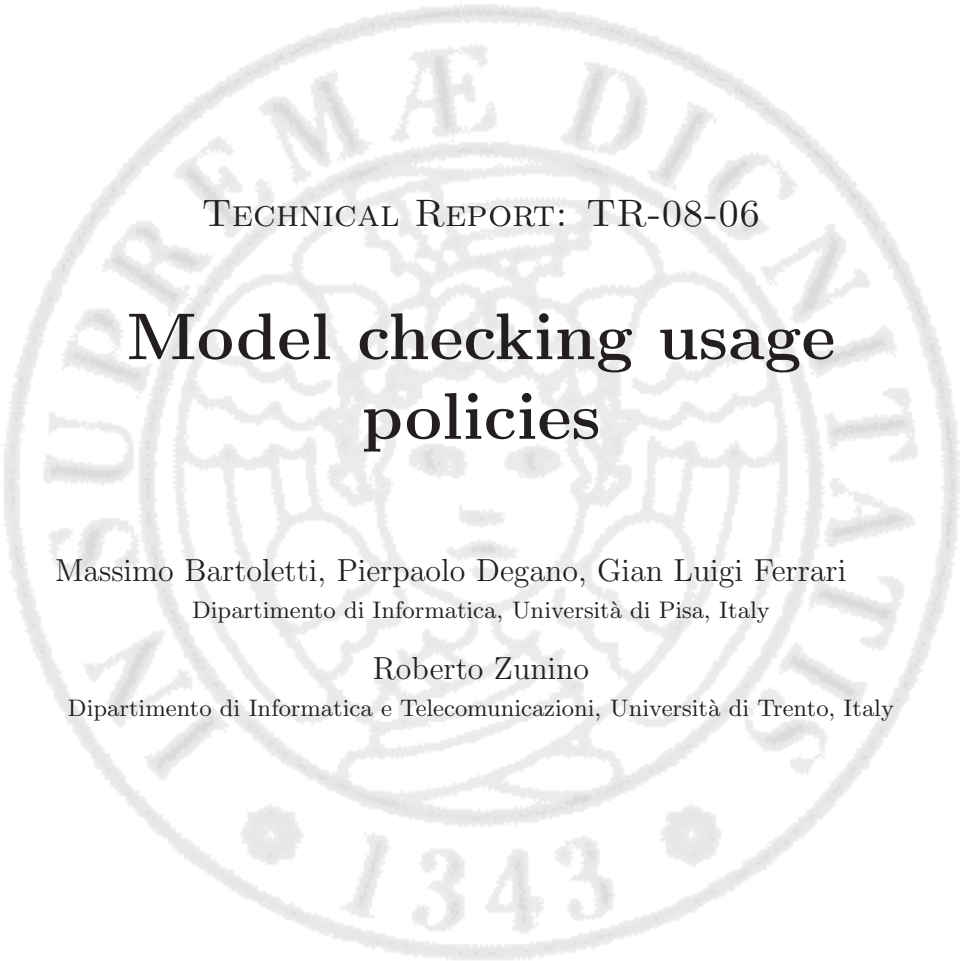
Università di Pisa
Dipartimento di Informatica



Technical Report: TR-08-06

Model checking usage policies

Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari
Dipartimento di Informatica, Università di Pisa, Italy

Roberto Zunino
Dipartimento di Informatica e Telecomunicazioni, Università di Trento, Italy

published 4/4/08 – revised 30/6/08

address: Largo B. Pontecorvo 3, 56127 Pisa, Italy. tel: +39 050 2212700 fax: +39 050 2212726

Model checking usage policies

Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari
Dipartimento di Informatica, Università di Pisa, Italy

Roberto Zunino
Dipartimento di Informatica e Telecomunicazioni, Università di Trento, Italy

published 4/4/08 – revised 30/6/08

Abstract

We propose a model for specifying, analysing and enforcing safe usage of resources. Our usage policies allow for parametricity over resources, and they can be enforced through finite state automata. The patterns of resource access and creation are described through a basic calculus of usages. In spite of the augmented flexibility given by resource creation and by policy parametrization, we devise an efficient (polynomial-time) model-checking technique for deciding when a usage is resource-safe, i.e. when it complies with all the relevant usage policies.

1 Introduction

A fundamental concern of security is to ensure that resources are used correctly. Devising expressive, flexible and efficient mechanisms to control resource usages is therefore a major issue in the design and implementation of security-aware programming languages. The problem is made even more crucial by the current programming trends, which provide for reusing code, and exploiting services and components, offered by (possibly untrusted) third parties. It is indeed common practice to pick from the Web some scripts, or plugins, or packages, and assemble them into a bigger program, with little or no control about the security of the whole. To cope with this situation, we proposed in [3] *local policies*, that formalise and enhance the concept of *sandbox*, while being more flexible than global policies and local checks spread over program code. Local policies smoothly allow for safe composition of programs with their own security requirements, also in mobile code scenarios, and they can drive call-by-contract composition of services [4, 6].

Our contribution is twofold. First, we propose a model for local usage policies. Our policies are quite general: in the spirit of history-based security [1], they can inspect the whole trace of security-relevant events generated by a running program. Unlike e.g. [18], our policies are not hard-wired to resources, yet they are *parametric* over resources. For instance, a policy $\varphi(x, y)$ means that

for all x and y the obligation expressed by φ must be obeyed. This is particularly relevant in mobile code scenarios, where you need to impose constraints on how external programs access the resources created in your local environment, without being able to alter the code (e.g. to insert local security checks). We prove that run-time enforcement of our policies is possible through finite state automata.

The second contribution is a model-checking technique to statically detect when a program violates the relevant local policies. The patterns of resource access and creation are described by a calculus of *usages*, which can be automatically inferred by a static analysis of programs [5]. We then devise a technique to decide whether a given usage, locally annotated with policies, respects them in every possible execution. Since programs may create an arbitrary number of fresh resources, this may give rise to an infinite number of formulae to be inspected while checking a policy. We solve this problem by suitably abstracting resources so that only a finite number of cases needs to be considered. This allows us to extract from a usage a Basic Process Algebra and a regular formula, to be used in model-checking [15]. The proposed technique correctly and completely handles the case in which the number of fresh resources generated at run-time has no bound known at static time. Our algorithm runs in polynomial time on the size of the checked usage and policies.

Examples. To illustrate our model, we present some examples of real-world usage policies, which are particularly relevant to the field of security.

Information flow. Consider a Web application that allows for editing documents, storing them on a remote site, and sharing them with other users. The editor is implemented as an applet run by a local browser. The user can tag any of her documents as *private*. To avoid direct information flows, the policy requires that private files cannot be sent to the server in plain text, yet they can be sent encrypted. This policy is modelled by $\varphi_{\mathsf{IF}}(x)$ in Fig. 1, left. After having tagged the file x as private (edge from q_0 to q_1), if x were to be sent to the server (edge from q_1 to q_2), then the policy would be violated: the double circle around q_2 marks it as an offending state. Instead, if x is encrypted (edge from q_1 to q_3), then x can be freely transmitted: indeed, the absence of paths from q_3 to an offending state indicates that once in q_3 the policy will not be violated on file x. A further policy is applied to our editor, to avoid information flow due to covert channels. It requires that, after reading a private file, any other file must be encrypted before it can be transmitted. This is modelled by $\varphi_{\mathsf{CC}}(x, y)$ in Fig. 1, right. A violation occurs if after some private file x is read (path from q'_0 to q'_2), then some other file y is sent (edge from q'_2 to the offending state q'_4).

Chinese Wall. A classical security policy used in commercial and corporate business services is the Chinese Wall policy [11]. In such scenarios, it is usual to assume that all the objects which concern the same corporation are grouped together into a *company dataset*, e.g. $bank_A, bank_B, oil_A, oil_B$, etc. A *conflict of interest class* groups together all company datasets whose corporations are in

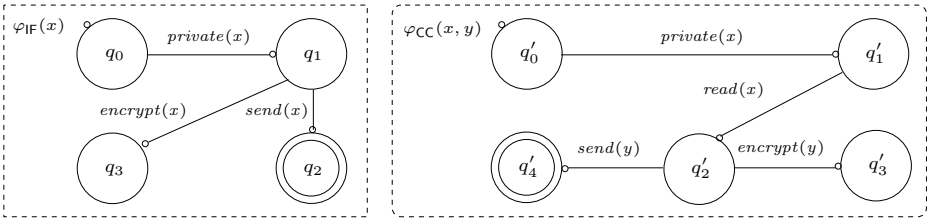


Figure 1: The information flow policy $\varphi_{\mathsf{IF}}(x)$ and the covert channels policy $\varphi_{\mathsf{CC}}(x, y)$.

competition, e.g. $Bank$ containing $bank_A, bank_B$ and Oil containing oil_A, oil_B. The Chinese Wall policy then requires that accessing an object is only permitted in two cases. Either the object is in the same company dataset as an object already accessed, or the object belongs to a different conflict of interest class. E.g., the trace $read(oil_A, Oil)\, read(bank_A, Bank)\, read(oil_B, Oil)$ violates the policy, because reading an object in the dataset oil_B is not permitted after having accessed oil_A, which is in the same conflict of interests class Oil. The Chinese Wall policy is specified by $\varphi_{\mathsf{CW}}(x, y)$ in Fig. 2, left. The edge from q_0 to q_1 represents accessing the company dataset x in the conflict of interests class y. The edge leading from q_1 to the offending state q_2 means that a dataset different from x (written as $\bar{x}$) has been accessed in the same conflict of interests class y.

Applet confinement. As a further example, consider the case of a Web browser which can run applets. Assume that an applet needs to create files on the local disk, e.g. to save and retrieve status information. Direct information flows are avoided by denying applets the right to access local files. Also, to prevent from interference, applets are not allowed to access files created by other applets. This behaviour is modelled by the policy $\varphi_{\mathsf{AC}}(f, a)$ in Fig. 2, right. The edge from q'_0 to q'_1 represents the applet a creating a file f. Accessing f is denied to any applet other than a (modelled by $\bar{a}$) through the edge from q'_1 to the offending state q'_2. The edge from q_0 to q'_2 prohibits accessing local files.

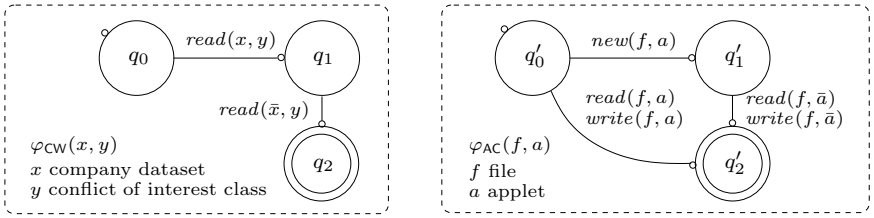


Figure 2: The Chinese Wall policy $\varphi_{\mathsf{CW}}(x, y)$ and the Applet Isolation policy $\varphi_{\mathsf{AI}}(f, a)$.

The paper is organized as follows. We first introduce our usage policies, and we show them enforceable through finite state automata. We then define a calculus of usages, and we characterize when a usage is valid. Our model checking technique follows, together with the main results, i.e. its correctness,

completeness and polynomial time complexity. We conclude by presenting some possible extensions, and we discuss some related work. For simplicity we shall only present the formal treatment of monadic usage policies (i.e. those with one formal parameter). The extension to the polyadic case $\varphi(x_1, \ldots, x_n)$ is quite direct; at each step of our formal development, we shall give the intuition for adapting our theory to the polyadic case. We have designed and implemented a tool for model checking usage policies [7]. The tool is written in Haskell, and it is quite general: it supports for all of the features presented in this paper, including polyadic policies and events. We have tested and experimented our tool with the case studies presented in this paper, and also in some more complex ones. The results and the performance are consistent with those anticipated by the theoretical results presented in this paper.

2 Usage policies

We start by introducing the needed syntactic categories. *Resources* $r, r', \ldots \in \mathsf{Res} = \mathsf{Res}_s \cup \mathsf{Res}_d$ are objects that can either be already available in the environment (*static*, included in the finite set Res_s), or be freshly created at run-time (*dynamic*, Res_d). We assume a distinguished resource $? \notin \mathsf{Res}$ to play the role of an "unknown" one (typically, ? will result from static approximations). Resources can be accessed through a given finite set of *actions* $\alpha, \alpha', \mathit{new}, \ldots \in \mathsf{Act}$. An *event* $\alpha(r) \in \mathsf{Ev}$ abstracts from accessing the resource r through the action α. The special actions *new* represent the creation of a fresh resource; this means that for each dynamically created resource r, the event $\mathit{new}(r)$ must precede any other $\alpha(r)$. In our examples, we sometimes use polyadic events such as $\alpha(r_1, r_2)$. These events do not increase the expressivity of our model, since the can be simulated by e.g. a sequence $\alpha_1(r_1)\alpha_2(r_2)$. A *trace* is a finite sequence of events, typically denoted by $\eta, \eta', \ldots \in \mathsf{Ev}^*$. For notational convenience, when the target resource of an action α is immaterial, we stipulate that α acts on some special (static) resource, and we write just α for the event.

Usage policies (Def. 1) constrain the usage of resources to obey a regular property. For instance, a file usage policy $\varphi(x)$ might require that "before reading or writing a file x, that file must have been opened, and not closed by the while". A usage policy gives rise to an automaton with finite states, named *policy automaton*, when the formal parameter x is instantiated to an actual resource r. Policy automata will be exploited to recognize those traces obeying φ.

