii: $T_1 \downarrow_l^p = \mathsf{rls}(\dots)$
If $pc \downarrow^p \sqsubseteq l$, then the proof is similar to **Subcase II.ii**

Subcase iii: $T_1 \downarrow_l^p = \mathsf{sntz}(\dots)$
The proof for this case is similar to **Subcase ii**

Subcase iv: $T_1 \downarrow_l^p = \mathsf{down}(\dots)$
By assumption,
(iv.1) $T_1 \downarrow_l^p = \mathsf{down}(\dots)$
But (iv.1) contradicts (4) when $p = c$ and (iv.1) contradicts (5) when $p = i$, so this case holds vacuously

Case V: $\mathcal{F}$ ends in Out
By assumption and from (4) and (5),
(V.1) $\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \overset{\tau'}{\Longrightarrow}{}^* K_l$ with
(V.2) $\mathsf{lowEH}(K_l)$
(V.3) $\forall (\alpha, pc) \in \tau', \alpha \in \{ch(\_), \bullet\} \wedge pc \downarrow^p \not\sqsubseteq l$
From (V.2),
(V.4) $\mathsf{ks}_l = (\kappa_l, pc_{src,l}, pc_l) :: \mathsf{ks}_l'$ with
(V.5) $pc_l \downarrow^p \sqsubseteq l$
From (V.1)-(V.3) and Lemma 25,
(V.6) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \overset{\tau}{\Longrightarrow}{}^* K_l) \downarrow_l^p = \cdot$

From (V.1), (V.6), and Lemma 10,
(V.7) $K_2 \approx_l^p K_l$
From (2) and (V.7),
(V.8) $K_1 \approx_l^p K_l$
From (V.8),
(V.9) $\mathcal{R}_1 = \mathcal{R}_l$
(V.10) $\mathcal{S}_1 = \mathcal{S}_l$
(V.11) $\Sigma_1 \approx_l^p \Sigma_l$
(V.12) $\mathrm{ks}_1 \approx_l^p \mathrm{ks}_l$
By assumption,
(V.13) $\mathcal{R}_1' = \mathcal{R}_1$
(V.14) $\mathcal{S}_1' = \mathcal{S}_1$
(V.15) $\alpha_{l,1} = (ch(v), pc_1)$
(V.16) $\mathcal{P}(ch) = pc_1$
(V.17) $\mathcal{R}_1 = (\rho_{d,1}, d_{d,1})$
(V.18) $\mathcal{S}_1 = (\rho_{e,1}, d_{e,1})$
(V.19) $\mathrm{ks}_1 = (\kappa_1, pc_{src,1}, pc_1) :: \mathrm{ks}_1''$
(V.20) $\mathrm{producer}(\kappa_1)$
(V.21) $\exists \mathcal{F}' :: pc_{src,1}, d_{d,1}, d_{e,1} \vdash \Sigma_1, \kappa_1 \xrightarrow{ch(v)}_{pc_1} \Sigma_1', \mathrm{ks}_1'''$
(V.22) $\mathrm{ks}_1' = \mathrm{ks}_1''' :: \mathrm{ks}_1''$
From (1), (V.15), and the definition of $\downarrow_l^p$ for T,
(V.23) $pc_1 \downarrow^p \sqsubseteq l$
From (V.23), (V.19), (V.5), (V.4), (V.12), and the definition of $\approx_l^p$ for ks,
(V.24) $pc_1 = pc_l$
(V.25) $(\kappa_1, pc_{src,1}, pc_1) = (\kappa_l, pc_{src,l}, pc_l)$
From (V.24), (V.23), (V.5), (V.9), (V.10), (V.17), and (V.18),
(V.27) $\mathcal{R}_l = (\rho_{d,l}, d_{d,l})$
(V.28) $\mathcal{S}_l = (\rho_{e,l}, d_{e,l})$ with
(V.29) $d_{d,1} = d_{d,l}$
(V.30) $d_{e,1} = d_{e,l}$
From (V.21), (V.25), (V.29), (V.30), (V.11), (V.24), (V.23), (V.5), and Lemma 23
(V.31) $\exists \mathcal{G}' :: pc_{src,l}, d_{d,l}, d_{e,l} \vdash \Sigma_l, \kappa_l \xrightarrow{ch(v)}_{pc_l} \Sigma_2', \mathrm{ks}_2''$
(V.32) $\Sigma_1' \approx_l^p \Sigma_2'$
(V.33) $\mathrm{ks}_1''' \approx_l^p \mathrm{ks}_2''$
From (V.20) and (V.25),
(V.34) $\mathrm{producer}(\kappa_2)$
From (V.31), (V.34), (V.16), and (V.24),
(V.35) Out may be applied to $\mathcal{R}_l, \mathcal{S}_l; \Sigma_l; \mathrm{ks}_l$, producing output $(ch(v), pc_l)$
From (V.35),
(V.36) $\mathcal{G} :: T_2 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_l \overset{(ch(v), pc_l)}{\Longrightarrow} K_2'$
(V.37) $\mathcal{R}_2' = \mathcal{R}_l$
(V.38) $\mathcal{S}_2' = \mathcal{S}_l$
(V.39) $\mathrm{ks}_2' = \mathrm{ks}_2'' :: \mathrm{ks}_l'$
From (V.23), (V.15), and (V.16),
(V.40) $T_1 \downarrow_l^p = ch(v)$
From (V.36), (V.6), (V.5) (V.24), and (V.16),
(V.41) $T_2 \downarrow_l^p = ch(v)$
From (V.40) and (V.41),
$T_1 \approx_l^p T_2$
From (V.22), (V.39), (V.33), (V.12), (V.19), (V.4), and (V.25),
(V.42) $\mathrm{ks}_1' \approx_l^p \mathrm{ks}_2'$
From (V.9), (V.13), (V.37), (V.10), (V.14), (V.38), (V.32), and (V.42),
$K_1' \approx_l^p K_2'$

Case VI: $\mathcal{F}$ ends in OUT-SKIP or OUT-SILENT

The proofs for theses cases are similar to **Case V**

Case VIi: $\mathcal{F}$ ends in OUT-NEXT

By assumption and from (4) and (5),

(VII.1) $\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \overset{\tau'}{\Longrightarrow}{}^* K_l$ with

(VII.2) $\mathsf{lowEH}(K_l)$

(VII.3) $\forall(\alpha, pc) \in \tau', \alpha \in \{ch(\_), \bullet\} \wedge pc \downarrow^p \not\sqsubseteq l$

From (VII.2),

(VII.4) $\mathsf{ks}_l = (\kappa_l, pc_{src,l}, pc_l) :: \mathsf{ks}'_l$ with

(VII.5) $pc_l \downarrow^p \sqsubseteq l$

From (VII.1)-(VII.3) and Lemma 25,

(VII.6) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \overset{\tau}{\Longrightarrow}{}^* K_l) \downarrow^p_l = \cdot$

From (VII.1), (VII.6), and Lemma 10,

(VII.7) $K_2 \approx^p_l K_l$

From (2) and (VII.7),

(VII.8) $K_1 \approx^p_l K_l$

From (VII.8),

(VII.9) $\mathcal{R}_1 = \mathcal{R}_l$

(VII.10) $\mathcal{S}_1 = \mathcal{S}_l$

(VII.11) $\Sigma_1 \approx^p_l \Sigma_l$

(VII.12) $\mathsf{ks}_1 \approx^p_l \mathsf{ks}_l$

By assumption,

(VII.13) $\mathcal{R}'_1 = \mathcal{R}_1$

(VII.14) $\mathcal{S}'_1 = \mathcal{S}_1$

(VII.15) $\Sigma'_1 = \Sigma_1$

(VII.16) $\alpha_{l,1} = (\bullet, pc_1)$

(VII.17) $\mathsf{ks}_1 = (\kappa_1, pc_{src,1}, pc_1) :: \mathsf{ks}'_1$

(VII.18) $\mathsf{consumer}(\kappa_1)$

From (1), (VII.16), and the definition of $\downarrow^p_l$ for T,

(VII.19) $pc_1 \downarrow^p \sqsubseteq l$

From (VII.19), (VII.17), (VII.5), (VII.4), (VII.12), and the definition of $\approx^p_l$ for ks,

(VII.20) $pc_1 = pc_2$

(VII.21) $(\kappa_1, pc_{src,1}, pc_1) = (\kappa_l, pc_{src,l}, pc_l)$

From (VII.18) and (VII.21),

(VII.22) $\mathsf{consumer}(\kappa_2)$

From (VII.22),

(VII.23) OUT-NEXT may be applied to $\mathcal{R}_l, \mathcal{S}_l; \Sigma_l; \mathsf{ks}_l$, producing output $(\bullet, pc_l)$

From (VII.23),

(VII.24) $\mathcal{G} :: T_2 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_l \overset{(\bullet, pc_l)}{\Longrightarrow} K'_2$ and

(VII.25) $\mathcal{R}'_2 = \mathcal{R}_l$

(VII.26) $\mathcal{S}'_2 = \mathcal{S}_l$

(VII.27) $\Sigma'_2 = \Sigma_l$

(VII.28) $\mathsf{ks}'_2 = \mathsf{ks}'_l$

From (VII.19) and (VII.16),

(VII.29) $T_1 \downarrow^p_l = \bullet$

From (VII.24), (VII.5), and (VII.6),

(VII.30) $T_2 \downarrow^p_l = \bullet$

From (VII.29) and (VII.30),

$T_1 \approx^p_l T_2$

From (VII.17), (VII.12), (VII.4), (VII.21), and (VII.28),

(VII.31) $\mathsf{ks}'_1 \approx^p_l \mathsf{ks}'_2$

From (VII.9), (VII.13), (VII.25), (VII.10), (VII.14), (VII.26), (VII.11), (VII.15), (VII.27), and (VII.31),

$K_1' \approx_l^p K_2'$

□

Lemma 23. If $\mathcal{F} :: pc_{src}, d_d, d_e \vdash \Sigma_1, \kappa \xrightarrow{\alpha}_{pc} \Sigma_1', \mathsf{ks}_1$ *with* $pc \downarrow^p \sqsubseteq l$ *and* $\Sigma_1 \approx_l^p \Sigma_2$*, then* $\exists \mathcal{G} :: pc_{src}, d_d, d_e \vdash \Sigma_2, \kappa \xrightarrow{\alpha}_{pc} \Sigma_2', \mathsf{ks}_2$ *with* $\Sigma_1' \approx_l^p \Sigma_2'$ *and* $\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$

Proof.

We examine each case of $\mathcal{F} :: pc_{src}, d_d, d_e \vdash \Sigma_1, \kappa \xrightarrow{\alpha}_{pc} \Sigma_1', \mathsf{ks}_1$

By assumption,

(1) $pc \downarrow^p \sqsubseteq l$

(2) $\Sigma_1 \approx_l^p \Sigma_2$

Case I $\mathcal{F}$ ends in PtoC

By assumption,

(I.1) $\kappa = \sigma, \mathsf{skip}, P, \cdot$

(I.2) $\alpha = \bullet$

(I.3) $\Sigma_1' = \Sigma_1$

(I.4) $\mathsf{ks}_1 = ((\sigma, \mathsf{skip}, C, \cdot), pc_{src}, pc)$

From (I.1),

(I.5) PtoC may be applied to Σ_2 and κ

From (I.5),

(I.6) $\exists \mathcal{G} :: pc_{src}, d_d, d_e \vdash \Sigma_2, \sigma, \mathsf{skip}, P, \cdot \xrightarrow{\bullet}_{pc} \Sigma_2, ((\sigma, \mathsf{skip}, C, \cdot), pc_{src}, pc)$

From (I.6),

(I.7) $\Sigma_2' = \Sigma_2$

(I.8) $\mathsf{ks}_2 = ((\sigma, \mathsf{skip}, C, \cdot), pc_{src}, pc)$

Form (2), (I.3), and (I.7),

$\Sigma_1' \approx_l^p \Sigma_2'$

From (1), (I.4), and (I.8),

$\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$

Case II: $\mathcal{F}$ ends in PtoLC

By assumption,

(II.1) $\kappa = \sigma, \mathsf{skip}, P, E$

(II.2) $E \neq \cdot$

(II.3) $\Sigma_1, E \rightsquigarrow \mathsf{ks}_1'$

(II.5) $\alpha = \bullet$

(II.6) $\Sigma_1' = \Sigma_1$

(II.7) $\mathsf{ks}_1 = ((\sigma, \mathsf{skip}, C, \cdot), pc_{src}, pc) :: \mathsf{ks}_1'$

From (II.1) and (II.2),

(II.8) PtoLC may be applied to Σ_2 and κ

From (II.8),

(II.9) $\exists \mathcal{G} :: pc_{src}, d_d, d_e \vdash \Sigma_2, \sigma, \mathsf{skip}, P, E \xrightarrow{\bullet}_{pc} \Sigma_2, ((\sigma, \mathsf{skip}, C, \cdot), pc_{src}, pc) :: \mathsf{ks}_2'$ for

(II.10) $\Sigma_2, E \rightsquigarrow \mathsf{ks}_2'$

From (II.9),

(II.11) $\Sigma_2' = \Sigma_2$

(II.12) $\mathsf{ks}_2 = ((\sigma, \mathsf{skip}, C, \cdot), pc_{src}, pc) :: \mathsf{ks}_2'$

From (2), (II.6), and (II.11),

$\Sigma_1' \approx_l^p \Sigma_2'$

From (2), (II.3), (II.10), and Lemma 19,

(II.13) $\mathsf{ks}_1' \approx_l^p \mathsf{ks}_2'$

From (II.7), (II.12), and (II.13),

$\mathsf{ks}_1 \approx_l^p \mathsf{ks}_2$

Case III: $\mathcal{F}$ ends in P

By assumption,

(III.1) $\kappa = \sigma, c, P, E$

(III.2) $\exists \mathcal{F}' :: pc_{src}, d_d, d_e \vdash \Sigma_1, \sigma, c \xrightarrow{\alpha}_{pc} \Sigma_1', \sigma_1, c_1, E_1$

(III.3) $\mathrm{ks}_1 = ((\sigma_1, c_1, P, E :: E_1), pc_{src}, pc)$

From (1), (2), (III.2), and Lemma 24,

(III.4) $\exists \mathcal{G}' :: pc_{src}, d_d, d_e \vdash \Sigma_2, \sigma, c \xrightarrow{\alpha}_{pc} \Sigma_2'', \sigma_2, c_2, E_2$ with

(III.5) $\Sigma_1' \approx_l^p \Sigma_2''$

(III.6) $\sigma_1 = \sigma_2$

(III.7) $c_1 = c_2$

(III.8) $E_1 = E_2$

From (III.4),

(III.9) P may be applied to Σ_2 and κ

From (III.9),

(III.10) $\exists \mathcal{G} :: pc_{src}, d_d, d_e \vdash \Sigma_2, \sigma, c, P, E \xrightarrow{\bullet}_{pc} \Sigma_2'', ((\sigma_2, c_2, P, E :: E_2), pc_{src}, pc)$

From (III.10),

(III.11) $\Sigma_2' = \Sigma_2''$

(III.12) $\mathrm{ks}_2 = ((\sigma_2, c_2, P, E :: E_2), pc_{src}, pc)$

From (III.5) and (III.11),

$\Sigma_1' \approx_l^p \Sigma_2'$

From (III.3), (III.12), (III.6), (III.7), and (III.8),

$\mathrm{ks}_1 \approx_l^p \mathrm{ks}_2$

□

Lemma 24. If $\mathcal{F} :: pc_{src}, d_d, d_e \vdash \Sigma_1, \sigma, c \xrightarrow{\alpha}_{pc} \Sigma_1', \sigma_1, c_1, E_1$ *with* $pc \downarrow^p \sqsubseteq l$ *and* $\Sigma_1 \approx_l^p \Sigma_2$*, then* $\exists \mathcal{G} :: pc_{src}, d_d, d_e \vdash \Sigma_2, \sigma, c \xrightarrow{\alpha}_{pc} \Sigma_2', \sigma_2, c_2, E_2$ *with* $\Sigma_1' \approx_l^p \Sigma_2'$*,* $\sigma_1 = \sigma_2$*,* $c_1 = c_2$*, and* $E_1 = E_2$

Proof.