Edges in a usage policy can be of three kinds: either $\vartheta = \alpha(r)$ for a static resource r, or $\vartheta = \alpha(x)$, or $\vartheta = \alpha(\bar{x})$, where $\bar{x}$ means "different from x". The extension to the polyadic case is as follows: the edges of a policy $\varphi(x, y)$ can mention static resources, the parameters x and y, and a special symbol $\bar{*}$ denoting "different from x and y". Quite notably, polyadic policies (unlike polyadic events) increase the expressivity of our model e.g. the policies in Fig. 2 cannot be expressed with a single parameter.

Given $r \in \mathsf{Res}$ and a set of resources $\mathcal{R}$, a usage policy $\varphi(x)$ is instantiated

Definition 1. Usage policies

A usage policy $\varphi(x)$ *is a 5-tuple* $\langle S, Q, q_0, F, E\rangle$*, where:*

- $S \subset \mathsf{Act} \times (\mathsf{Res}_s \cup \{x, \bar{x}\})$ *is the input alphabet,*
- Q *is a* finite *set of states,*
- $q_0 \in Q \setminus F$ *is the start state,*
- $F \subset Q$ *is the set of final "offending" states,*
- $E \subseteq Q \times S \times Q$ *is a finite set of edges, written* $q \overset{\vartheta}{\multimap} q'$

into a policy automaton $A_{\varphi(r,\mathcal{R})}$ by binding x to the resource r and, accordingly, making $\bar{x}$ range over each resource in $\mathcal{R} \setminus \{r\}$ (see Def. 2). The auxiliary relation $\ddot{\delta}$ (i) instantiates x to the given resource r, (ii) instantiates $\bar{x}$ with all $r' \neq r$, (iii) maintains the transitions $\alpha(r')$ for r' static. Then. the relation $\dot{\delta}$ adds self-loops for all the events not explicitly mentioned in the policy. Finally, we derive δ by adding transitions to deal with ?, that represents resources not known at static time. Intuitively, any transition $q \xrightarrow{\alpha(r)} q'$ can also be played by $\alpha(?)$. Note that policy automata are non-deterministic, e.g. because of the transitions labelled $\alpha(?)$. Given a trace η and a policy φ, we want that *all* the paths of the instances of $\varphi(x)$ comply with φ. This is a form of diabolic (or internal) non-determinism. To account for that, we make the "offending" states as final — thus going into a final state represents a violation of the policy, while the other states mean compliance to the policy. The construction of Def. 2 can be easily adapted to polyadic policies such as $\varphi(x, y)$, yielding the automata $A_{\varphi(r,r',\mathcal{R})}$, where x is instantiated to r, y to r', and $\bar{*}$ to $\mathcal{R} \setminus \{r, r'\}$.

According to Def. 2, to check $\eta \models \varphi$ we should instantiate *infinitely* many policy automata (one for each $r \in \mathsf{Res}$). These automata have a finite number of states, but they may have *infinitely* many transitions, e.g. instantiating the abstract edge $q \overset{\alpha(\bar{x})}{\multimap} q'$ originates $q \xrightarrow{\alpha(r)} q'$ for all but one $r \in \mathsf{Res}$. Similarly for self-loops. Interestingly enough, the compliance of a trace with a policy is a decidable property: the next lemma shows that checking $\eta \models \varphi$ can be decided through a *finite* set of *finite* state automata.

Lemma 3. *Let* η *be a trace, let* φ *be a policy, and let* $\mathsf{R}(\eta)$ *be the set of resources occurring in* η*. Then,* $\eta \models \varphi$ *if and only if* $\eta \lhd A_{\varphi(r,\mathsf{R}(\eta))}$ *for each* $r \in \mathsf{R}(\eta) \cup \{r'\}$*, where* r' *is an arbitrary resource in* $\mathsf{Res} \setminus \mathsf{R}(\eta)$*.*

When polyadic policies are used, e.g. $\varphi(x, y)$, then Lemma 3 should be changed to require $\eta \lhd A_{\varphi(r,r',\mathsf{R}(\eta))}$ for each $r, r' \in \mathsf{R}(\eta) \cup \{r''\}$, where r'' is an arbitrary resource in $\mathsf{Res} \setminus \mathsf{R}(\eta)$.

Definition 2. Policy automata and policy compliance

Let $\varphi(x) = \langle S, Q, q_0, F, E\rangle$ *be a usage policy, let* $r \in \mathsf{Res}$*, and let* $R \subseteq \mathsf{Res}$*. The* usage automaton $A_{\varphi(r,R)} = \langle \Sigma, Q, q_0, F, \delta\rangle$ *is defined as follows:*

$$\Sigma = \{\, \alpha(r') \mid \alpha \in \mathsf{Act} \text{ and } r' \in R \,\}$$
$$\delta = \dot\delta \cup \{\, q \xrightarrow{\alpha(?)} q' \,|\, \exists \zeta \in R : q \xrightarrow{\alpha(\zeta)} q' \in \dot\delta \,\} \qquad (\textit{unknown resources})$$
$$\dot\delta = \ddot\delta \cup \{\, q \xrightarrow{\alpha(r')} q \,|\, r' \in R, \nexists q' : q \xrightarrow{\alpha(r')} q' \in \ddot\delta \,\} \qquad (\textit{self-loops})$$

where the relation $\ddot\delta$ *is defined as follows:*

$$\begin{aligned}
\ddot\delta \;=\; & \{\, q \xrightarrow{\alpha(r)} q' \mid q \xrightarrow{\alpha(x)}\!\circ\; q' \in E \,\} && (\textit{instantiation of } x)\\
\cup\; & \textstyle\bigcup_{r' \in (R \cup \{?\}) \setminus \{r\}} \{\, q \xrightarrow{\alpha(r')} q' \mid q \xrightarrow{\alpha(\bar x)}\!\circ\; q' \in E \,\} && (\textit{instantiation of } \bar x)\\
\cup\; & \{\, q \xrightarrow{\alpha(r')} q' \mid q \xrightarrow{\alpha(r')}\!\circ\; q' \in E \,\} && (\textit{static resources})
\end{aligned}$$

We write $\eta \lhd A_{\varphi(r,\mathcal{R})}$ *when* η *is not in the language of the automaton* $A_{\varphi(r,\mathcal{R})}$*. We say that* η respects φ*, written* $\eta \models \varphi$*, when* $\eta \lhd A_{\varphi(r,\mathsf{Res})}$*, for all* $r \in \mathsf{Res}$*. Otherwise, we say that* η violates φ*, written* $\eta \not\models \varphi$*.*

Example 1. *Consider the policy requiring that, in a history* $\alpha(r_1)\alpha(r_2)\alpha(r_3)$*, the three resources* r_0, r_1, r_2 *are actually distinct. This is modelled by the usage policy* $\varphi_3(x, y)$ *in Fig. 3. A straightforward generalisation of the above allows for keeping distinct* k *resources, using a policy with arity* $k - 1$*.* □

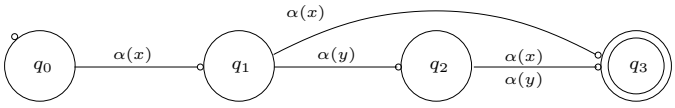


Figure 3: The usage policy $\varphi_3(x, y)$ allows for keeping distinct three resources.

Example 2. *To justify the extra resource of Lemma 3, let* $\varphi_{\neg\alpha}(x)$ *be the policy* $\langle\{\alpha(\bar x)\}, \{q_0, q_1\}, q_0, \{q_1\}, \{q_0 \xrightarrow{\alpha(\bar x)}\!\circ\; q_1\}\rangle$*. When a trace* η *contains some event* $\alpha(r)$*, the instantiation* $A_{\varphi_{\neg\alpha}(r',\mathsf{Res})}$ *recognizes* η *as offending, for all* $r' \neq r$ *– so* $\varphi_{\neg\alpha}(x)$ *actually forbids any* α *actions. Consider e.g.* $\eta = \alpha(r_0)\,\beta(r_0)$*. Although* $\eta \lhd A_{\varphi(r,\mathsf{R}(\eta))}$ *for all* $r \in \mathsf{R}(\eta) = \{r_0\}$*, as noted above* η *violates* $\varphi_{\neg\alpha}$*, which motivates Lemma 3 checking* $A_{\varphi(r',\mathsf{R}(\eta))}$ *also for some* $r' \in \mathsf{Res} \setminus \mathsf{R}(\eta)$*. Finally, note that replacing* $\alpha(\bar x)$ *with* $\alpha(x)$ *in the policy would not affect trace validity, because* x *is implicitly universally quantified in the definition of validity.* □

Example 3. *Consider a trojan-horse applet that maliciously attempts to send spam e-mail through the mail server of the browser site. To do that, the applet first connects to the site it was downloaded from, and it implements some apparently useful and harmless activity. Then, the applet inquires the local host to*

check if relaying of e-mails is allowed: if so, it connects to the local SMTP server to send unsolicited bulk e-mails. To protect from such malicious behaviour, the browser prevents the applet from connecting to two different URLs. This can be enforced by sandboxing the applet with the policy $\varphi_{\mathsf{Spam}}(x)$ *defined in Fig. 3, left. E.g., consider the trace* $\eta = start\, connect(u_0)\, stop\, start\, connect(u_1)\, connect(u_2)$. *The policy automaton* $A_{\varphi_{\mathsf{Spam}}(u_1,\mathcal{R})}$, *with* $\mathcal{R} = \{u_0, u_1, u_2\}$, *is in Fig. 3, right (the self-loops for events other than* $start, stop$ *and* $connect$ *are omitted). Since* η *drives an offending run in the policy automaton* $A_{\varphi_{\mathsf{Spam}}(u_1,\mathcal{R})}$, *then* $\eta \not\models \varphi_{\mathsf{Spam}}$. *This correctly models the fact that the policy* φ_{Spam} *prevents applets from connecting to two different sites. Observe that removing the last* $connect(u_2)$ *event from* η *would make the trace obey* φ_{Spam}, *since the stop event resets the policy* φ_{Spam} *to the starting state. Actually,* $start\, connect(u_0)\, stop\, start\, connect(u_1)$ *is* not *in the language of any* $A_{\varphi_{\mathsf{Spam}}(r,\mathcal{R}')}$, *for all* $r \in \mathsf{Res}$ *and* $\mathcal{R}' \subseteq \mathsf{Res}$. □

3 A calculus for usage control

We now introduce a basic calculus for usage control (the abstract syntax of our calculus is in Def. 4). Usages are *history expressions* [5], that inherit from Basic Process Algebras (BPAs, [10]) sequential composition, non-deterministic choice, and recursion (though with a slightly different syntax). Quite differently from BPAs, our atomic actions have a parameter which indicates the resource upon which the action is performed; more precisely, our atomic actions are the events from Sec. 2. We also have events the target of which is a *name* $n, n', \ldots \in \mathsf{Nam}$. In $\nu n.U$, the ν acts as a binder of the free occurrences of the name n in U. The intended meaning is to keep track of the binding between n and a freshly created resource. In a recursion $\mu h.U$, the free occurrences of h in U are bound by μ. A usage is *closed* when it has no free names and variables, and it is *initial* when closed and with no dynamic resources. The sandboxing construct $\varphi[U]$ asserts that each step of the execution of U must obey the policy φ. Aiming at minimality, it is convenient to treat $\varphi[U]$ as syntactic sugar for $[_\varphi \cdot U \cdot]_\varphi$, where $[_\varphi$ and $]_\varphi$ are special *framing* events that represent opening/closing of the scope of φ. We assume that such framing events are disjoint from those in Ev.

The behaviour of a usage (Def. 5) is defined as the set of sequential traces of its events. As usual, ε denotes the empty trace, and $\varepsilon\eta = \eta = \eta\varepsilon$. The trace

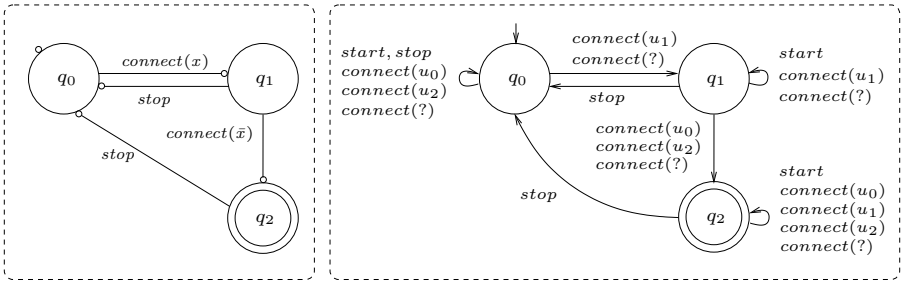


Figure 4: The policy $\varphi_{\mathsf{Spam}}(x)$ (left) and the policy automaton $A_{\varphi_{\mathsf{Spam}}(u_1,\{u_0,u_1,u_2\})}$ (right).

Definition 4. Usages

U, U'	$::=$	ε	*empty*	
		h	*variable*	
		$\alpha(\rho)$	*event*	$(\rho \in \mathsf{Res} \cup \mathit{Nam} \cup \{?\})$
		$U \cdot V$	*sequence*	
		$U + V$	*choice*	
		$\nu n.U$	*resource creation*	
		$\mu h.\, U$	*recursion*	
		$\varphi[U]$	*sandboxing*	$(\varphi[U] = [_\varphi \cdot U \cdot]_\varphi)$

semantics is defined through a labelled transition relation $U, \mathcal{R} \xrightarrow{a} U', \mathcal{R}'$, where $a \in \mathsf{Ev} \cup \{\varepsilon\}$. The set $\mathcal{R}$ in configurations accumulates the resources created at run-time, so that no resource can be created twice. We assume that $\mathsf{Res}_s \subseteq \mathcal{R}$, to prevent dynamically created resources from colliding with static ones.

Definition 5. Trace semantics of usages

$$\alpha(r),\, \mathcal{R} \xrightarrow{\alpha(r)} \varepsilon,\, \mathcal{R} \qquad \nu n.\, U,\, \mathcal{R} \xrightarrow{\varepsilon} U\{r/n\},\, \mathcal{R} \cup \{r\} \quad \textit{if } r \in \mathsf{Res}_d \setminus \mathcal{R}$$

$$\varepsilon \cdot U,\, \mathcal{R} \xrightarrow{\varepsilon} U,\, \mathcal{R} \qquad U \cdot V,\, \mathcal{R} \xrightarrow{a} U' \cdot V,\, \mathcal{R}' \quad \textit{if } U,\, \mathcal{R} \xrightarrow{a} U',\, \mathcal{R}'$$

$$U + V,\, \mathcal{R} \xrightarrow{\varepsilon} U,\, \mathcal{R} \qquad U + V,\, \mathcal{R} \xrightarrow{\varepsilon} V,\, \mathcal{R} \qquad \mu h.\, U,\, \mathcal{R} \xrightarrow{\varepsilon} U\{\mu h.\, U/h\},\, \mathcal{R}$$

Example 4. *Let $U = \mu h.\, \nu n.\, (\varepsilon + \mathit{new}(n) \cdot \alpha(n) \cdot h)$. Given a starting $\mathcal{R}$, the traces of U are the prefixes of the strings with form $\mathit{new}(r_1)\alpha(r_1) \cdots \mathit{new}(r_k)\alpha(r_k)$ for all $k \geq 0$ and pairwise distinct resources r_i such that $r_i \notin \mathcal{R}$.* □

A trace is *well-formed* when (i) no static resource is the target of a *new* event, (ii) no *new* is fired twice on the same resource, and (iii) no event $\alpha(r)$, with r dynamic and $\alpha \neq \mathit{new}$, is fired without a prior $\mathit{new}(r)$. Hereafter, we shall only consider usages U with well-formed traces. We conjecture this is a decidable property of usages, e.g. suitably adapting the techniques of [17] should enable us to identify and discard those U that produce non well-formed traces.

We now define when a trace respects all the relevant usage policies, i.e. when the trace is *valid* (Def. 6). For example, let $\eta = \mathit{private}(f)\, \mathit{read}(f)\, [_{\varphi_{\mathsf{Ed}}}\, \mathit{send}(f)\,]_{\varphi_{\mathsf{Ed}}}$ where φ_{Ed} is the Editor policy of Sect. 1. Then, η is *not* valid, because the *send* event occurs within a framing enforcing φ_{Ed}, and $\mathit{private}(f)\, \mathit{read}(f)\, \mathit{send}(f)$ does not obey φ_{Ed}. Note that we check the *whole* past, and not just the $\mathit{send}(f)$ event within the framing, as we follow the history–based security approach. This makes our policies more expressive than those that can only look at the part of the history enclosed within the framing events.

Definition 6. Active policies and validity

The multiset $act(\eta)$ of the active policies of a trace η is defined as follows:

$$act(\varepsilon) = \{\!|\,|\!\} \qquad act(\eta\,[_\varphi) = act(\eta) \cup \{\!|\varphi|\!\}$$
$$act(\eta\,\alpha(\rho)) = act(\eta) \qquad act(\eta\,]_\varphi) = act(\eta) \setminus \{\!|\varphi|\!\}$$

A trace η is valid *when $\models \eta$, defined inductively as follows:*

$$\models \varepsilon \qquad \models \eta'\beta \quad \text{if} \quad \models \eta' \text{ and } (\eta'\beta)^{-[]} \models \varphi \text{ for all } \varphi \in act(\eta'\beta)$$

where $\eta^{-[]}$ is the trace η depurated from all the framing events.
A usage U is valid *when, for all $U', \mathcal{R}, \mathcal{R}'$ and η: $U, \mathcal{R} \xrightarrow{\eta} U', \mathcal{R}' \implies \models \eta$.*

Validity of traces is a prefix-closed (i.e., safety) property (Lemma 7), yet it is not compositional: in general, $\models \eta$ and $\models \eta'$ do not imply $\models \eta\eta'$. This is a consequence of the assumption that no past events can be hidden (see Ex. 7).

Lemma 7. *For all traces η and η', if $\eta\eta'$ is valid, then η is valid.*

Example 5. *Let $\eta = [_\varphi \alpha_1 [_{\varphi'} \alpha_2]_{\varphi'} \alpha_3$. The active policies of η are as follows:*

$$\begin{aligned} act(\eta) &= act([_\varphi \alpha_1 [_{\varphi'} \alpha_2]_{\varphi'}) = act([_\varphi \alpha_1 [_{\varphi'} \alpha_2) \setminus \{\!|\varphi'|\!\} = act([_\varphi \alpha_1 [_{\varphi'}) \setminus \{\!|\varphi'|\!\} \\ &= (act([_\varphi \alpha_1) \cup \{\!|\varphi'|\!\}) \setminus \{\!|\varphi'|\!\} = act([_\varphi \alpha_1) = act([_\varphi) = \{\!|\varphi|\!\} \end{aligned}$$

Thereaore, η is valid if and only if: $\varepsilon \models \varphi$, $\alpha_1 \models \varphi$, $\alpha_1\alpha_2 \models \varphi$, $\alpha_1\alpha_2\alpha_3 \models \varphi$, $\alpha_1 \models \varphi'$, and $\alpha_1\alpha_2 \models \varphi'$. □

Example 6. *Consider the policy φ_{Loan} that prevents from taking out a loan if your account is in the red: $\langle \{red, black\}, \{q_0, q_1\}, q_0, \{q_1\}, \{q_0 \xrightarrow{red}\!\circ\, q_1, q_1 \xrightarrow{black}\!\circ\, q_0\}\rangle$. Let $\eta = red\ black\,[_{\varphi_{\mathsf{Loan}}}$, which is valid. Indeed, we have that $red\ black \models \varphi_{\mathsf{Loan}}$ – while the prefix red is not required to respect the policy φ_{Loan} (so matching the intuition that one can recover from a red balance and obtain a loan). Note that, differently from validity, the relation $\eta \models \varphi$ is* not *prefix-closed. So, we can define policies which, as in this example, permit to recover from bad states.* □

Example 7. *Consider the trace $\eta = \alpha[_\varphi\, \alpha\,]_\varphi \alpha$, where the policy φ requires that α is not executed three times. Since $\alpha \models \varphi$ and $\alpha\alpha \models \varphi$, then η is valid. Note that the first α is checked, even though it is outside of the framing: since it happens in the past, our policies can inspect it. Instead, the third α occurs after the framing has been closed, therefore it is not checked. Now, consider the trace $\eta' = \alpha\eta$. In spite of both α and η being valid, their composition η' is not. To see why, consider the trace $\bar{\eta} = \alpha\alpha[_\varphi\, \alpha$, which is a prefix of η'. Then, $act(\bar{\eta}) = \{\!|\varphi|\!\}$, but $\bar{\eta}^{-[]} = \alpha\alpha\alpha \not\models \varphi$. This shows that validity is not compositional.* □

We specify in Def. 8 a transformation of policy automata that makes them able to recognize valid traces. This is done by instrumenting a policy automaton $A_{\varphi(r,\mathcal{R})}$ with framing events, so obtaining a *framed* usage automaton $A_{\varphi_{[]}(r,\mathcal{R})}$ that will recognize those traces that are valid with respect to the policy φ.

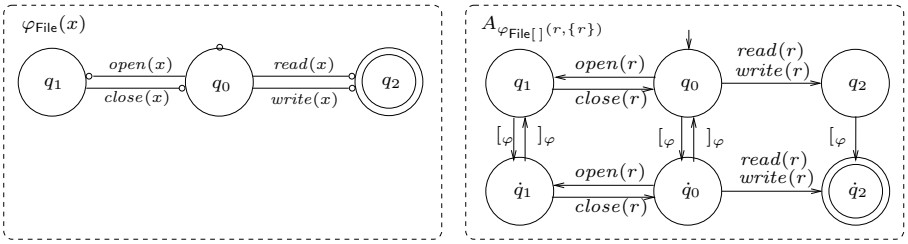


Figure 5: The file usage policy $\varphi_{\mathsf{File}}(x)$ and the framed usage automaton $A_{\varphi_{\mathsf{File}}(r,\{r\})}$.

Definition 8. Instrumenting policy automata with framing events

Let $A_{\varphi(r,\mathcal{R})} = \langle \Sigma, Q, q_0, F, \delta \rangle$ *be a policy automaton. Then, we define* $A_{\varphi_{[\,]}(r,\mathcal{R})} = \langle \Sigma', Q', q_0, F', \delta' \rangle$ *as follows:* $\Sigma' = \Sigma \cup \{[_\varphi,]_\varphi, [_{\varphi'},]_{\varphi'}, \ldots\}$, $Q' = Q \cup \{\, \dot{q} \mid q \in Q \,\}$, $F' = \{\, \dot{q} \mid q \in F \,\}$, *and:*

$$\delta' = \delta \;\cup\; \{\, q \xrightarrow{[_\varphi} \dot{q} \mid q \in Q \,\} \;\cup\; \{\, \dot{q} \xrightarrow{]_\varphi} q \mid q \in Q \setminus F \,\} \;\cup\; \{\, \dot{q} \xrightarrow{\vartheta} \dot{q} \mid \dot{q} \in F' \,\}$$
$$\cup\, \{\, \dot{q} \xrightarrow{\vartheta} \dot{q}' \mid q \xrightarrow{\vartheta} q' \in \delta \text{ and } q \in Q \setminus F \,\} \;\cup\; \{\, q \xrightarrow{[_\psi} q, q \xrightarrow{]_\psi} q \mid \psi \neq \varphi \,\}$$

Intuitively, the automaton $A_{\varphi_{[\,]}(r,\mathcal{R})}$ is partitioned into two layers. Both are copies of $A_{\varphi(r,\mathcal{R})}$, but in the first layer of $A_{\varphi_{[\,]}(r,\mathcal{R})}$ all the states are made non-final. This represents being compliant with φ. The second layer is reachable from the first one when opening a framing for φ, while closing gets back – unless we are in a final (i.e. offending) state. The transitions in the second layer are a copy of those in $A_{\varphi(r,\mathcal{R})}$, the only difference being that the final states are sinks. The final states in the second layer are exactly those final in $A_{\varphi(r,\mathcal{R})}$. Note that we shall only consider traces without "redundant" framings, i.e. of the form $\eta\,[_\varphi\,\eta'\,[_\varphi$ with $]_\varphi \notin \eta'$. In [3] we defined a static transformation of usages that removes these redundant framings. For instance, $\varphi[U \cdot \varphi[U']]$ is rewritten as $\varphi[U \cdot U']$ since the inner $\varphi[\ldots]$ is redundant. Hereafter, we assume that usages have been undergone to this transformation (minor modifications of [3] suffice).

Example 8. *Consider the file usage policy* $\varphi_{\mathsf{File}}(x)$ *in Fig. 5 (left), requiring that only open files can be read or written. The initial state* q_0 *represents the file being closed, while* q_1 *is for an open file. Reading and writing* x *in* q_0 *leads to the offending state* q_2*, while in* q_1 *you can read and write* x*. The instrumentation* $A_{\varphi_{\mathsf{File}[\,]}(r,\{r\})}$ *of* $A_{\varphi_{\mathsf{File}}(r,\{r\})}$ *is in Fig. 5 (right) – the self-loops are omitted.* □

We now relate framed usage automata with validity. A trace η (which has no redundant framings, after the assumed transformation) is valid if and only if it complies with the framed automata $A_{\varphi_{[\,]}(r,\mathsf{R}(\eta))}$ for all the policies φ spanning over η. The adaptation of Lemma 9 to polyadic policies proceeds as for Lemma 3.

Lemma 9. *A trace* η *is valid if and only if* $\eta \lhd A_{\varphi_{[\,]}(r,\mathsf{R}(\eta))}$*, for all* φ *occurring in* η*, and for all* $r \in \mathsf{R}(\eta) \cup \{r'\}$*, where* r' *is an arbitrary resource in* $\mathsf{Res} \setminus \mathsf{R}(\eta)$*.*

4 Model checking validity of usages

We statically verify the validity of usages by model-checking Basic Process Algebras with policy automata. Note that the arbitrary nesting of framings and the infinite alphabet of resources make validity *non-regular*, e.g. the usage $\mu h.\, \nu n.\, new(n) \cdot \alpha(n) + h \cdot h + \varphi[h]$ has traces with unbounded pairs of balanced $[_\varphi$ and $]_\varphi$ and unbounded number of symbols - so it is *not* a regular language. This prevents us from directly applying the standard decision technique for verifying that a BPA P satisfies a regular property φ, i.e. checking the emptiness of the pushdown automaton resulting from the conjunction of P and the finite state automaton recognizing $\neg\varphi$.