By induction on the structure of $\mathcal{F} :: pc_{src}, d_d, d_e \vdash \Sigma_1, \sigma, c \xrightarrow{\alpha}_{pc} \Sigma_1', \sigma_1, c_1, E_1$

By assumption,

(1) $pc \downarrow^p \sqsubseteq l$

(2) $\Sigma_1 \approx_l^p \Sigma_2$

Case I: $\mathcal{F}$ ends in SKIP, DECLASSIFY, or ENDORSE

The proofs for these cases are straightforward

Case II: $\mathcal{F}$ ends in SEQ

The proof for this case follows from the IH

Case III: $\mathcal{F}$ ends in ASSIGN-L

By assumption,

(III.1) $c = x := e$

(III.2) $\Sigma_1(pc) = (\sigma^g, \_)$

(III.3) $x \notin \sigma^g$

(III.4) $\Sigma_1' = \Sigma_1$

(III.5) $[\![e]\!]_{\sigma,\Sigma_1}^{pc} = v$

(III.6) $\sigma_1 = \sigma[x \mapsto v]$

(III.7) $c_1 = \mathrm{skip}$

(III.8) $E_1 = \cdot$

From (1) and (2),

(III.9) $\Sigma_1(pc) = \Sigma_2(pc)$

From (III.9) and (III.1)-(III.3),

(III.10) ASSIGN-L may be applied to Σ_2, σ, c

From (III.10),

(III.11) $\exists \mathcal{G} :: pc_{src}, d_d, d_e \vdash \Sigma_2, \sigma, c \xrightarrow{\bullet}_{pc} \Sigma_2, \sigma[x \mapsto v'], \mathrm{skip}, \cdot$ for

(III.12) $[\![e]\!]_{\sigma,\Sigma_2}^{pc} = v'$

From (1), (2), (III.5), and (III.12),
(III.13) $v = v'$
From (III.11),
(III.14) $\Sigma_2' = \Sigma_2$
(III.15) $\sigma_2 = \sigma[x \mapsto v']$
(III.16) $c_2 = \mathsf{skip}$
(III.17) $E_2 = \cdot$
From (2), (III.4), and (III.14),
$\Sigma_1' \approx_l^p \Sigma_2'$
From (III.6), (III.15), and (III.13),
$\sigma_1 = \sigma_2$
From (III.7) and (III.16),
$c_1 = c_2$
From (III.8) and (III.17),
$E_1 = E_2$

Case IV: $\mathcal{F}$ ends in ASSIGN-G
By assumption,
(IV.1) $c = x := e$
(IV.2) $\Sigma_1(pc) = (\sigma^g, \sigma^{EH})$
(IV.3) $x \in \sigma^g$
(IV.4) $[\![e]\!]^{pc}_{\sigma,\Sigma_1} = v$
(IV.5) $\sigma_1^g = \sigma^g[x \mapsto v]$
(IV.6) $\Sigma_1' = \Sigma_1[pc \mapsto (\sigma_1^g, \sigma^{EH})$
(IV.7) $\sigma_1 = \sigma$
(IV.8) $c_1 = \mathsf{skip}$
(IV.9) $E_1 = \cdot$
From (1) and (2),
(IV.10) $\Sigma_1(pc) = \Sigma_2(pc)$
From (IV.10) and (IV.1)-(IV.3),
(IV.11) ASSIGN-G may be applied to Σ_2, σ, c
From (IV.11) and (IV.10),
(IV.12) $\exists \mathcal{G} :: pc_{src}, d_d, d_e \vdash \Sigma_2, \sigma, c \xrightarrow{\bullet}_{pc} \Sigma_2'', \sigma, \mathsf{skip}, \cdot$ for
(IV.13) $[\![e]\!]^{pc}_{\sigma,\Sigma_2} = v'$
(IV.14) $\Sigma_2'' = \Sigma_2[pc \mapsto (\sigma_2^g, \sigma^{EH})]$
(IV.15) $\sigma_2^g = \sigma^g[x \mapsto v']$
From (1), (2), (IV.4), and (IV.13),
(IV.16) $v = v'$
From (IV.12),
(IV.17) $\Sigma_2' = \Sigma_2''$
(IV.18) $\sigma_2 = \sigma$
(IV.19) $c_2 = \mathsf{skip}$
(IV.20) $E_2 = \cdot$
From (2), (IV.6), (IV.17), (IV.14), (IV.5), (IV.15), and (IV.16),
$\Sigma_1' \approx_l^p \Sigma_2'$
From (IV.7) and (IV.18),
$\sigma_1 = \sigma_2$
From (IV.8) and (IV.19),
$c_1 = c_2$
From (IV.9) and (IV.20),
$E_1 = E_2$

Case V: $\mathcal{F}$ ends in UPDATE
The proof for this case is similar to **Case IV**

Case VI: $\mathcal{F}$ ends in IF-TRUE
By assumption,
(VI.1) $c = \mathsf{if}\ e\ \mathsf{then}\ c_1'\ \mathsf{else}\ c_2'$
(VI.2) $[\![e]\!]^{pc}_{\sigma,\Sigma_1} = \mathsf{true}$
(VI.3) $c_1 = c_1'$
(VI.4) $\Sigma_1' = \Sigma_1$
(VI.5) $\sigma_1 = \sigma$
(VI.6) $E_1 = \cdot$
From (1), (2), and (VI.2),
(VI.7) $[\![e]\!]^{pc}_{\sigma,\Sigma_2} = \mathsf{true}$
From (VI.7) and (VI.1),
(VI.8) IF-TRUE may be applied to Σ_2, σ, c
From (VI.8),
(VI.9) $\exists \mathcal{G} :: pc_{src}, d_d, d_e \vdash \Sigma_2, \sigma, c \xrightarrow{\bullet}_{pc} \Sigma_2, \sigma, c_1', \cdot$ for
From (VI.9),
(VI.10) $\Sigma_2' = \Sigma_2$
(VI.11) $\sigma_2 = \sigma$
(VI.12) $c_2 = c_1'$
(VI.13) $E_2 = \cdot$
From (2), (VI.4), and (VI.10),
$\Sigma_1' \approx^p_l \Sigma_2'$
From (VI.5) and (VI.11),
$\sigma_1 = \sigma_2$
From (VI.3) and (VI.12),
$c_1 = c_2$
From (VI.6) and (VI.13),
$E_1 = E_2$

Case VII: $\mathcal{F}$ ends in IF-FALSE, WHILE-TRUE, or WHILE-FALSE
The proofs for these cases are similar to **Case VI**

Case VIII: $\mathcal{F}$ ends in OUTPUT
By assumption,
(VIII.1) $c = \mathsf{output}\ ch\ e$
(VIII.2) $[\![e]\!]^{pc}_{\sigma,\Sigma_1} = v$
(VIII.3) $c_1 = \mathsf{skip}$
(VIII.4) $\Sigma_1' = \Sigma_1$
(VIII.5) $\sigma_1 = \sigma$
(VIII.6) $E_1 = \cdot$
From (1), (2), and (VIII.2),
(VIII.7) $[\![e]\!]^{pc}_{\sigma,\Sigma_2} = v$
From (VIII.7) and (VIII.1),
(VIII.8) OUTPUT may be applied to Σ_2, σ, c producing output $ch(v)$
From (VIII.8),
(VIII.9) $\exists \mathcal{G} :: pc_{src}, d_d, d_e \vdash \Sigma_2, \sigma, c \xrightarrow{ch(v)}_{pc} \Sigma_2, \sigma, c_1', \cdot$
From (VIII.9),
(VIII.10) $\Sigma_2' = \Sigma_2$
(VIII.11) $\sigma_2 = \sigma$
(VIII.12) $c_2 = \mathsf{skip}$
(VIII.13) $E_2 = \cdot$
From (2), (VIII.4), and (VIII.10),
$\Sigma_1' \approx^p_l \Sigma_2'$
From (VIII.5) and (VIII.11),
$\sigma_1 = \sigma_2$
From (VIII.3) and (VIII.12),

$c_1 = c_2$
From (VIII.6) and (VIII.13),
$E_1 = E_2$

Case IX: $\mathcal{F}$ ends in EVENT-TRIGGER
By assumption,
(IX.1) $c = \mathsf{trigger}\ id.Ev(e)$
(IX.2) $[\![e]\!]^{pc}_{\sigma,\Sigma_1} = v$
(IX.3) $c_1 = \mathsf{skip}$
(IX.4) $\Sigma_1' = \Sigma_1$
(IX.5) $\sigma_1 = \sigma$
(IX.6) $E_1 = (id.Ev(v), pc)$
From (IX.1),
(IX.7) EVENT-TRIGGER may be applied to Σ_2, σ, c
From (IX.7),
(IX.8) $\exists \mathcal{G} :: pc_{src}, d_d, d_e \vdash \Sigma_2, \sigma, c \xrightarrow{\bullet}_{pc} \Sigma_2, \sigma, \mathsf{skip}, (id.Ev(v'), pc)$ for
(IX.9) $[\![e]\!]^{pc}_{\sigma,\Sigma_2} = v'$
From (1), (2), (IX.2), and (IX.9),
(IX.10) $v = v'$
From (IX.9),
(IX.11) $\Sigma_2' = \Sigma_2$
(IX.12) $\sigma_2 = \sigma$
(IX.13) $c_2 = \mathsf{skip}$
(IX.14) $E_2 = (id.Ev(v'), pc)$
From (2), (IX.4), and (IX.11),
$\Sigma_1' \approx^p_l \Sigma_2'$
From (IX.5) and (IX.12),
$\sigma_1 = \sigma_2$
From (IX.3) and (IX.13),
$c_1 = c_2$
From (IX.6), (IX.14), and (IX.10),
$E_1 = E_2$