To cope with the first source of non-regularity – due to the arbitrary nesting of framings – we use the static transformation of usages that removes the redundant framings [3]. For the second source of non-regularity, due to the ν-binders, the major challenge for verification is that usages may create fresh resources, while BPAs cannot. A naïve solution could lead to the generation of an unbounded set of automata $A_{\varphi(r)}$ that must be checked to verify validity. For example, the traces denoted by $U = \varphi[\mu h.\, (\varepsilon + \nu n.\, new(n) \cdot \alpha(n) \cdot h)]$ must satisfy all the policies $\varphi(r_0), \varphi(r_1), \ldots$ for each fresh resource. Thus, we would have to intersect an infinite number of finite state automata to verify U valid, which is unfeasible.

To this purpose, we shall define a mapping from usages to BPAs that reflects and preserves validity. Our mapping groups freshly created resources in just two categories. The intuition is that any policy $\varphi(x)$ can only distinguish between x and all the other resources, represented by $\bar{x}$. There is no way for $\varphi(x)$ to further discriminate among the dynamic resources. Thus, it is sound to consider only two representatives of dynamic resources: the "witness" resource # that represents x, and the "don't care" resource _ for $\bar{x}$.

The transformation from usages U into BPAs $\mathsf{B}(U)$ is given in Def. 10. The syntax of a BPA P and its trace semantics $[\![P]\!]$ are standard; for reference, we include them in the Appendix (Defs. 14 and 15). Events, variables, concatenation and choice are mapped by $\mathsf{B}(U)$ into the corresponding BPA counterparts. A usage $\mu h.U$ is mapped to a fresh BPA variable X, bound to the translation of U in the set of definitions Δ. The crucial case is that of new name generation $\nu n.U$, which is dealt with two rules. If $d = \_$, then we generate a

Definition 10. Mapping usages to BPAs

The BPA associated with a usage U is defined as $\mathsf{B}(U) = \mathsf{B}_\_(U)_\emptyset$, where $\mathsf{B}_d(U)_\Theta$, inductively defined below, takes as input a usage U and a function Θ from variables h to BPA variables X. The parameter d can either be _ or #.

$$\mathsf{B}_d(\varepsilon)_\Theta = \langle 0, \emptyset \rangle \qquad \mathsf{B}_d(h)_\Theta = \langle \Theta(h), \emptyset \rangle \qquad \mathsf{B}_d(\alpha(\rho))_\Theta = \langle \alpha(\rho), \emptyset \rangle$$

$$\mathsf{B}_d(U \cdot V)_\Theta = \mathsf{B}_d(U)_\Theta \cdot \mathsf{B}_d(V)_\Theta \qquad \mathsf{B}_d(U + V)_\Theta = \mathsf{B}_d(U)_\Theta + \mathsf{B}_d(V)_\Theta$$

$$\mathsf{B}_\_(\nu n.U)_\Theta = \mathsf{B}_\_(U\{\_/n\})_\Theta + \mathsf{B}_\#(U\{\#/n\})_\Theta \qquad \mathsf{B}_\#(\nu n.U)_\Theta = \mathsf{B}_\#(U\{\_/n\})_\Theta$$

$$\mathsf{B}_d(\mu h.U)_\Theta = \langle X, \Delta \cup \{X \triangleq p\} \rangle \text{ where } \langle p, \Delta \rangle = \mathsf{B}_d(U)_{\Theta\{X/h\}},\ X \text{ fresh}$$

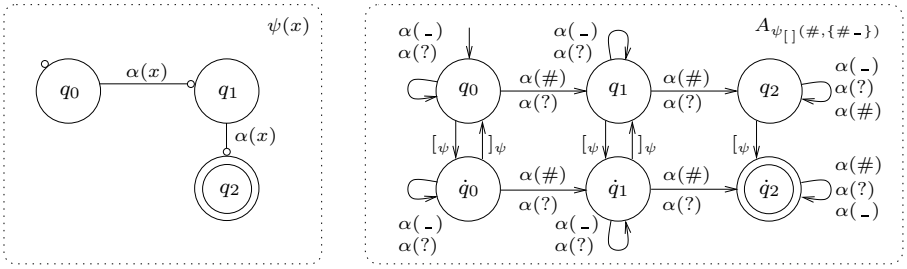


Figure 6: The policy $\psi(x)$ and its instantiation $A_{\psi_{[]}(\#,\{\#,\_\})}$ (some self-loops omitted).

choice between two BPA processes: in the first, the name n is replaced by the "don't care" resource $\_$, while in the second, n is replaced by the "witness" resource $\#$. If $d = \#$, this means that we have already witnessed $\#$, so we proceed by generating $\_$.

Theorem 12 below states the correspondence between usages and BPAs. The traces of a usage are all and only the strings that label the computations of the extracted BPA, where resources are suitably renamed to permit model-checking.

Definition 11. *Let σ be a substitution from $\mathsf{Res} \cup \{?\}$ to $\mathsf{Res} \cup \{?\}$. We say that σ is* name-collapsing *when, for some set of dynamic resources $\mathcal{R} \subset \mathsf{Res}_d$:*

$$\sigma(r) = \# \quad \text{if } r \in \mathcal{R} \qquad \sigma(\#) = \# \qquad \sigma(?) = ?$$
$$\sigma(r) = \_ \quad \text{if } r \in \mathsf{Res}_d \setminus \mathcal{R} \qquad \sigma(\_) = \_ \qquad \sigma(r) = r \quad \text{otherwise}$$

We call such σ uniquely-collapsing *if $\sigma(r) = \#$ for exactly one $r \neq \#$.*

Theorem 12. *For each initial usage U, trace η, and name-collapsing σ:*

$$U, \mathcal{R} \xrightarrow{\eta} U', \mathcal{R}' \implies \exists P \,:\, \mathsf{B}(U) \xrightarrow{\eta\sigma} P$$

Example 9. *Let $U = \mu h.\,(\nu n.\,\varepsilon + new(n) \cdot \alpha(n) \cdot h)$. Then, $\mathsf{B}(U) = \langle X, \Delta \rangle$, where $\Delta = \{X \triangleq 0 + new(\_) \cdot \alpha(\_) \cdot X + 0 + new(\#) \cdot \alpha(\#) \cdot X\}$. Consider the trace $\eta = new(r_0)\alpha(r_0)new(r_1)\alpha(r_1)new(r_2)\alpha(r_2)$ of U. Let $\sigma = \{\_/r_0, \#/r_1, \_/r_2\}$. Then, $\eta\sigma = new(\_)\alpha(\_)new(\#)\alpha(\#)new(\_)\alpha(\_)$ is a trace in $[\![\langle X, \Delta \rangle]\!]$.* □

Example 10. *Let $U = \psi[(\nu n.\, new(n) \cdot \alpha(n)) \cdot (\nu n'.\, new(n') \cdot \alpha(n')) \cdot \alpha(?)]$ where the policy ψ asks that the action α cannot be performed twice on the same resource (left-hand side of Fig. 6). Then:*

$$\mathsf{B}(U) = [_\psi \cdot \big(new(\_) \cdot \alpha(\_) + new(\#) \cdot \alpha(\#)\big) \cdot \big(new(\_) \cdot \alpha(\_) + new(\#) \cdot \alpha(\#)\big) \cdot \alpha(?) \cdot]_\psi$$

The BPA $\mathsf{B}(U)$ violates $A_{\psi_{[]}(\#,\{\_,\#\})}$, displayed in right-hand side of Fig. 6 (where we have omitted all the self-loops for the new action). The violation is consistent with the fact that the wildcard ? represents any resource, e.g. $new(r')\alpha(r')new(r)\alpha(r)\alpha(r)$ is a trace of U that violates ψ. Although U is valid whenever $\mathsf{B}(U)$ is valid, such verification technique would not be complete. Consider e.g. ψ' requiring that α is not executed three *times on the same resource, and let $U' = \psi'[(\nu n.\, new(n) \cdot \alpha(n)) \cdot (\nu n'.\, new(n') \cdot \alpha(n')) \cdot \alpha(?)]$. Note that $\mathsf{B}(U')$ violates $\psi'_{[]}(\#)$, while all the traces denoted by U' respect ψ'. Theorem 13 below provides us with a sound and complete verification technique.* □

As shown in Ex. 10, the validity of U does not imply the validity of $\mathsf{B}(U)$, so leading to a sound but incomplete decision procedure. The problem is that $\mathsf{B}(U)$ uses the same "witness" resource # for all the resources created in U. This leads to violations of policies, e.g. those that prevent some action from being performed twice (or more) on the same resource, because $\mathsf{B}(U)$ identifies (as #) resources that are distinct in U. To recover a (sound and) complete decision procedure for validity, it suffices to check any trace of $\mathsf{B}(U)$ only *until* the second "witness" resource is generated (i.e. before the second occurrence of $new(\#)$). This is accomplished by composing $\mathsf{B}(U)$ with the "unique witness" automaton through a "weak until" operator. The weak until W is standard; the unique witness $A_{\#}$ is a finite state automaton that reaches an offending state on those traces containing more than one $new(\#)$ event (see Def. 16 for details).

Example 11. *Consider again the usage U' in Ex. 10. The maximal traces generated by $\mathsf{B}(U')$ are shown below.*

$$[_{\psi'}\ new(\_)\alpha(\_)\ new(\#)\alpha(\#)\ \alpha(?)\ ]_{\psi'} \qquad [_{\psi'}\ new(\#)\alpha(\#)\ new(\_)\alpha(\_)\ \alpha(?)\ ]_{\psi'}$$
$$[_{\psi'}\ new(\_)\alpha(\_)\ new(\_)\alpha(\_)\ \alpha(?)\ ]_{\psi'} \qquad [_{\psi'}\ new(\#)\alpha(\#)\ new(\#)\alpha(\#)\ \alpha(?)\ ]_{\psi'}$$

The first three traces above comply with $A_{\psi'_{[\,]}(\#,\{\#,\_\})}$, which instead is violated by the last trace. Indeed, in $\eta = [_{\psi'}\ new(\#)\alpha(\#)\ new(\#)\alpha(\#)\ \alpha(?)\ ]_{\psi'}$ the two fresh resources are identical, so contrasting with the semantics of usages. To avoid incompleteness, η is not considered in model-checking, since $\eta \lhd (A_{\psi'_{[\,]}(\#,\{\#,\_\})}\ \mathsf{W}\ A_{\#})$, i.e. η is accepted by $A_{\#}$ and filtered out by the weak until. Note also that there is no need to consider the automaton $A_{\psi'_{[\,]}(\_,\{\#,\_\})}$, because if a violation has to occur, it will occur on $A_{\psi'_{[\,]}(\#,\{\#,\_\})}$. □

The following theorem enables us to efficiently verify the validity of a usage U by (i) extracting the BPA $\mathsf{B}(U)$ from U, and (ii) model-checking $\mathsf{B}(U)$ against a finite set of finite state automata.

Theorem 13. *Let $\Phi(U) = \{\, A_{\varphi_{[\,]}(r_0,\mathsf{R}(U))} \mid r_0, \varphi \in U \,\} \cup \{\, A_{\varphi_{[\,]}(\#,\mathsf{R}(U))} \mid \varphi \in U \,\}$, where $\mathsf{R}(U)$ comprises #, _, and all the static resources occurring in U.*

(a) *An initial usage U is valid if and only if, for all $A_{\varphi_{[\,]}(r,\mathcal{R})} \in \Phi(U)$:*

$$[\![\mathsf{B}(U)]\!] \lhd A_{\varphi_{[\,]}(r,\mathcal{R})}\ \mathsf{W}\ A_{\#}$$

(b) *The computational complexity of this method is PTIME in the size U.*

Valid usages are recognized by checking all $A_{\varphi_{[\,]}(r,\mathcal{R})}$ in $\Phi(U)$. The set $\Phi(U)$ contains, for each policy φ and static resource r_0, the framed usage automaton $A_{\varphi_{[\,]}(r_0,\mathsf{R}(U))}$. Also, $\Phi(U)$ includes the instantiations $A_{\varphi_{[\,]}(\#,\mathsf{R}(U))}$, to be ready on controlling φ on dynamic resources, represented by the witness #.

Example 12. *Let $U = \mu h.\,(\varepsilon + \nu n.\, new(n) \cdot \alpha(n) \cdot h)$. Consider then $U_\varphi = \varphi[U]$, where $\varphi(x)$ asks that, for each resource x, the first occurrence of the event $\alpha(x)$ is necessarily followed by another $\alpha(x)$ (Fig. 7a). Then, U_φ is* not *valid, because e.g. $\eta = [_\varphi new(r)\alpha(r)new(r')\alpha(r')$ is a trace of U_φ, and $\eta \not\models A_{\varphi_{[\,]}(r,\{r,r'\})}$ (the*

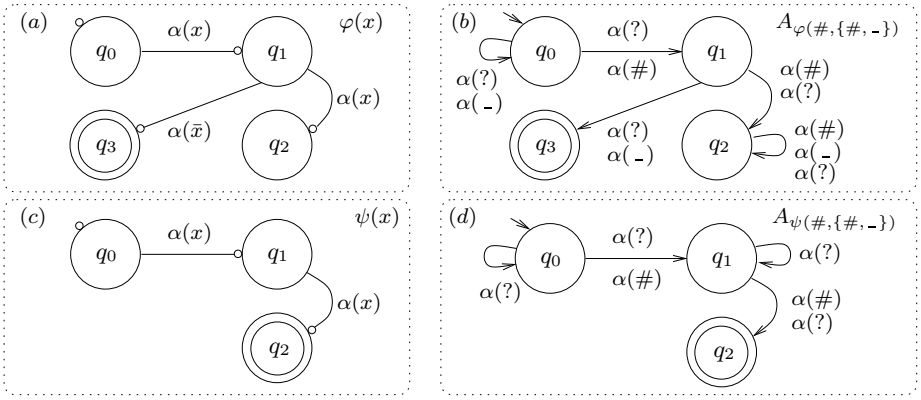


Figure 7: The usage policies $\varphi(x)$ and $\psi(x)$ and the policy automata $A_{\varphi(\#,\{\#,\_\})}$ and $A_{\psi(\#,\{\#,\_\})}$. The self-loops for the *new* events are omitted.

non-framed policy automaton is in Fig. 7b). So, $\Phi(U_\varphi) = \{A_{\varphi_{[\,]}(\#,\{\#,\_\})}\}$*, and:*

$$\mathsf{B}(U_\varphi) = \langle [_\varphi \cdot X \cdot]_\varphi, X \triangleq \varepsilon + (new(\#) \cdot \alpha(\#) \cdot X) + (new(\_) \cdot \alpha(\_) \cdot X) \rangle$$

and the trace $[_\varphi new(\#)\alpha(\#)new(\_)\alpha(\_) \in [\![\mathsf{B}(U_\varphi)]\!]$ *drives the framed automaton* $A_{\varphi_{[\,]}(\#,\{\#,\_\})}$ *to an offending state. Consider now* $U_\psi = \psi[U]$*, where the policy* $\psi(x)$ *says that the action* α *cannot be performed twice on the same resource* x *(Fig. 7c). We have that* $\Phi(U_\psi) = \{A_{\psi_{[\,]}(\#,\{\#,\_\})}\}$*, and:*

$$\mathsf{B}(U_\psi) = \langle [_\psi \cdot X \cdot]_\psi, X \triangleq \varepsilon + (new(\#) \cdot \alpha(\#) \cdot X) + (new(\_) \cdot \alpha(\_) \cdot X) \rangle$$

Although U_ψ *obeys* ψ*, the BPA does not, since* $[_\psi new(\#)\alpha(\#)new(\#)\alpha(\#)]_\psi$ *violates* $A_{\psi_{[\,]}(\#,\{\#,\_\})}$ *(the non-framed instantiation* $A_{\psi(\#,\{\#,\_\})}$ *is in Fig. 7d). Completeness is recovered through the weak until and unique witness automata, that filter the traces, like the one above, where* $new(\#)$ *is fired twice.* □

Example 13. *Recall from Ex. 8 the policy* φ_{File} *(only open files can be read/written) and consider the policy* φ_{DoS} *that forbids the creation of more than* k *files. Let:*

$$U = \varphi_{\mathsf{File}}[\varphi_{\mathsf{DoS}}[\mu h.\, \varepsilon + \nu n.\, new(n) \cdot open(n) \cdot read(n) \cdot close(n) \cdot h]]$$

Then, $\Phi(U) = \{\varphi_{\mathsf{File}}(\#), \varphi_{\mathsf{DoS}}(\#)\}$*, and* $\mathsf{B}(U) = \langle [_\varphi \cdot [_{\varphi_{\mathsf{DoS}}} \cdot X \cdot]_{\varphi_{\mathsf{DoS}}} \cdot]_\varphi, \Delta \rangle$*, where* Δ *comprises the following definition:*

$$\begin{aligned} X \triangleq\ & \varepsilon + (new(\_) \cdot open(\_) \cdot read(\_) \cdot close(\_) \cdot X) \\ & + (new(\#) \cdot open(\#) \cdot read(\#) \cdot close(\#) \cdot X) \end{aligned}$$

Note that each computation of $\mathsf{B}(U)$ *obeys* $A_{\varphi_{\mathsf{File}[\,]}(\#,\{\#,\_\})}$*, while there exist computations that violate* $A_{\varphi_{\mathsf{DoS}[\,]}(\#,\{\#,\_\})}$*.* □

To handle polyadic policies (say with arity k), our technique can be adjusted as follows. We use k witnesses $\#_1, \ldots, \#_k$. The set $\Phi(U)$ is computed as $\Phi(U) = \{\, A_{\varphi_{[\,]}(r_1,\ldots,r_k,\mathsf{R}(U))} \mid \varphi \in U,\ \forall i.\ r_i \in U \vee r_i \in \{\#_1, \ldots, \#_k\} \,\}$. Moreover, now $A_\#$ must check that each $\#_i$ has at most an associated *new* event. The transformation of usages into BPAs should then be modified as follows:

$$\begin{aligned}
\mathsf{B}_{_}(\nu n.U)_\Theta &= \mathsf{B}_{_}(U\{_/n\})_\Theta + \mathsf{B}_{\#_1}(U\{\#_1/n\})_\Theta \\
\mathsf{B}_{\#_i}(\nu n.U)_\Theta &= \mathsf{B}_{\#_i}(U\{_/n\})_\Theta + \mathsf{B}_{\#_{i+1}}(U\{\#_{i+1}/n\})_\Theta \qquad \text{if } i < k \\
\mathsf{B}_{\#_k}(\nu n.U)_\Theta &= \mathsf{B}_{\#_k}(U\{_/n\})_\Theta
\end{aligned}$$

Theorem 13 still holds. The complexity of model-checking is still PTIME in the size of U, and EXPTIME in k. Note that being EXPTIME in k has no big impact in practice, as one expects k to be very small even for complex policies.

5 Conclusions

We proposed a model for policies that control the usage of resources. Usage policies can be enforced through finite state automata. A basic calculus of usages was presented to describe the patterns of resource access and creation, and obligation to respect usage policies. We call a usage *valid* when *all* its possible traces comply with *all* the relevant usage policies. In spite of the augmented flexibility given by resource creation and by policy parametrization, we devised an efficient (polynomial-time) and complete model-checking technique for deciding the validity of usages. Our technique manages to represent the generation of an unbounded number of resources in a finitary manner. Yet, we do not lose the possibility of verifying interesting security properties of programs.

Local usage policies were originally introduced in [5]. The current work thoroughly improves and simplifies them. Usage policies can now have an arbitrary number of parameters, which augments their expressive power and allows for modelling significant real-world policies (see Sec. 1). The model checking technique in [5] has exponential time complexity, while the current one is polynomial.

Extensions. A first simple extension to our model is to assign a *type* to resources. To do that, the set Act of actions is partitioned to reflect the types of resources (e.g. $\mathsf{Act} = \mathsf{File} \cup \mathsf{Socket} \cup \cdots$), where each element of the partition contains the actions admissible for the given type (e.g. $\mathsf{File} = \{new_{\mathsf{File}}, open, close, read, write\}$, where new_{T} represents creating a new resource of type T). The syntax of the ν-constructor is extended with the type, i.e. $\nu n : T.\, U$. Validity should now check that the actions fired on a resource also respect its type. Our model checking technique can be smoothly adapted by using a #-witness for each type (e.g. $\#_{\mathsf{File}}, \#_{\mathsf{Socket}}$, *etc.*), to be dealt with the corresponding "unique witness" automata (e.g. $A_{\#_{\mathsf{File}}}, A_{\#_{\mathsf{Socket}}}$, *etc.*). The time complexity is unaltered.

The language of usages has two simple extensions. The first consists in attaching policies to resources upon creation, similarly to [18]. The construct $\nu n : \varphi.\, U$ is meant to enforce the policy φ on the freshly created resource. An encoding into the existing constructs is possible. First, the whole usage has to be sandboxed with the policy $\varphi_\nu(x)$, obtained from $\varphi(x)$ by adding a new initial state q_ν and an edge labelled $check(x)$ from q_ν to the old initial state. The encoding then transforms $\nu n : \varphi.\, U$ into $\nu n.\, check(n) \cdot U$. The second extension consists in allowing parallel usages $U|V$. Model checking is still possible by

transforming usages into Basic Parallel Processes [12] instead of BPAs. However, the time complexity becomes exponential in the number of parallel branches [20].

Related work. Many authors [13, 14, 19, 22] mix static and dynamic techniques to transform programs and make them obey a given policy. Our model allows for local, polyadic policies and events parametrized over dynamically created resources, while the above-mentioned papers only consider global policies and no parametrized events. Polymer [9] is a language for specifying, composing and enforcing (global) security policies. In the lines of *edit automata* [8], a policy can intervene in the program trace, to insert or suppress some events. Policy composition can then be unsound, because the events inserted by a policy may interfere with those monitored by another policy. To cope with that, the programmer must explicitly fix the order in which policies are applied. Being Turing-equivalent, Polymer policies are more expressive than ours, but this gain in expressivity has some disadvantages. First, a policy may potentially be unable to decide whether an event is accepted or not (i.e. it may not terminate). Second, no static guarantee is given about the compliance of a program with the imposed policy. Run-time monitoring is then necessary to enforce the policy, while our model-checking tecnique may avoid this overhead. A typed λ-calculus is presented in [18], with primitives for creating and accessing resources, and for defining their permitted usages. Type safety guarantees that well-typed programs are resource-safe, yet no effective algorithm is given to check compliance of the inferred usages with the permitted ones. The policies of [18] can only speak about the usage of *single* resources, while ours can span over many resources, e.g. a policy requiring that no socket connections can be opened after a local file has been read. A further limitation of [18] is that policies are attached to resources. In mobile code scenarios, e.g. a browser that runs untrusted applets, it is also important that you can impose constraints on how external programs manage the resources created in your local environment. E.g., an applet may create an unbounded number of resources on the browser site, and never release them, so leading to denial-of-service attacks. Our local policies can deal with this kind of situations. *Shallow history automata* are investigated in [16]. These automata can keep track of the *set* of past access events, rather than the *sequence* of events. Although shallow history automata can express some interesting security properties, they are clearly less expressive than our usage policies. A model for history-based access control is proposed in [23]. It uses control-flow graphs enriched with permissions and a primitive to check them, similarly to [2]. The run-time permissions are the intersection of the static permissions of all the nodes visited in the past. The model-checking technique can decide in EXPTIME (in the number of permissions) if all the permitted traces of the graph respect a given regular property on its nodes. Unlike our local policies, that can enforce any regular policy on traces, the technique of [23] is less general, because there is no way to enforce a policy unless it is encoded as a suitable assignment of permissions to nodes.

References

[1] M. Abadi and C. Fournet. Access control based on execution history. In *Proc. 10th Annual Network and Distributed System Security Symposium*, 2003.

[2] M. Bartoletti, P. Degano, and G. L. Ferrari. Static analysis for stack inspection. In *International Workshop on Concurrency and Coordination*, 2001.

[3] M. Bartoletti, P. Degano, and G. L. Ferrari. History based access control with local policies. In *Proc. Fossacs*, 2005.

[4] M. Bartoletti, P. Degano, and G. L. Ferrari. Types and effects for secure service orchestration. In *Proc. 19th CSFW*, 2006.

[5] M. Bartoletti, P. Degano, G. L. Ferrari, and R. Zunino. Types and effects for resource usage analysis. In *Proc. Fossacs*, 2007.

[6] M. Bartoletti, P. Degano, G. L. Ferrari, and R. Zunino. Semantics-based design for secure web services. *IEEE Transactions on Software Engineering*, 34(1), 2008.

[7] M. Bartoletti and R. Zunino. LocUsT: a tool for checking usage policies. Technical Report TR-08-07, Dip. Informatica, Univ. Pisa, 2008.

[8] L. Bauer, J. Ligatti, and D. Walker. More enforceable security policies. In *Foundations of Computer Security (FCS)*, 2002.

[9] L. Bauer, J. Ligatti, and D. Walker. Composing security policies with Polymer. In *Proc. PLDI*, 2005.

[10] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985.

[11] D. F. C. Brewer and M. J. Nash. The chinese wall security policy. In *Proc. of Symp. on Security and Privacy*, 1989.

[12] S. Christensen. *Decidability and Decomposition in Process Algebras*. PhD thesis, Edinburgh University, 1993.

[13] T. Colcombet and P. Fradet. Enforcing trace properties by program transformation. In *Proc. 27th ACM SIGPLAN-SIGACT PoPL*, 2000.

[14] Ú. Erlingsson and F. B. Schneider. SASI enforcement of security policies: a retrospective. In *Proc. 7th New Security Paradigms Workshop*, 1999.

[15] J. Esparza. On the decidability of model checking for several μ-calculi and Petri nets. In *Proc. 19th Int. Colloquium on Trees in Algebra and Programming*, 1994.

[16] P. W. Fong. Access control by tracking shallow execution history. In *IEEE Symposium on Security and Privacy*, 2004.

[17] A. Igarashi and N. Kobayashi. Type reconstruction for linear -calculus with i/o subtyping. *Inf. Comput.*, 161(1):1–44, 2000.