Case X: $\mathcal{F}$ ends in NEW
By assumption,
(X.1) $c = \mathsf{new}(id, e)$
(X.2) $[\![e]\!]^{pc}_{\sigma,\Sigma_1} = v$
(X.3) $\Sigma_1(pc) = (\sigma^g, \sigma^{EH})$
(X.4) $id \notin \sigma^{EH}$
(X.5) $\sigma_1^{EH} = \sigma^{EH}[id \mapsto (v, \cdot, pc_{src})]$
(X.6) $\Sigma_1' = \Sigma_1[pc \mapsto (\sigma^g, \sigma_1^{EH})]$
(X.7) $c_1 = \mathsf{skip}$
(X.8) $\sigma_1 = \sigma$
(X.9) $E_1 = \cdot$
From (X.1),
(X.10) NEW may be applied to Σ_2, σ, c producing output $\mathsf{new}(id, pc_{src})$
From (1) and (2),
(X.11) $\Sigma_1(pc) = \Sigma_2(pc)$
From (X.10) and (X.11),
(X.12) $\exists \mathcal{G} :: pc_{src}, d_d, d_e \vdash \Sigma_2, \sigma, c \xrightarrow{\bullet}_{pc} \Sigma_2'', \sigma, \mathsf{skip}, \cdot$ for
(X.13) $[\![e]\!]^{pc}_{\sigma,\Sigma_2} = v'$
(X.14) $\Sigma_2'' = (\sigma^g, \sigma_2^{EH})$
(X.15) $\sigma_2^{EH} = \sigma^{EH}[id \mapsto (v', \cdot, pc_{src})]$
From (1), (2), (X.2), and (X.13),
(X.16) $v = v'$

From (X.12),

(X.17) $\Sigma_2' = \Sigma_2''$

(X.18) $\sigma_2 = \sigma$

(X.19) $c_2 = \mathsf{skip}$

(X.20) $E_2 = \cdot$

From (2), (X.5), (X.6), (X.14), and (X.15),

$\Sigma_1' \approx_l^p \Sigma_2'$

From (X.8) and (X.18),

$\sigma_1 = \sigma_2$

From (X.7) and (X.19),

$c_1 = c_2$

From (X.9) and (X.20),

$E_1 = E_2$

Case XI: $\mathcal{F}$ ends in ADD-EH

By assumption,

(XI.1) $c = \mathsf{addEH}(id, eh)$

(XI.2) $\Sigma_1(pc) = (\sigma^g, \sigma^{EH})$,

(XI.3) $\sigma^{EH}(id) = (v, M, pc_{id})$

(XI.4) $M(Ev) = EH_{Ev}$

(XI.5) $M' = M[Ev \mapsto EH_{Ev} \cup \{(eh, pc_{src})\}]$

(XI.6) $\sigma_1^{EH} = \sigma^{EH}[id \mapsto (v, M', pc_{id})]$

(XI.7) $\Sigma_1' = \Sigma_1[pc \mapsto (\sigma^g, \sigma_1^{EH})]$

(XI.8) $\alpha = \mathsf{newEH}(id, eh, pc_{id}, pc_{src})$

(XI.9) $c_1 = \mathsf{skip}$

(XI.10) $\sigma_1 = \sigma$

(XI.11) $E_1 = \cdot$

From (1) and (2),

(XI.12) $\Sigma_1(pc) = \Sigma_2(pc)$

From (XI.1), (XI.12), (XI.2), (XI.3), and (XI.8),

(XI.13) ADD-EH may be applied to Σ_2, σ, c producing α

From (XI.12), (XI.2), (XI.3), (XI.5), and (XI.13),

(XI.14) $\exists \mathcal{G} :: pc_{src}, d_d, d_e \vdash \Sigma_2, \sigma, c \xrightarrow{\alpha}_{pc} \Sigma_2'', \sigma, \mathsf{skip}, \cdot$ for

(XI.15) $\Sigma_2'' = (\sigma^g, \sigma_2^{EH}$

(XI.16) $\sigma_2^{EH} = \sigma^{EH}[id \mapsto (v, M', pc_{id})]$

From (XI.14),

(XI.17) $\Sigma_2' = \Sigma_2''$

(XI.18) $\sigma_2 = \sigma$

(XI.19) $c_2 = \mathsf{skip}$

(XI.20) $E_2 = \cdot$

From (2), (XI.5)-(XI.7), and (XI.15)-(XI.17),

$\Sigma_1' \approx_l^p \Sigma_2'$

From (XI.10) and (XI.18),

$\sigma_1 = \sigma_2$

From (XI.9) and (XI.19),

$c_1 = c_2$

From (XI.11) and (XI.20),

$E_1 = E_2$

□

Lemma 25. If $T = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{\tau}{\Longrightarrow}{}^* K'$ with $\forall(\alpha, pc) \in \tau, \alpha \in \{ch(\_), \bullet\} \wedge pc \downarrow^p \not\sqsubseteq l$, then $T \downarrow_l^p = \cdot$

Proof.

By induction on the length of T

By assumption,

(1) $\forall(\alpha, pc) \in \tau, \alpha \in \{ch(\_), \bullet\} \wedge pc \downarrow^p \not\sqsubseteq l$

Base Case: $\mathsf{len}(T) = 0$

By assumption, $T = K$. Then, from the definition of $\downarrow_l^p$ for T, $T \downarrow_l^p = \cdot$

Inductive Case: $\mathsf{len}(T) = n + 1$

By assumption,

(I.1) $\mathcal{F} :: T = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{\tau'}{\Longrightarrow}^* K'' \overset{\alpha_l}{\Longrightarrow} K'$

From (1) and (I.1),

(I.2) $\tau = \tau' :: \alpha_l$ with

(I.3) $\forall (\alpha, pc) \in \tau', \alpha \in \{ch(\_), \bullet\} \wedge pc \downarrow^p \not\sqsubseteq l$

(I.4) $\alpha_l = (\alpha', pc')$ with $\alpha' \in \{ch(\_), \bullet\} \wedge pc' \downarrow^p \not\sqsubseteq l$

From (I.3),

(I.5) the IH may be applied on $\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{\tau'}{\Longrightarrow}^* K''$

From (I.5) and the IH,

(I.6) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K \overset{\tau'}{\Longrightarrow}^* K'') \downarrow_l^p = \cdot$

From (I.4),

(I.7) $\mathcal{F}$ must end in an output rule

Subcase i: $\mathcal{F}$ ends in Out-Skip, Out-Silent or Out-Next,

By assumption and from (1) and (I.4),

(i.1) $\alpha' = \bullet$

From (i.1), (I.4) and the definition of $\downarrow_l^p$ for T,

(i.2) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K'' \overset{\alpha_l}{\Longrightarrow} K') \downarrow_l^p = \cdot$

From (I.1), (I.6), and (i.2),

$T \downarrow_l^p = \cdot$

Subcase ii: $\mathcal{F}$ ends in Out

By assumption,

(ii.1) $\alpha' = ch(v)$ and

(ii.2) $\mathcal{P}(ch) = pc'$

From (ii.1), (ii.2), (I.4), and the definition of $\downarrow_l^p$ for T,

(ii.3) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K'' \overset{\alpha_l}{\Longrightarrow} K') \downarrow_l^p = \cdot$

From (I.1), (I.6), and (ii.3),

$T \downarrow_l^p = \cdot$

□

Lemma 26. If $\Sigma_1 \approx_l^p \Sigma_2$ *with* $pc \downarrow^p \sqsubseteq l$, *with* $\mathsf{robust}(\Sigma_1, E, pc) = E_1$ *and* $\mathsf{robust}(\Sigma_2, E, pc) = E_2$, *then* $E_1 \approx_l^p E_2$ *if* $p = c$ *and* $E_1 = E_2$ *if* $p = i$

Proof.