[18] A. Igarashi and N. Kobayashi. Resource usage analysis. In *Proc. 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2002.

[19] K. Marriott, P. J. Stuckey, and M. Sulzmann. Resource usage verification. In *Proc. First Asian Programming Languages Symposium*, 2003.

[20] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems.* PhD thesis, Technischen Universität München, 1998.

[21] F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50, 2000.

[22] P. Thiemann. Enforcing safety properties using type specialization. In *Proc. ESOP*, 2001.

[23] J. Wang, Y. Takata, and H. Seki. HBAC: A model for history-based access control and its model checking. In *Proc. ESORICS*, 2006.

A Auxiliary definitions

Definition 14. Syntax of Basic Process Algebras

A BPA process *is given by the following abstract syntax, where* $\beta \in \mathsf{Ev}$*:*

$$p, p' \;::=\; 0 \mid \beta \mid p \cdot p' \mid p + p' \mid X$$

A BPA definition *has the form* $X \triangleq p$*. A set* Δ *of BPA definitions is* coherent *when* $X \triangleq p \in \Delta$ *and* $X \triangleq p' \in \Delta$ *imply* $p = p'$*. A* BPA *is a pair* $\langle p, \Delta \rangle$ *such that* (i) Δ *is finite and coherent, and* (ii) *for all* X *occurring in* p *or* Δ*, there exists some* q *such that* $X \triangleq q \in \Delta$*. We write* $\Delta + \Delta'$ *for* $\Delta \cup \Delta'$*, when coherent (otherwise,* $\Delta + \Delta'$ *is undefined). For all* $P = \langle p, \Delta \rangle$ *and* $P' = \langle p', \Delta' \rangle$*, we write* $P \cdot P'$ *for* $\langle p \cdot p', \Delta + \Delta' \rangle$*, and* $P + P'$ *for* $\langle p + p', \Delta + \Delta' \rangle$*.*

Definition 15. LTS for Basic Process Algebras

The semantics $[\![P]\!]$ *of a BPA* $P = \langle p_0, \Delta \rangle$ *is the set of the histories labelling finite computations* [a] $\{ \eta = a_1 \cdots a_i \mid p_0 \xrightarrow{a_1} \cdots \xrightarrow{a_i} p_i \}$*, where* $a \in \mathsf{Ev} \cup \{\varepsilon\}$*, and the relation* $\xrightarrow{a}$ *is inductively defined by the following rules:*

$$0 \cdot p \xrightarrow{\varepsilon} p \qquad \beta \xrightarrow{\beta} 0 \qquad p + q \xrightarrow{\varepsilon} p \qquad p + q \xrightarrow{\varepsilon} q$$

$$p \cdot q \xrightarrow{a} p' \cdot q \quad \text{if } p \xrightarrow{a} p' \qquad X \xrightarrow{\varepsilon} p \quad \text{if } X \triangleq p \in \Delta$$

[a] Since validity is a safety property [21] – nothing bad can happen in any (finite) trace of execution steps – it is sound to consider finite computations only.

Definition 16. Weak until and Unique Witness automata

Let $A_0 = \langle \Sigma, Q_0, \bar{q}_0, F_0, \rho_0 \rangle$ *and* $A_1 = \langle \Sigma, Q_1, \bar{q}_1, F_1, \rho_1 \rangle$ *be usage automata. Assume that* A_0 *and* A_1 *are* complete*, i.e. for each state* q *and* $\beta \in \Sigma$*, there exists a transition from* q *labelled* β*. The* weak until *automaton* $A_0 \,\mathsf{W}\, A_1$ *is defined as* $\langle \Sigma, Q_0 \times Q_1, \langle \bar{q}_0, \bar{q}_1 \rangle, F_0 \times (Q_1 \setminus F_1), \rho_W \rangle$*, where:*

$$\rho_W = \{ \langle q_0, q_1 \rangle \xrightarrow{\vartheta} \langle q_0', q_1' \rangle \mid q_0 \xrightarrow{\vartheta} q_0' \in \rho_0,\ q_1 \xrightarrow{\vartheta} q_1' \in \rho_1,\ q_1 \in Q_1 \setminus F_1 \}$$
$$\cup \{ \langle q_0, q_1 \rangle \xrightarrow{\vartheta} \langle q_0, q_1 \rangle \mid q_1 \in F_1, \vartheta \in \Sigma \}$$

The unique witness *automaton* $A_\#$ *is* $\langle \Sigma, \{q_0, q_1, q_2\}, q_0, \{q_2\}, \rho_\# \rangle$*, where:*

$$\rho_\# = \{q_0 \xrightarrow{new(\#)} q_1,\ q_1 \xrightarrow{new(\#)} q_2\}$$
$$\cup \{ q_0 \xrightarrow{\vartheta} q_0,\ q_1 \xrightarrow{\vartheta} q_1 \mid \vartheta \in \Sigma \setminus \{new(\#)\} \} \cup \{ q_2 \xrightarrow{\vartheta} q_2 \mid \vartheta \in \Sigma \}$$

B Proofs

Lemma B.1. *For each trace η, policy φ, and resource r:*

$$\exists \mathcal{R} \supseteq \mathsf{R}(\eta).\ \eta \lhd A_{\varphi(r,\mathcal{R})} \implies \forall \mathcal{R}' \supseteq \mathsf{R}(\eta).\ \eta \lhd A_{\varphi(r,\mathcal{R}')}$$

Proof. We prove the contrapositive. Assume that there exists $\mathcal{R}' \supseteq \mathsf{R}(\eta)$ such that $\eta \not\lhd A_{\varphi(r,\mathcal{R}')}$, i.e. there is an offending run $q_0 \xrightarrow{\eta} q_k$ of $A_{\varphi(r,\mathcal{R}')}$. For all $\mathcal{R} \supseteq \mathsf{R}(\eta)$, we show how to reconstruct an offending run of $A_{\varphi(r,\mathcal{R})}$ on η.

Let $\delta_{\mathcal{R}}, \dot{\delta}_{\mathcal{R}}, \ddot{\delta}_{\mathcal{R}}$ and $\delta_{\mathcal{R}'}, \dot{\delta}_{\mathcal{R}'}, \ddot{\delta}_{\mathcal{R}'}$ be the sets of transitions (see Def. 2) of the policy automata $A_{\varphi(r,\mathcal{R})}$ and $A_{\varphi(r,\mathcal{R}')}$, respectively.

First, we show that, for each $r' \in \mathsf{R}(\eta)$, $q \xrightarrow{\alpha(r')} q'$ in $\ddot{\delta}_{\mathcal{R}'}$ if and only if $q \xrightarrow{\alpha(r')} q'$ in $\ddot{\delta}_{\mathcal{R}}$. Consider an edge $q \xrightarrow{\vartheta} \!\!\circ\ q'$ of the usage policy $\varphi(x)$. There are the following exhaustive cases:

- if $\vartheta = \alpha(r')$ with $r' \in \mathsf{Res}_s$, then $q \xrightarrow{\alpha(r')} q'$ is in $\ddot{\delta}_{\mathcal{R}'}$, as well as in $\ddot{\delta}_{\mathcal{R}}$.
- if $\vartheta = \alpha(x)$, then $q \xrightarrow{\alpha(r)} q'$ is in $\ddot{\delta}_{\mathcal{R}'}$, as well as in $\ddot{\delta}_{\mathcal{R}}$.
- if $\vartheta = \alpha(\bar{x})$, then $q \xrightarrow{\alpha(r')} q'$ is in $\ddot{\delta}_{\mathcal{R}'}$ for all $r' \in (\mathcal{R}' \cup \{?\}) \setminus \{r\}$, and it is in $\ddot{\delta}_{\mathcal{R}}$ for all $r' \in (\mathcal{R} \cup \{?\}) \setminus \{r\}$. Since $r' \in \mathsf{R}(\eta)$, $\mathcal{R}' \supseteq \mathsf{R}(\eta)$, and $\mathcal{R} \supseteq \mathsf{R}(\eta)$, then $q \xrightarrow{\alpha(r')} q'$ is in $\ddot{\delta}_{\mathcal{R}'}$ as well as in $\ddot{\delta}_{\mathcal{R}}$.

Let $r' \in \mathsf{R}(\eta)$. The above and $\mathsf{R}(\eta) \subseteq \mathcal{R}, \mathcal{R}'$ imply $q \xrightarrow{\alpha(r')} q'$ in $\dot{\delta}_{\mathcal{R}'}$ if and only if $q \xrightarrow{\alpha(r')} q'$ in $\dot{\delta}_{\mathcal{R}}$, which in turn implies the thesis. □

Lemma 3. For all traces η and policies φ:

$$\eta \models \varphi \iff \exists r' \in \mathsf{Res} \setminus \mathsf{R}(\eta).\ \forall r \in \mathsf{R}(\eta) \cup \{r'\}.\ \eta \lhd A_{\varphi(r,\mathsf{R}(\eta))}$$

Proof. For the "only if" part, assume that $\eta \models \varphi$, i.e. that, for all $\bar{r} \in \mathsf{Res}$, $\eta \lhd A_{\varphi(\bar{r},\mathsf{Res})}$. Choose any $r' \in \mathsf{Res} \setminus \mathsf{R}(\eta)$. Given $r \in \mathsf{R}(\eta) \cup \{r'\}$, we have $\eta \lhd A_{\varphi(r,\mathsf{Res})}$. By Lemma B.1, this implies $\eta \lhd A_{\varphi(r,\mathsf{R}(\eta))}$.

For the "if" part, we prove the contrapositive. Assume that $\eta \not\models \varphi$, i.e. there exists some $\bar{r} \in \mathsf{Res}$ such that $\eta \not\lhd A_{\varphi(\bar{r},\mathsf{Res})}$. We now prove that for all $r' \in \mathsf{Res} \setminus \mathsf{R}(\eta)$ there is an $r \in \mathsf{R}(\eta) \cup \{r'\}$ such that $\eta \not\lhd A_{\varphi(r,\mathsf{R}(\eta))}$. If $\bar{r} \in \mathsf{R}(\eta)$, then we are done by choosing $r = \bar{r}$, because Lemma B.1 implies that $\eta \not\lhd A_{\varphi(r,\mathsf{R}(\eta))}$. If $\bar{r} \notin \mathsf{R}(\eta)$, then for each $r' \in \mathsf{Res} \setminus \mathsf{R}(\eta)$, we let $r = r'$ and we prove that $\eta \not\lhd A_{\varphi(r',\mathsf{Res})}$, which is the thesis. To do that, consider an offending run $q_0 \xrightarrow{\eta} q_k$ of the automaton $A_{\varphi(\bar{r},\mathsf{Res})}$ on η. We show how to reconstruct an offending run of $A_{\varphi(r',\mathsf{R}(\eta))}$ on η. Let $\ddot{\delta}_{\bar{r},\mathsf{Res}}$ and $\ddot{\delta}_{r',\mathsf{R}(\eta)}$ be the respective transition sets of these automata, as in Def. 2. We will show in a while that, for each $r'' \in \mathsf{R}(\eta)$, $q \xrightarrow{\alpha(r'')} q'$ in $\ddot{\delta}_{\bar{r},\mathsf{Res}}$ if and only if $q \xrightarrow{\alpha(r'')} q'$ in $\ddot{\delta}_{r',\mathsf{R}(\eta)}$. Similarly

to the proof of Lemma B.1, this implies that $A_{\varphi(\bar{r},\mathsf{Res})}$ and $A_{\varphi(r',\mathsf{R}(\eta))}$ have the same transitions on the resources in $\mathsf{R}(\eta)$, which concludes the proof.

Let $r'' \in \mathsf{R}(\eta)$, and consider an edge $q \xrightarrow{\vartheta}\!\!\!\!\circ\; q'$ of $\varphi(x)$. There are the following exhaustive cases:

- if $\vartheta = \alpha(r'')$ with $r'' \in \mathsf{Res}_s$, then $q \xrightarrow{\alpha(r'')} q'$ is in $\ddot{\delta}_{\bar{r},\mathsf{Res}}$ and in $\ddot{\delta}_{r',\mathsf{R}(\eta)}$.
- the case $\vartheta = \alpha(x)$ does not apply, because $\bar{r}, r' \notin \mathsf{R}(\eta)$, while $r'' \in \mathsf{R}(\eta)$.
- if $\vartheta = \alpha(\bar{x})$, then $q \xrightarrow{\alpha(r'')} q'$ is in $\ddot{\delta}_{\bar{r},\mathsf{Res}}$ if $r'' \in (\mathsf{Res} \cup \{?\}) \setminus \{\bar{r}\} \supseteq \mathsf{R}(\eta)$, and it is in $\ddot{\delta}_{r',\mathsf{R}(\eta)}$ if $r'' \in (\mathsf{R}(\eta) \cup \{?\}) \setminus \{r'\} \supseteq \mathsf{R}(\eta)$. Since $r'' \in \mathsf{R}(\eta)$, then $q \xrightarrow{\alpha(r'')} q'$ is in $\ddot{\delta}_{\bar{r},\mathsf{Res}}$ as well as in $\ddot{\delta}_{r',\mathsf{R}(\eta)}$. □

Lemma 7. For all traces η and η', if $\eta\eta'$ is valid, then η is valid.

Proof. Trivial from the definition of $\models$ in Def. 6. □

Lemma 9. A trace η is valid if and only if $\eta \lhd A_{\varphi_{[\,]}(r,\mathsf{R}(\eta))}$, for all φ occurring in η, and for all $r \in \mathsf{R}(\eta) \cup \{r'\}$, where r' is an arbitrary resource in $\mathsf{Res} \setminus \mathsf{R}(\eta)$.

Proof. The statement holds trivially for $\eta = \varepsilon$, because ε is valid ($act(\varepsilon) = \{\!|\,|\!\}$), and $\varepsilon \lhd A_{\varphi_{[\,]}(r,\mathsf{R}(\eta))}$ (indeed, the initial state is in the first layer of the framed policy automaton, where all the states are non-offending). Let then $\eta = \eta'\beta$, where $\eta' = \beta_1 \cdots \beta_n$.

For the "if" part, we proceed by induction on the length of η. Assume that, for all policies φ occurring in η, $\eta \lhd A_{\varphi_{[\,]}(r,\mathsf{R}(\eta))}$. Let $r' \in \mathsf{Res} \setminus \mathsf{R}(\eta)$. Since all the offending states in $A_{\varphi_{[\,]}(r,R(\eta))}$ are sinks by construction, then $\eta' \lhd A_{\varphi_{[\,]}(r,\mathsf{R}(\eta))}$. By the induction hypothesis, this implies that η' is valid. To prove that also η is valid, we have to consider the following cases on the form of the event β:

- if $\beta \in \mathsf{Ev}$, then $act(\eta) = act(\eta')$. Let $\bar{\eta} = \eta^{-[\,]} = (\eta')^{-[\,]}\,\beta$. So, we have to prove that $\models \eta'$, and $\bar{\eta} \models \varphi$ for all $\varphi \in act(\eta')$. The first statement follows from the induction hypothesis. For the second statement, assume by contradiction that $\bar{\eta} \not\models \varphi$. Then, by Lemma 3 there is an offending run $\bar{q}^{(0)} \xrightarrow{\bar{\eta}} \bar{q}^{(\bar{n})}$ of $A_{\varphi(r,\mathsf{R}(\eta))}$, for some $r \in \mathsf{R}(\eta) \cup \{r'\}$. We shall show in a while how to construct an offending run $q^{(0)} \xrightarrow{\eta} q^{(n+1)}$ of $A_{\varphi_{[\,]}(r,\mathsf{R}(\eta))}$.

  First, we introduce some notation. Let Q and $\dot{Q}$ be respectively the states in the first and in the second layer of $A_{\varphi_{[\,]}(r,R(\eta))}$. Let F and $\dot{F}$ be the final states in $A_{\varphi(r,\mathsf{R}(\eta))}$ and in $A_{\varphi_{[\,]}(r,\mathsf{R}(\eta))}$. For each state q in $A_{\varphi_{[\,]}(r,\mathsf{R}(\eta))}$, let $q@Q$ and $q@\dot{Q}$ be the projections of q on the first and on second layer.

  At the first step of our construction for an offending run, let $q^{(0)} = \bar{q}^{(0)}$. For the remaining steps, we shall scan the trace η with the index k, the trace $\bar{\eta}$ with $\bar{k}$, and at each step we shall define $q^{(k)}$. For the k-th step, there are the following exhaustive cases:

- if $\beta_k = [_\varphi$, then $q^{(k)} \in Q$, because η has no redundant framings. Let $q^{(k+1)} = q^{(k)}@\dot{Q}$. By construction (see Def. 8), $A_{\varphi_{[\,]}(r,\mathsf{R}(\eta))}$ has a transition $q^{(k)} \xrightarrow{[_\varphi} q^{(k+1)}$.
- if $\beta_k =]_\varphi$, then $q^{(k)} \in \dot{Q}$, because η has no redundant framings. Moreover, $\bar{q}^{(\bar{k})} \notin F$, otherwise we have found a (strict) prefix of η that is offending for $A_{\varphi_{[\,]}(r,\mathsf{R}(\eta))}$. Let $q^{(k+1)} = q^{(k)}@Q$. Then, $A_{\varphi_{[\,]}(r,\mathsf{R}(\eta))}$ can take the transition $q^{(k)} \xrightarrow{]_\varphi} q^{(k+1)}$.
- if $\beta_k = [_{\varphi'}$ or $\beta_k =]_{\varphi'}$ for some $\varphi' \neq \varphi$, then let $q^{(k+1)} = q^{(k)}$.
- otherwise, if $\beta_k \in \mathsf{Ev}$, there are the following two subcases:
  - if $q^{(k)} \in Q$, then choose $q^{(k+1)} = \bar{q}^{(\bar{k}+1)}$.
  - if $q^{(k)} \in \dot{Q}$, then $\bar{q}^{(\bar{k})} \notin F$: otherwise, we have found a (strict) prefix of η that is offending for $A_{\varphi_{[\,]}(r,\mathsf{R}(\eta))}$. Therefore, $A_{\varphi(r,\mathsf{R}(\eta))}$ has a transition $\bar{q}^{(\bar{k})} \xrightarrow{\beta_{\bar{k}}} \bar{q}^{(\bar{k}+1)}$. Let $q^{(k+1)} = \bar{q}^{(\bar{k}+1)}@\dot{Q}$. Then, $A_{\varphi_{[\,]}(r,\mathsf{R}(\eta))}$ has the transition $q^{(k)} \xrightarrow{\beta_k} q^{(k+1)}$.

We have then constructed a run $q^{(0)} \xrightarrow{\eta} q^{(n+1)}$ of $A_{\varphi_{[\,]}(r,\mathsf{R}(\eta))}$. Since $\varphi \in act(\eta)$, then in η the scope of φ has been entered but not exited. Thus, $q^{(n+1)}$ is in the second layer of $A_{\varphi_{[\,]}(r,\mathsf{R}(\eta)}$. The state $q^{(n+1)}$ is offending, because $\bar{q}_k \in F$. This yields the expected contradiction.

- if $\beta =]_\varphi$, then $act(\eta) \subseteq act(\eta')$, so the induction hypothesis suffices.

- if $\beta = [_\varphi$, then $act(\eta) = act(\eta') \cup \{\!|\varphi|\!\}$. Thus, η is valid if η' is valid (which is true by the induction hypothesis), $(\eta')^{-[\,]} \models \varphi'$ for all $\varphi' \in act(\eta')$, and $(\eta')^{-[\,]} \models \varphi$. Similarly to the case $\beta \in \mathsf{Ev}$, we can prove that the hypothesis $(\eta')^{-[\,]} \not\models \varphi$ leads to a contradiction, i.e. that $\eta \not\models A_{\varphi_{[\,]}(r,\mathsf{R}(\eta))}$ for some $r \in \mathsf{R}(\eta) \cup \{r'\}$ and $r' \in \mathsf{Res} \setminus \mathsf{R}(\eta)$.

For the "only if" part, assume that η is valid. We proceed by induction on the length of η. The base case $\eta = \varepsilon$ is trivial. For the inductive case, assume that $\eta = \eta'\beta$. We consider the following cases on the form of the event β:

- if $\beta \in \mathsf{Ev}$, then $act(\eta) = act(\eta')$. Let $\bar{\eta} = \eta^{-[\,]} = (\eta')^{-[\,]}\beta$. Since η is valid, then Lemma 7 implies that η' is valid. Also, $\bar{\eta} \models \varphi$ for all $\varphi \in act(\eta)$. From the first fact, the induction hypothesis implies that $\eta' \lhd A_{\varphi'_{[\,]}(r,\mathsf{R}(\eta))}$ for all $\varphi' \in \eta'$. By contradiction, assume that $\eta \not\lhd A_{\varphi_{[\,]}(r,\mathsf{R}(\eta)}$ for some φ, i.e. η drives $A_{\varphi_{[\,]}(r,\mathsf{R}(\eta))}$ to an offending state (note that any run on η' is non-offending). Let $q^{(0)} \xrightarrow{\eta'} q^{(n)} \xrightarrow{\beta} q^{(n+1)}$ be such an offending run, with $q^{(n+1)}@Q \in F$. We now show that this contradicts the hypothesis $\bar{\eta} \models \varphi$. To do that, we construct an offending run $\bar{q}^{(0)} \xrightarrow{(\eta')^{-[\,]}} q^{(\bar{n})} \xrightarrow{\beta} q^{(\bar{n}+1)}$ of $A_{\varphi(r,\mathsf{R}(\eta))}$ on $\bar{\eta}$.

At the first step of this construction, let $\bar{q}^{(0)} = q^{(0)}$. For the remaining steps, we shall scan the trace η with the index k, the trace $\bar{\eta}$ with $\bar{k}$, and at each step we shall define $\bar{q}^{(\bar{k})}$. For the k-th step, Since framing events are not present in $\bar{\eta} = \eta^{-[]}$, it suffices to deal with the case $\beta_k \in \mathsf{Ev}$. Consider the transition $q^{(k)} \xrightarrow{\beta_k} q^{(k+1)}$ in $A_{\varphi_{[]}(r,\mathsf{R}(\eta))}$. Let $\bar{q}^{(\bar{k}+1)} = q^{(k+1)}@Q$. Then, $A_{\varphi(r,\mathsf{R}(\eta))}$ has the transition $\bar{q}^{(\bar{k})} \xrightarrow{\beta_{\bar{k}}} \bar{q}^{(\bar{k}+1)}$. Since $\bar{q}^{(\bar{k}+1)} = q^{(n+1)}@Q \in F$, we have then found an offending run of $A_{\varphi(r,\mathsf{R}(\eta))}$ on $\eta^{-[]}$ – contradiction.

- if $\beta = [_\varphi$, then $act(\eta) = act(\eta') \cup \{\!|\varphi|\!\}$. Since η is valid, then Lemma 7 implies that η' is valid. Also, $\eta^{-[]} \models \varphi'$ for all $\varphi' \in act(\eta')$, and $\eta^{-[]} \models \varphi$. From the first fact, the induction hypothesis implies that $\eta' \lhd A_{\varphi'_{[]}(r,\mathsf{R}(\eta))}$ for all $\varphi' \in \eta'$. If $\varphi' \neq \varphi$, then appending the framing event $[_\varphi$ to η' results in a self-loop in $A_{\varphi'_{[]}(r,\mathsf{R}(\eta))}$, and so $\eta \lhd A_{\varphi'_{[]}(r,\mathsf{R}(\eta))}$. If $\varphi' = \varphi$, then (similarly to the previous case $\beta \in \mathsf{Ev}$) we use the fact that $\eta^{-[]} \models \varphi$ in a contrapositive argument, to show that there cannot be offending runs of $A_{\varphi_{[]}(r,\mathsf{R}(\eta))}$ on η.
- if $\beta =]_\varphi$, then $act(\eta) = act(\eta') \setminus \{\!|\varphi|\!\}$. For all φ', the automaton $A_{\varphi'_{[]}(r,\mathsf{R}(\eta))}$ has an edge labelled $]_{\varphi'}$ from the second to the first layer, which has no offending states. Therefore, since $\eta' \lhd A_{\varphi'_{[]}(r,\mathsf{R}(\eta))}$, then the event β results in a transition to the first layer, and so $\eta \lhd A_{\varphi'_{[]}(r,\mathsf{R}(\eta))}$. □

Example 14. *We now show a policy that would not be expressible without abstract edges labelled $\bar{x}$. Consider the policy $\psi(x)$ that prohibits $\alpha(x)\alpha(x)\beta$ and $\alpha(x)\alpha(y)\gamma$ for all x and $y \neq x$, and that allows $\alpha(x)\alpha(x)\gamma$ for all $\gamma \neq \beta$. This is modelled by the usage policy in Fig. 8.* □

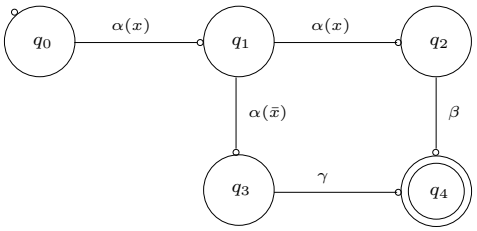


Figure 8: The usage policy $\psi(x)$.

Lemma B.2. *For each well-formed trace η with $\# \notin \mathsf{R}(\eta)$, policy φ, resource r, and name-collapsing σ such that $\sigma(r) \neq \_$:*

(a) if $\sigma^{-1}(\#) \subseteq \{r, \#\}$ (i.e., only r and $\#$ are mapped to $\#$ by σ), then:

$$\eta \lhd A_{\varphi(r,\mathsf{R}(\eta))} \iff \eta\sigma \lhd A_{\varphi(r\sigma,\mathsf{R}(\eta\sigma))}$$

(b) if $\eta\sigma \lhd A_{\#}$ *and* $r \in \mathsf{R}(\eta)$*, then:*

$$\eta \lhd A_{\varphi(r,\mathsf{R}(\eta))} \implies \eta\sigma \lhd A_{\varphi(r\sigma,\mathsf{R}(\eta\sigma))}$$

(c) if $r \notin \mathsf{R}(\eta) \cup \mathsf{R}(\eta\sigma)$*, then:*

$$\eta \lhd A_{\varphi(r,\mathsf{R}(\eta))} \implies \eta\sigma \lhd A_{\varphi(r,\mathsf{R}(\eta\sigma))}$$

Proof. Let $\eta = \beta_1 \cdots \beta_k$. Below, we shall refer to transition sets of the policy automaton $A_{\varphi(r,R)}$ as $\delta_{r,R}, \dot{\delta}_{r,R}$ and $\ddot{\delta}_{r,R}$. (see Def. 2).