By induction on the structure of $\mathcal{F} :: \mathsf{robust}(\Sigma_1, E, pc) = E_1$ and $\mathcal{G} :: \mathsf{robust}(\Sigma_2, E, pc) = E_2$

By assumption,

(1) $\Sigma_1 \approx_l^p \Sigma_2$

(2) $pc \downarrow^p \sqsubseteq l$

Case I: $\mathcal{F}$ ends in robust

By assumption,

(I.1) $E = (id.Ev(v), pc') :: E'$

(I.2) $\Sigma_1(pc') = (\_, \sigma_1^{EH})$

(I.3) $\sigma_1^{EH}(id) \downarrow^i \sqsubseteq pc \downarrow^i$

(I.4) $\exists \mathcal{F}' :: \mathsf{robust}(\Sigma_1, E', pc) = E_1'$

(I.5) $E_1 = (id.Ev(v), pc') :: E_1'$

From (I.1),

(I.6) $\mathcal{G}$ ends in robust or not-robust

From (I.6),

(I.7) $\exists \mathcal{G}' :: \text{robust}(\Sigma_2, E', pc) = E_2'$

From (1), (I.4), and (I.7), IH may be applied on $\mathcal{F}'$ and $\mathcal{G}'$,

(I.8) $E_1' \approx_l^p E_2'$ if $p = c$ and $E_1' = E_2'$ if $p = i$

Subcase i: $pc' \downarrow^p \sqsubseteq l$

By assumption and from (1),

(i.1) $\Sigma_1(pc') = \Sigma_2(pc')$

From (i.1) and (I.2),

(i.2) $\Sigma_2(pc') = (\_, \sigma_2^{EH}) = (\_, \sigma_1^{EH})$

From (i.2) and (I.3),

(i.3) $\sigma_2^{EH}(id) \downarrow^i \sqsubseteq pc \downarrow^i$

From (i.3),

(i.4) $\mathcal{G}$ ends in ROBUST

From (i.4),

(i.5) $E_2 = (id.Ev(v), pc') :: E_2'$

From (I.5), (i.5), and (I.8),

$E_1 \approx_l^p E_2$ if $p = c$ and $E_1 = E_2$ if $p = i$

Subcase ii: $pc' \downarrow^p \not\sqsubseteq l$ and $p = c$

By assumption and from (I.5),

(ii.1) $E_1 \approx_l^p E_1'$

If $\mathcal{G}$ ends in ROBUST, then

(ii.2) $E_2 = (id.Ev(v), pc') :: E_2'$

By assumption and from (ii.2),

(ii.3) $E_2 \approx_l^p E_2'$

From (ii.1), (ii.3), and (I.8),

$E_1 \approx_l^p E_2$

Otherwise, $\mathcal{G}$ ends in NOT-ROBUST and

(ii.4) $E_2 = E_2'$

From (ii.1), (ii.4), and (I.8),

$E_1 \approx_l^p E_2$

Subcase iii: $pc' \downarrow^p \not\sqsubseteq l$ and $p = i$

By assumption and from (1), (I.2), and $\Sigma_2(pc') = (\_, \sigma_2^{EH})$,

(iii.1) $\sigma_1^{EH} \downarrow_l^i = \sigma_2^{EH} \downarrow_l^i$

From (2) and (I.3),

(iii.2) $\sigma_1^{EH}(id) \downarrow^i \sqsubseteq l$

From (iii.1) and (iii.2),

(iii.3) $\sigma_2^{EH}(id) \downarrow^i = \sigma_1^{EH}(id) \downarrow^i$

From (iii.3) and (I.3),

(iii.4) $\sigma_2^{EH}(id) \downarrow^i \sqsubseteq pc \downarrow^i$

From (iii.4),

(iii.5) $\mathcal{G}$ msut end in ROBUST

From (iii.5),

(iii.6) $E_2 = (id.Ev(v), pc') :: E_2'$

From (I.5), (iii.6), and (I.8),

$E_1 = E_2$

Case II: $\mathcal{F}$ ends in NOT-ROBUST

By assumption,

(II.1) $E = (id.Ev(v), pc') :: E'$

(II.2) $\Sigma_1(pc') = (\_, \sigma_1^{EH})$

(II.3) $id \notin \sigma_1^{EH}$ or $\sigma_1^{EH}(id) \downarrow^i \not\sqsubseteq pc \downarrow^i$

(II.4) $\exists \mathcal{F}' :: \text{robust}(\Sigma_1, E', pc) = E_1'$

From (II.4),

(II.5) $E_1 = E_1'$

From (II.1),

(II.6) $\mathcal{G}$ ends in ROBUST or NOT-ROBUST

From (II.6),

(II.7) $\exists \mathcal{G}' :: \text{robust}(\Sigma_2, E', pc) = E_2'$

From (1), (II.4), and (II.7), IH may be applied on $\mathcal{F}'$ and $\mathcal{G}'$,

(II.8) $E_1' \approx_l^p E_2'$ if $p = c$ and $E_1' = E_2'$ if $p = i$

Subcase i: $pc' \downarrow^p \sqsubseteq l$

By assumption and from (1),

(i.1) $\Sigma_1(pc') = \Sigma_2(pc')$

From (i.1) and (II.2),

(i.2) $\Sigma_2(pc') = (\_, \sigma_2^{EH}) = (\_, \sigma_1^{EH})$

From (i.2) and (II.3),

(i.3) $\sigma_2^{EH}(id) \downarrow^i \not\sqsubseteq pc \downarrow^i$

From (i.3),

(i.4) $\mathcal{G}$ ends in NOT-ROBUST

From (i.4),

(i.5) $E_2 = E_2'$

From (II.5), (i.5), and (II.8),

$E_1 \approx_l^p E_2$ if $p = c$ and $E_1 = E_2$ if $p = i$

Subcase ii: $pc' \downarrow^p \not\sqsubseteq l$ and $p = c$

If $\mathcal{G}$ ends in ROBUST, then

(ii.1) $E_2 = (id.Ev(v), pc') :: E_2'$

By assumption and from (ii.1),

(ii.2) $E_2 \approx_l^p E_2'$

From (II.8) and (ii.2),

$E_1 \approx_l^p E_2$

Otherwise, $\mathcal{G}$ ends in NOT-ROBUST and

(ii.3) $E_2 = E_2'$

From (II.8) and (ii.3),

$E_1 \approx_l^p E_2$ if $p = c$ and $E_1 = E_2$ if $p = i$

Subcase iii: $pc' \downarrow^p \not\sqsubseteq l$ and $p = i$

By assumption and from (1), (II.2), and $\Sigma_2(pc') = (\_, \sigma_2^{EH})$

(iii.1) $\sigma_1^{EH} \downarrow_l^i = \sigma_2^{EH} \downarrow_l^i$

From (2) and (II.3),

(iii.2) $\sigma_1^{EH}(id) \downarrow^i \not\sqsubseteq l$

From (iii.1) and (iii.2),

(iii.3) $\sigma_2^{EH}(id) \downarrow^i = \sigma_1^{EH}(id) \downarrow^i$

From (iii.3) and (II.3),

(iii.4) $\sigma_2^{EH}(id) \downarrow^i \not\sqsubseteq pc \downarrow^i$

From (iii.4),

(iii.5) $\mathcal{G}$ must end in NOT-ROBUST

From (iii.5),

(iii.6) $E_2 = E_2'$

From (II.5), (iii.6), and (II.8),

$E_1 = E_2$

Case III: $\mathcal{F}$ ends in ROBUST-EMP

By assumption, $E_1 = E_2 = \cdot$

□

Lemma 27. *If* $\Sigma_1 \approx_l^p \Sigma_2$ *and* $pc \downarrow^p \sqsubseteq l$*, with* $\mathsf{transparent}(\Sigma_1, E, pc) = E_1$ *and* $\mathsf{transparent}(\Sigma_2, E, pc) = E_2$*, then* $E_1 \approx_l^p E_2$ *if* $p = c$ *and* $E_1 = E_2$ *if* $p = i$

Proof (sketch): The proof is by induction on the structure of $\mathcal{F} :: \mathsf{transparent}(\Sigma_1, E, pc) = E_1$ and $\mathcal{G} :: \mathsf{transparent}(\Sigma_2, E, pc) = E_2$, similar to the one for Lemma 26. □

Lemma 28. *If* $\Sigma_1 \approx_l^i \Sigma_2$*, with* $pc_{Ev}, \mathsf{r} \vdash \Sigma_1, E \rightsquigarrow \mathsf{ks}_1$*,* $pc_{Ev}, \mathsf{r} \vdash \Sigma_2, E \rightsquigarrow \mathsf{ks}_2$ *and* $pc_{Ev} \downarrow^i \sqsubseteq l$ $E = \mathsf{robust}(\Sigma_1, \_, pc) = \mathsf{robust}(\Sigma_2, \_, pc)$ *then* $\mathsf{ks}_1 = \mathsf{ks}_2$

Proof.

By induction on the structure of $\mathcal{F} :: pc_{Ev}, \mathsf{f} \vdash \Sigma_1, E \rightsquigarrow \mathsf{ks}_1$ and
$\mathcal{G} :: pc_{Ev}, \mathsf{f} \vdash \Sigma_2, E \rightsquigarrow \mathsf{ks}_2$
By assumption,
(1) $\Sigma_1 \approx_l^i \Sigma_2$
(2) $pc_{Ev} \downarrow^i \sqsubseteq l$
(3) $E = \mathsf{robust}(\Sigma_1, \_, pc) = \mathsf{robust}(\Sigma_2, \_, pc)$

Case I: $\mathcal{F}$ ends in LOOKUP-R
By assumption,
(I.1) $E = (id.Ev(v), pc') :: E'$
(I.2) $\Sigma_1(pc') = (\_, \sigma_1^{EH})$
(I.3) $\sigma_1^{EH}(id) = (\_, M_1, pc'')$
(I.4) $EH_1 = M_1(Ev) \downarrow_{l_{Ev}}^i \neq \cdot$ for $l_{Ev} = pc_{Ev} \downarrow^i$
(I.5) $pc', pc'', v \vdash EH_1 \rightsquigarrow \mathsf{ks}_1'$
(I.6) $\mathcal{F}' :: pc_{Ev}, \mathsf{r} \vdash \Sigma_1, E' \rightsquigarrow \mathsf{ks}_1''$
(I.7) $\mathsf{ks}_1 = \mathsf{ks}_1' :: \mathsf{ks}_1''$
From (3) and (I.3),
(I.8) $pc'' \downarrow^i \sqsubseteq pc_{Ev} \downarrow^i$
From (2) and (I.8),
(I.9) $pc'' \downarrow^i \sqsubseteq l$

Subcase i: $pc' \downarrow^i \sqsubseteq l$
By assumption and from (1) and (I.2),
(i.1) $\Sigma_1(pc') = (\_, \sigma_1^{EH}) = (\_, \sigma_2^{EH}) = \Sigma_2(pc')$
From (i.1) and (I.3),
(i.2) $\sigma_2^{EH}(id) = \sigma_1^{EH}(id) = (\_, M_1, pc'')$
From (i.2) and (I.4),
(i.3) $\mathcal{G}$ ends in LOOKUP-R
From (i.2), (i.3), and (I.4),
(i.4) $pc', pc'', v \vdash EH_1 \rightsquigarrow \mathsf{ks}_2'$
(i.5) $\mathcal{G}' :: pc_{Ev}, \mathsf{r} \vdash \Sigma_2, E' \rightsquigarrow \mathsf{ks}_2''$
(i.6) $\mathsf{ks}_2 = \mathsf{ks}_2' :: \mathsf{ks}_2''$
From (I.5) and (i.4),
(i.7) $\mathsf{ks}_1' = \mathsf{ks}_2'$
From (1), (I.6), and (i.5), and the IH on $\mathcal{F}'$ and $\mathcal{G}'$,
(i.8) $\mathsf{ks}_1'' = \mathsf{ks}_2''$
From (I.7) and (i.6)-(i.8),
$\mathsf{ks}_1 = \mathsf{ks}_2$

Subcase ii: $pc' \downarrow^i \not\sqsubseteq l$
By assumption and from (1), (I.2), and $\Sigma_2 = (\_, \sigma_2^{EH})$,
(ii.1) $(\_, \sigma_1^{EH}) \downarrow_l^i = (\_, \sigma_2^{EH}) \downarrow_l^i$
From (I.3), (I.9), and (ii.1),
(ii.2) $\sigma_2^{EH}(id) = (\_, M_2, pc'')$ with
(ii.3) $M_1 \downarrow_l^i = M_2 \downarrow_l^i$
From (2), (I.4), and (ii.3),
(ii.4) $EH_2 = M_2(Ev) \downarrow_{l_{Ev}}^i = EH_1$ for $l_{Ev} = pc_{Ev} \downarrow^i$
From (I.4) and (ii.4),

(ii.5) $EH_2 \neq \cdot$

From (ii.2), (ii.4), and (ii.5),

(ii.6) $\mathcal{G}$ ends in LOOKUP-R

From (ii.4) and (ii.6),

(ii.7) $pc', pc'', v \vdash EH_2 \rightsquigarrow \mathsf{ks}_2'$

(ii.8) $\mathcal{G}' :: pc_{Ev}, \mathsf{r} \vdash \Sigma_2, E' \rightsquigarrow \mathsf{ks}_2''$

(ii.9) $\mathsf{ks}_2 = \mathsf{ks}_2' :: \mathsf{ks}_2''$

From (I.5), (ii.7), and (ii.4)

(ii.10) $\mathsf{ks}_1' = \mathsf{ks}_2'$

From (1), (I.6), and (ii.8), and the IH on $\mathcal{F}'$ and $\mathcal{G}'$,

(ii.11) $\mathsf{ks}_1'' = \mathsf{ks}_2''$

From (I.7) and (ii.9)-(ii.11),

$\mathsf{ks}_1 = \mathsf{ks}_2$

Case II: $\mathcal{F}$ ends in LOOKUP-NOTR

By assumption,

(II.1) $E = (id.Ev(v), pc') :: E'$

(II.2) $\Sigma_1(pc') = (\_, \sigma_1^{EH})$

(II.3) $\sigma_1^{EH}(id) = (\_, M_1, pc'')$

(II.4) $Ev \notin M_1$ or $M_1(Ev) \downarrow^i_{l_{Ev}} = \cdot$ for $l_{Ev} = pc_{Ev} \downarrow^i$

(II.5) $\mathcal{F}' :: pc, \mathsf{r} \vdash \Sigma_1, E' \rightsquigarrow \mathsf{ks}_1$

From (3) and (II.3),

(II.6) $pc'' \downarrow^i \sqsubseteq pc_{Ev} \downarrow^i$

From (2) and (II.6),

(II.7) $pc'' \downarrow^i \sqsubseteq l$

Subcase i: $pc' \downarrow^i \sqsubseteq l$

By assumption and from (1) and (II.2),

(i.1) $\Sigma_1(pc') = (\_, \sigma_1^{EH}) = (\_, \sigma_2^{EH}) = \Sigma_2(pc')$

From (i.1) and (II.3),

(i.2) $\sigma_2^{EH}(id) = \sigma_1^{EH}(id) = (\_, M_1, pc'')$

From (i.2) and (II.4),

(i.3) $\mathcal{G}$ ends in LOOKUP-NOTR

From (II.1) and (i.3),

(i.4) $\mathcal{G}' :: pc_{Ev}, \mathsf{r} \vdash \Sigma_2, E' \rightsquigarrow \mathsf{ks}_2$

From (1), (II.5), and (i.4), and the IH on $\mathcal{F}'$ and $\mathcal{G}'$,

$\mathsf{ks}_1 = \mathsf{ks}_2$

Subcase ii: $pc' \downarrow^i \not\sqsubseteq l$

By assumption and from (1), (II.2), and $\Sigma_2 = (\_, \sigma_2^{EH})$,

(ii.1) $(\_, \sigma_1^{EH}) \downarrow^i_l = (\_, \sigma_2^{EH}) \downarrow^i_l$

From (II.3), (II.7), and (ii.1),

(ii.2) $\sigma_2^{EH}(id) = (\_, M_2, pc'')$ with

(ii.3) $M_1 \downarrow^i_l = M_2 \downarrow^i_l$

From (2), (II.4), and (ii.3),

(ii.4) $Ev \notin M_2$ or $M_2(Ev) \downarrow^i_{l_{Ev}} = \cdot$ for $l_{Ev} = pc_{Ev} \downarrow^i$

From (ii.2) and (ii.4),

(ii.5) $\mathcal{G}$ ends in LOOKUP-NOTR

From (II.1) and (ii.5),

(ii.6) $\mathcal{G}' :: pc_{Ev}, \mathsf{r} \vdash \Sigma_2, E' \rightsquigarrow \mathsf{ks}_2$

From (1), (II.5), and (ii.6), and the IH on $\mathcal{F}'$ and $\mathcal{G}'$,

$\mathsf{ks}_1 = \mathsf{ks}_2$

Case III: $\mathcal{F}$ ends in LOOKUP-R-EMP

The proofs for LOOKUP-T-EMP and LOOKUP-RT-EMP are similar

By assumption, $\mathsf{ks}_1 = \mathsf{ks}_2 = \cdot$

□

Lemma 29. *If* $\Sigma_1 \approx^c_l \Sigma_2$, *with* $pc_{Ev}, \mathsf{t} \vdash \Sigma_1, E \rightsquigarrow \mathsf{ks}_1$, $pc_{Ev}, \mathsf{t} \vdash \Sigma_2, E \rightsquigarrow \mathsf{ks}_2$ *and* $pc_{Ev} \downarrow^c \sqsubseteq l$ $E = \mathsf{transparent}(\Sigma_1, \_, pc) = \mathsf{transparent}(\Sigma_2, \_, pc)$ *then* $\mathsf{ks}_1 = \mathsf{ks}_2$

Proof (sketch): The proof is by induction on the structure of $\mathcal{F} :: pc_{Ev}, \mathsf{t} \vdash \Sigma_1, E \rightsquigarrow \mathsf{ks}_1$ and $\mathcal{G} :: pc_{Ev}, \mathsf{t} \vdash \Sigma_2, E \rightsquigarrow \mathsf{ks}_2$, similar to Lemma 28.

□

Lemma 30 (Strong One-step – Downgrade). *If* $K_1 \approx^p_l K_2$, $T_1 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \overset{\alpha_{l,1}}{\Longrightarrow} K_1'$ *with* $T_1 \downarrow^p_l = \tau = \mathsf{down}(\_)$ *and* $\mathsf{prog}(K_2)$, *with* $\mathsf{releaseT}(K_2, \tau, l)$ *if* $p = c$, *and* $\mathsf{sanitizeT}(K_2, \tau, l)$ *if* $p = i$, *then* $\exists K_2', T_2$ *s.t.* $T_2 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_2'$ *with* $T_1 \approx^p_l T_2$ *and* $K_1' \approx^p_l K_2'$

Proof.

Denote $\mathcal{F} :: \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_1 \overset{\alpha_{l,1}}{\Longrightarrow} K_1'$
Without loss of generality, assume that $p = c$ (the proof for $p = i$ is similar)
By assumption,
(1) $K_1 \approx^p_l K_2$
(2) $T_1 \downarrow^p_l = \tau = \mathsf{down}(\_)$
(3) $\mathsf{prog}(K_2)$
(4) $\mathsf{releaseT}(K_2, \tau, l)$
From (1),
(5) $\mathcal{R}_1 = \mathcal{R}_2$
(6) $\mathcal{S}_1 = \mathcal{S}_2$
(7) $\Sigma_1 \approx^p_l \Sigma_2$
From (2),
(8) $\mathcal{F}$ ends in In-DE
From (4), $\exists K_C, K_2'$ s.t.
(9) $T_2 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_C \overset{\alpha_{l,2}}{\Longrightarrow} K_2'$ with
(10) $\mathsf{consumer}(K_C)$
(11) $(\mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K_C) \downarrow^p_l = \cdot$
(12) $T_2 \downarrow^p_l = \mathsf{down}(\ldots)$
From (11) and Lemma 10,
(13) $K_2 \approx^p_l K_C$
From (13) and (5)-(7),
(14) $\mathcal{R}_1 = \mathcal{R}_C$
(15) $\mathcal{S}_1 = \mathcal{S}_C$
(16) $\Sigma_1 \approx^p_l \Sigma_C$
From (2), (10), and (12),
(17) $\mathcal{F}$ ends in In-DE
From (17),
(18) $\alpha_{l,1} = (id.Ev(v), pc)$
(19) $\mathcal{P}(id.Ev(v)) = pc'$
(20) $\Sigma_1(pc) = (\_, \sigma^{EH})$
(21) $\sigma^{EH}(id) \downarrow^i \sqsubseteq pc \downarrow^i$
(22) $\sigma^{EH}(id) \downarrow^c \sqsubseteq pc \downarrow^c$
(23) $E_1 = ((id.Ev(v), pc'') \mid pc \sqcup pc' \sqsubseteq pc'')$
(24) $\Sigma_1, E_1 \rightsquigarrow \mathsf{ks}_1''$
(25) $\mathsf{downgrade}_{\mathcal{D}}(\mathcal{R}_1, \Sigma_1, \alpha_{l,1}, pc') = (\mathcal{R}_1', E_{d,1})$
(26) $pc, \mathsf{r} \vdash \Sigma_1, E_{d,1} \rightsquigarrow \mathsf{ks}_{d,1}$
(27) $\mathsf{downgrade}_{\mathcal{E}}(\mathcal{S}_1, \Sigma_1, \alpha_{l,1}, pc') = (\mathcal{S}_1', E_{e,1})$
(28) $pc, \mathsf{t} \vdash \Sigma_1, E_{e,1} \rightsquigarrow \mathsf{ks}_{e,1}$
(29) $\mathsf{downgrade}_{\mathcal{D},\mathcal{E}}(\mathcal{R}_1, \Sigma_1, \alpha_{l,1}, pc') = E_{m,1}$
(30) $pc, \mathsf{rt} \vdash \Sigma_1, E_{m,1} \rightsquigarrow \mathsf{ks}_{m,1}$
(31) $\Sigma_1' = \Sigma_1$
(32) $\mathsf{ks}_1' = \mathsf{ks}_1'' :: \mathsf{ks}_{d,1} :: \mathsf{ks}_{e,1} :: \mathsf{ks}_{m,1}$
From (25) and the definition of $\mathsf{downgrade}_{\mathcal{D}}$
(33) $\mathcal{R}_1 = (\rho_{d,1}, d_{d,1})$

(34) $E'_{d,1} = ((id.Ev(v), (l_c, l_i)) \mid pc \downarrow^c \sqsubseteq l_c \sqsubset pc' \downarrow^c \wedge l_i = pc \downarrow^i \sqcup pc' \downarrow^i)$
(35) $\mathcal{D}((id.Ev(v), pc), pc', \rho_{d,1}) = (\rho'_{d,1}, v_{d,1}, E''_{d,1})$
(36) $d'_{d,1} = \mathsf{update}(d_{d,1}, v_{d,1})$
(37) $\mathcal{R}'_1 = (\rho'_{d,1}, d'_{d,1})$
(38) $E_{d,1} = \mathsf{robust}(\Sigma_1, E'_{d,1} :: E''_{d,1}, pc)$
From (27) and the definition of downgrade$_{\mathcal{E}}$
(39) $\mathcal{S}_1 = (\rho_{e,1}, d_{e,1})$
(40) $E'_{e,1} = ((id.Ev(v), (l_c, l_i)) \mid pc \downarrow^i \sqsubseteq l_i \sqsubset pc' \downarrow^i \wedge l_c = pc \downarrow^c \sqcup pc' \downarrow^c)$
(41) $\mathcal{E}((id.Ev(v), pc), pc', \rho_{e,1}) = (\rho'_{e,1}, v_{e,1}, E''_{e,1})$
(42) $d'_{e,1} = \mathsf{update}(d_{e,1}, v_{e,1})$
(43) $\mathcal{S}'_1 = (\rho'_{e,1}, d'_{e,1})$
(44) $E_{e,1} = \mathsf{transparent}(\Sigma_1, E'_{e,1} :: E''_{e,1}, pc)$
From (29) and the definition of downgrade$_{\mathcal{D},\mathcal{E}}$,
(45) $E_{m,1} = \mathsf{mergeEvents}(E'_{d,1} :: E''_{d,1}, E'_{e,1} :: E''_{e,1})$
Denote $\mathcal{G} :: \mathcal{P}, \mathcal{E}, \mathcal{E} \vdash K_C \Longrightarrow K'_2$
From (2),
(46) $\tau = \mathsf{down}(id.Ev(v), \tau_{\mathsf{rls}}, \tau_{\mathsf{sntz},1}, E_{m,1}, pc)$
From (46),
(47) $\mathcal{G}$ ends in In-DE with input $\alpha_{l,2} = id.Ev(v)$, producing trace
$T_2 = \mathcal{P}, \mathcal{D}, \mathcal{E} \vdash K_2 \Longrightarrow^* K'_2$
Want to show $T_1 \approx^p_l T_2$ and $K'_1 \approx^p_l K'_2$
From (47) and (19),
(48) $\mathcal{P}(id.Ev(v)) = pc'$
From (47) and (48),
(49) $E_2 = ((id.Ev(v), pc'') \mid pc \sqcup pc' \sqsubseteq pc'')$
(50) $\Sigma_C, E_2 \rightsquigarrow \mathsf{ks}''_2$
(51) $\mathsf{downgrade}_{\mathcal{D}}(\mathcal{R}_C, \Sigma_C, \alpha_{l,2}, pc') = (\mathcal{R}'_2, E_{d,2})$
(52) $pc, \mathsf{r} \vdash \Sigma_C, E_{d,2} \rightsquigarrow \mathsf{ks}_{d,2}$
(53) $\mathsf{downgrade}_{\mathcal{E}}(\mathcal{S}_C, \Sigma_C, \alpha_{l,2}, pc') = (\mathcal{S}'_2, E_{e,2})$
(54) $pc, \mathsf{t} \vdash \Sigma_C, E_{e,2} \rightsquigarrow \mathsf{ks}_{e,2}$
(55) $\mathsf{downgrade}_{\mathcal{D},\mathcal{E}}(\mathcal{R}_C, \Sigma_C, \alpha_{l,2}, pc') = E_{m,2}$
(56) $pc, \mathsf{rt} \vdash \Sigma_C, E_{m,2} \rightsquigarrow \mathsf{ks}_{m,2}$
(57) $\Sigma'_2 = \Sigma_C$
(58) $\mathsf{ks}'_2 = \mathsf{ks}''_2 :: \mathsf{ks}_{d,2} :: \mathsf{ks}_{e,2} :: \mathsf{ks}_{m,2}$
From (51) and the definition of downgrade$_{\mathcal{D}}$,
(59) $\mathcal{R}_C = (\rho_{d,C}, d_{d,C})$
(60) $E'_{d,2} = ((id.Ev(v), (l_c, l_i)) \mid pc \downarrow^c \sqsubseteq l_c \sqsubset pc' \downarrow^c \wedge l_i = pc \downarrow^i \sqcup pc' \downarrow^i)$
(61) $\mathcal{D}((id.Ev(v), pc), pc', \rho_{d,C}) = (\rho'_{d,2}, v_{d,2}, E''_{d,2})$
(62) $d'_{d,2} = \mathsf{update}(d_{d,C}, v_{d,2})$
(63) $\mathcal{R}'_2 = (\rho'_{d,2}, d'_{d,2})$
(64) $E_{d,2} = \mathsf{robust}(\Sigma_C, E'_{d,2} :: E''_{d,2}, pc)$
From (53) and the definition of downgrade$_{\mathcal{E}}$,
(65) $\mathcal{S}_C = (\rho_{e,C}, d_{e,C})$
(66) $E'_{e,2} = ((id.Ev(v), (l_c, l_i)) \mid pc \downarrow^i \sqsubseteq l_i \sqsubset pc' \downarrow^i \wedge l_c = pc \downarrow^c \sqcup pc' \downarrow^c)$
(67) $\mathcal{E}((id.Ev(v), pc), pc', \rho_{e,C}) = (\rho'_{e,2}, v_{e,2}, E''_{e,2})$
(68) $d'_{e,2} = \mathsf{update}(d_{e,C}, v_{e,2})$
(69) $\mathcal{S}'_2 = (\rho'_{e,2}, d'_{e,2})$
(70) $E_{e,2} = \mathsf{transparent}(\Sigma_C, E'_{e,2} :: E''_{e,2}, pc)$
From (55) and the definition of downgrade$_{\mathcal{D},\mathcal{E}}$,
(71) $E_{m,2} = \mathsf{mergeEvents}(E'_{d,2} :: E''_{d,2}, E'_{e,2} :: E''_{e,2})$
From (46), (4), and (12),
(72) $T_2 \downarrow^p_l = \mathsf{down}(id.Ev(v), \tau_{\mathsf{rls}}, \_, E_{m,1}, pc)$
From (46) and the definition of trDowngrade,

(73) $pc \downarrow^p \sqsubseteq l$
(74) $\mathsf{ks}_{m,1} \downarrow_l^p \neq \cdot$
From (14), (15), (33), (39), (59), and (65),
(75) $(\rho_{d,1}, d_{d,1}) = (\rho_{d,C}, d_{d,C})$
(76) $(\rho_{e,1}, d_{e,1}) = (\rho_{e,C}, d_{e,C})$

Case I: $pc \downarrow^p \sqsubseteq l$
By assumption and from (16),
(I.1) $\Sigma_1(pc) = \Sigma_C(pc)$
From (75), (35), and (61),
(I.2) $(\rho'_{d,1}, v_{d,1}, E''_{d,1}) = (\rho'_{d,2}, v_{d,2}, E''_{d,2})$
From (75), (I.2), (36), and (62),
(I.3) $d'_{d,1} = d'_{d,2}$
From (I.2), (I.3), (37), and (63),
(I.4) $\mathcal{R}'_1 = \mathcal{R}'_2$
From (76), (41), and (67),
(I.5) $(\rho'_{e,1}, v_{e,1}, E''_{e,1}) = (\rho'_{e,2}, v_{e,2}, E''_{e,2})$
From (76), (I.5), (42), and (68),
(I.6) $d'_{e,1} = d'_{e,2}$
From (I.5), (I.6), (43), and (69),
(I.7) $\mathcal{S}'_1 = \mathcal{S}'_2$
From (16), (31), and (57),
(I.8) $\Sigma'_1 \approx_l^p \Sigma'_2$
From (23) and (49),
(I.9) $E_1 = E_2$
From (16), (I.9), (24), (50), and Lemma 19,
(I.10) $\mathsf{ks}''_1 \approx_l^p \mathsf{ks}''_2$
From (46) and (72),
(I.11) $E_{d,1} \downarrow_l^p = E_{d,2} \downarrow_l^p$ or $\mathsf{ks}_{d,1} \downarrow_l^p = \mathsf{ks}_{d,2} \downarrow_l^p = \cdot$
From (16), (I.11), (26), (52), and Lemma 20,
(I.12) $\mathsf{ks}_{d,1} \approx_l^p \mathsf{ks}_{d,2}$
From (40) and (66),
(I.13) $E'_{e,1} = E'_{e,2}$
From (I.1), (I.5), (I.13), (44), and (70),
(I.14) $E_{e,1} = E_{e,2}$
From (I.1), (I.14), (28), and (54),
(I.15) $\mathsf{ks}_{e,1} = \mathsf{ks}_{e,2}$
From (46) and (72),
(I.16) $E_{m,1} \downarrow_l^p = E_{m,2} \downarrow_l^p$
From (16), (I.16), (30), (56), and Lemma 20,
(I.17) $\mathsf{ks}_{m,1} \approx_l^p \mathsf{ks}_{m,2}$
From (32), (58), (I.10), (I.12), (I.15), and (I.17),
(I.18) $\mathsf{ks}'_1 \approx_l^p \mathsf{ks}'_2$
From (46), (72), (I.5), and (I.15),
$T_1 \approx_l^p T_2$
From (I.4), (I.7), (I.8), and (I.18),
$K'_1 \approx_l^p K'_2$

Case II: $pc \downarrow^p \not\sqsubseteq l$
From (75), (35), and (61),
(II.1) $(\rho'_{d,1}, v_{d,1}, E''_{d,1}) = (\rho'_{d,2}, v_{d,2}, E''_{d,2})$
From (75), (II.1), (36), and (62),
(II.2) $d'_{d,1} = d'_{d,2}$
From (II.1), (II.2), (37), and (63),

(II.3) $\mathcal{R}_1' = \mathcal{R}_2'$

From (76), (41), and (67),

(II.4) $(\rho_{e,1}', v_{e,1}, E_{e,1}'') = (\rho_{e,2}', v_{e,2}, E_{e,2}'')$

From (76), (II.4), (42), and (68),

(II.5) $d_{e,1}' = d_{e,2}'$

From (II.4), (II.5), (43), and (69),

(II.6) $\mathcal{S}_1' = \mathcal{S}_2'$

From (16), (31), and (57),

(II.7) $\Sigma_1' \approx_l^p \Sigma_2'$

From (23) and (49),

(II.8) $E_1 = E_2$

From (16), (II.8), (24), (50), and Lemma 19,

(II.9) $\mathsf{ks}_1'' \approx_l^p \mathsf{ks}_2''$

From (46) and (72),

(II.10) $E_{d,1} \downarrow_l^p = E_{d,2} \downarrow_l^p$ or $\mathsf{ks}_{d,1} \downarrow_l^p = \mathsf{ks}_{d,2} \downarrow_l^p = \cdot$

From (16), (II.10), (26), (52), and Lemma 20,

(II.11) $\mathsf{ks}_{d,1} \approx_l^p \mathsf{ks}_{d,2}$

By assumption and from (41), (67), $p = c$, and since $\mathcal{E}$ will only change l_c to be more secret,

(II.12) $E_{e,1}'' \downarrow_l^p = E_{e,2}'' \downarrow_l^p = \cdot$

By assumption and from (40), (66), and $p = c$,

(II.13) $E_{e,1'} \downarrow_l^p = E_{e,2}' \downarrow_l^p = \cdot$

From (II.12), (II.13), (44), (70), and the definition of transparent,

(II.14) $E_{e,1} \downarrow_l^p = E_{e,1} \downarrow_l^p = \cdot$

From (28), (54), (II.14), and Lemma 21,

(II.15) $\mathsf{ks}_{e,1} \downarrow_l^p = \mathsf{ks}_{e,2} \downarrow_l^p = \cdot$

From (46) and (72),

(II.16) $E_{m,1} \downarrow_l^p = E_{m,2} \downarrow_l^p$

From (16), (II.16), (30), (56), and Lemma 20,

(II.17) $\mathsf{ks}_{m,1} \approx_l^p \mathsf{ks}_{m,2}$

From (32), (58), (II.9), (II.11), (II.15), and (II.17),

(II.18) $\mathsf{ks}_1' \approx_l^p \mathsf{ks}_2'$

By assumption and from (46), (72), $p = c$, and the definition of trTransparent,

$T_1 \approx_l^p T_2$

From (II.3), (II.6), (II.9), and (II.18),

$K_1' \approx_l^p K_2'$

$\square$