For (B.2a), we prove the contrapositive. Assume that $\eta \not\lhd A_{\varphi(r,\mathsf{R}(\eta))}$. Then, there exists an offending run of $A_{\varphi(r,\mathsf{R}(\eta))}$ on η:

$$q_0 \xrightarrow{\beta_0} q_1 \xrightarrow{\beta_1} \cdots \xrightarrow{\beta_{k-1}} q_k \in F$$

We now show how to construct an offending run of $A_{\varphi(r\sigma,\mathsf{R}(\eta)\sigma)}$ on $\eta\sigma$. Consider a transition $q_i \xrightarrow{\beta_i} q_{i+1}$, for $i \geq 0$. There are three cases, according to Def. 2: either this transition was introduced by the relation $\ddot{\delta}_{r,\mathsf{R}(\eta)}$, or it was introduced as a self loop by $\dot{\delta}_{r,\mathsf{R}(\eta)}$, or as an unknown-resource transition by $\delta_{r,\mathsf{R}(\eta)}$.

We consider first the case in which $q_i \xrightarrow{\beta_i} q_{i+1}$ comes from the relation $\ddot{\delta}$. We have three subcases.

- the transition comes from instantiating $q_i \xrightarrow{\alpha(x)}\!\!\circ\; q'$, and $\beta_i = \alpha(r)$. The thesis comes from instantiating x with $r\sigma$.
- the transition comes from instantiating $q_i \xrightarrow{\alpha(\bar{x})}\!\!\circ\; q'$, and $\beta_i = \alpha(r')$ with $r' \in (\mathsf{R}(\eta) \cup \{?\}) \setminus \{r\}$. To instantiate $\bar{x}$ with $r'\sigma$, we must prove that $r'\sigma \in (\mathsf{R}(\eta\sigma) \cup \{?\}) \setminus \{r\sigma\}$. Since $r'\sigma \in \mathsf{R}(\eta\sigma) \cup \{?\}$ is trivially implied by the above, it suffices to check $r'\sigma \neq r\sigma$. There are the following cases.
  - If $r = \#$, since $r' \neq r$ then $r'\sigma \neq \# = r\sigma$ by the hypothesis of (a).
  - If $r \in \mathsf{Res}_s$, we have $r\sigma = r \neq r' = r'\sigma$.
  - If $r \in \mathsf{Res}_d$, since by hypothesis $r\sigma \neq \_$ then $r\sigma = \#$. By the hypothesis of (a), $r \neq r'$ implies $r'\sigma \neq \# = r\sigma$.
- the transition comes from $q_i \xrightarrow{\alpha(r')}\!\!\circ\; q'$ for some $r' \in \mathsf{Res}_s$ and $\beta_i = \alpha(r')$. Since $\beta_i\sigma = \beta_i$, the thesis trivially holds.

If instead the transition has been introduced by $\dot{\delta}$ as a self-loop, then $\nexists q'.\ q_i \xrightarrow{\beta_i} q' \in \ddot{\delta}_{r,\mathsf{R}(\eta)}$. To show that there is a corresponding self-loop $q_i \xrightarrow{\beta_i\sigma} q_{i+1}$ in $\dot{\delta}_{r\sigma,\mathsf{R}(\eta)\sigma}$, we show that $\nexists q'.\ q_i \xrightarrow{\beta_i\sigma} q' \in \ddot{\delta}_{r\sigma,\mathsf{R}(\eta)\sigma}$. We proceed by proving the contrapositive. There are the following three cases.

- the transition comes from instantiating $q_i \xrightarrow{\alpha(x)}\!\!\circ\; q'$, and $\beta_i\sigma = \alpha(r\sigma)$. We have that $\beta_i = \alpha(r')$ for some r', and therefore $r\sigma = r'\sigma$. We have the following three cases.

- If $r = \#$, then by the hypothesis of (a) we obtain $r' = \#$ which implies the thesis.
- Otherwise, if $r \in \mathsf{Res}_s$, then $r\sigma = r = r'\sigma$ and so $r' = r$.
- Otherwise, since $r \in \mathsf{Res}_d$ and $r\sigma \neq \_$ by hypothesis, we must have $r\sigma = \# = r'\sigma$. By the hypothesis of (a) there is only one such r. Thus, $r = r'$, which implies the thesis.

- the transition comes from instantiating $q_i \xrightarrow{\alpha(\bar{x})}\!\!\circ\; q'$, and $\beta_i\sigma = \alpha(r'')$ with $r'' \in (\mathsf{R}(\eta\sigma) \cup \{?\}) \setminus \{r\sigma\}$. We have that $\beta_i = \alpha(r')$ for some r' such that $r'\sigma = r''$, and $r'\sigma \neq r\sigma$, which implies $r' \neq r$. From this and $r' \in \mathsf{R}(\eta)$, we can instantiate $\bar{x}$ with r' to build a corresponding transition in $\ddot{\delta}_{r,\mathsf{R}(\eta)}$.

- the transition comes from $q_i \xrightarrow{\alpha(r')}\!\!\circ\; q'$ with $r' \in \mathsf{Res}_s$, $\beta_i = \alpha(r')$, and $r'\sigma = r'$. The thesis trivially holds.

The above settles the cases in which the transition $q_i \xrightarrow{\beta_i} q_{i+1}$ has been introduced by $\ddot{\delta}$ or by $\dot{\delta}$. If instead the transition has been introduced as an unknown-resource transition by δ, then $\beta_i = \alpha(?)$ was derived from a transition labelled $\alpha(r')$, for some $r' \in \mathsf{R}(\eta)$, in the relation $\dot{\delta}_{r,\mathsf{R}(\eta)}$. By the above, there exists a corresponding transition $q_i \xrightarrow{\alpha(r\sigma)} q_{i+1}$ in $\dot{\delta}_{r\sigma,\mathsf{R}(\eta)\sigma}$. Therefore, we have a transition $q_i \xrightarrow{\alpha(?)} q_{i+1}$ in $\delta_{r\sigma,\mathsf{R}(\eta)\sigma}$.

The proof of (B.2b) is similar, by contrapositive. Assume that $\eta\sigma \not\triangleleft A_{\varphi(r\sigma,\mathsf{R}(\eta)\sigma)}$. Then, there exists an offending run of $A_{\varphi(r\sigma,\mathsf{R}(\eta)\sigma)}$ on $\eta\sigma$:

$$q_0 \xrightarrow{\beta_0\sigma} q_1 \xrightarrow{\beta_1\sigma} \cdots \xrightarrow{\beta_{k-1}\sigma} q_k \in F$$

We now show how to construct an offending run of $A_{\varphi(r,\mathsf{R}(\eta))}$ on η. Consider a transition $q_i \xrightarrow{\beta_i\sigma} q_{i+1}$, for $i \geq 0$.

There are three cases, according to Def. 2: either this transition was introduced by the relation $\ddot{\delta}$, or it was introduced as a self loop by $\dot{\delta}$, or as an unknown-resource transition by δ.

We first consider the case in which $q_i \xrightarrow{\beta_i\sigma} q_{i+1}$ comes from the relation $\ddot{\delta}_{r\sigma,\mathsf{R}(\eta)\sigma}$. We have three subcases.

- the transition comes from instantiating $q_i \xrightarrow{\alpha(x)}\!\!\circ\; q'$, and $\beta_i\sigma = \alpha(r\sigma)$. So, $\beta_i = \alpha(r')$ with $r\sigma = r'\sigma$. If $r' \in \mathsf{Res}_s$, this implies $r = r'$, and the thesis comes from instantiating x with $r\sigma$. Otherwise $r' \in \mathsf{Res}_d$, and so $r \in \mathsf{Res}_d$. Since $r\sigma \neq \_$, we must have $r'\sigma = r\sigma = \#$. We now show $r = r'$, which implies the thesis. If $r \neq r'$, by the well-formedness of η, we have two distinct events $new(r)$ and $new(r')$ in η. This would imply that we have a duplicate $new(\#)$ in $\eta\sigma$, and thus $\eta\sigma \not\triangleleft A_{\#}$, contradicting the hypothesis.

- the transition comes from instantiating $q_i \xrightarrow{\alpha(\bar{x})}\!\!\circ\; q'$, and $\beta_i\sigma = \alpha(r')$ for some $r' \in \mathsf{R}(\eta)$ such that $r'\sigma \in (\mathsf{R}(\eta)\sigma \cup \{?\}) \setminus \{r\sigma\}$. Similarly to the previous case, this implies that $r' \neq r$. So, we can instantiate $\bar{x}$ with r', and obtain the thesis.

- the transition comes from $q_i \xrightarrow{\alpha(r')}\!\!\circ\; q'$ for some $r' \in \mathsf{Res}_s$ and $\beta_i = \alpha(r')$. Since $\beta_i\sigma = \beta_i$, the thesis trivially holds.

If the transition has been added as a self-loop by $\dot{\delta}_{r\sigma,\mathsf{R}(\eta)\sigma}$, then $\nexists q'.\ q_i \xrightarrow{\beta_i\sigma} q' \in \ddot{\delta}_{r\sigma,\mathsf{R}(\eta)\sigma}$. To show that there is a corresponding self-loop in $\dot{\delta}_{r,\mathsf{R}(\eta)}$, we show that $\nexists q'.\ q_i \xrightarrow{\beta_i} q' \in \ddot{\delta}_{r,\mathsf{R}(\eta)}$. We proceed by proving the contrapositive. There are the following three cases.

- the transition comes from $q_i \xrightarrow{\alpha(x)}\!\!\circ\; q'$, and $\beta_i = \alpha(r)$. Since $r \in \mathsf{R}(\eta)$ by the hypotheses of (b), then $r\sigma \in \mathsf{R}(\eta\sigma)$, and so $q_i \xrightarrow{\beta_i\sigma} q' \in \ddot{\delta}_{r\sigma,\mathsf{R}(\eta\sigma)}$.

- the transition comes from $q_i \xrightarrow{\alpha(\bar{x})}\!\!\circ\; q'$, and $\beta_i = \alpha(r')$ with $r' \in (\mathsf{R}(\eta) \cup \{?\}) \setminus \{r\}$. If $r' \in \mathsf{Res}_s$, then $\beta_i = \beta_i\sigma$ which implies the thesis. If $r' \in \mathsf{Res}_d$, we have either $r'\sigma = \_$ or $r'\sigma = \#$. In the first case, since $r\sigma \neq \_$ by hypothesis, we have that $r'\sigma \in (\mathsf{R}(\eta\sigma) \cup \{?\}) \setminus \{r\sigma\}$, so we can instantiate $\bar{x}$ with $r'\sigma$. In the second case, we have $r\sigma \neq \#$: otherwise, the well-formedness of η would imply that both $\mathit{new}(r)$ and $\mathit{new}(r')$ occur in η, so there would be two $\mathit{new}(\#)$ in $\eta\sigma$, so contradicting the hypothesis $\eta\sigma \lhd A_{\#}$. Therefore, we have $r'\sigma \in (\mathsf{R}(\eta\sigma) \cup \{?\}) \setminus \{r\sigma\}$, which implies the thesis.

- the transition comes from $q_i \xrightarrow{\alpha(r')}\!\!\circ\; q'$ and $\beta_i = \alpha(r')$ with $r'\sigma = r' \in \mathsf{Res}_s$. The thesis trivially holds.

If instead the transition has been introduced as an unknown-resource transition by $\delta_{r\sigma,\mathsf{R}(\eta\sigma)}$, then $\beta_i\sigma = \alpha(?)\sigma = \alpha(?)$ has been derived from a transition $q_i \xrightarrow{\alpha(r')} q_{i+1}$ in $\dot{\delta}_{r\sigma,\mathsf{R}(\eta\sigma)}$, for some $r' \in \mathsf{R}(\eta\sigma)$. By the above, there exists a corresponding transition $q_i \xrightarrow{\alpha(r'')} q_{i+1}$ in $\dot{\delta}_{r,\mathsf{R}(\eta)}$, for some r'' such that $r'\sigma = r''$. Therefore, we have a transition $q_i \xrightarrow{\alpha(?)} q_{i+1}$ in $\delta_{r,\mathsf{R}(\eta)}$.

For (B.2c), again we prove the contrapositive. Assume that $\eta\sigma \not\lhd A_{\varphi(r,\mathsf{R}(\eta)\sigma)}$. Then, there exists an offending run of $A_{\varphi(r,\mathsf{R}(\eta)\sigma)}$ on $\eta\sigma$:

$$q_0 \xrightarrow{\beta_0\sigma} q_1 \xrightarrow{\beta_1\sigma} \cdots \xrightarrow{\beta_{k-1}\sigma} q_k \in F$$

We now show how to construct an offending run of $A_{\varphi(r,\mathsf{R}(\eta))}$ on η. Consider a transition $q_i \xrightarrow{\beta_i\sigma} q_{i+1}$, for $i \geq 0$. Similarly to the above items, we deal with the following cases.

If $q_i \xrightarrow{\beta_i\sigma} q_{i+1}$ has been introduced by $\ddot{\delta}_{r,\mathsf{R}(\eta)\sigma}$, there are three subcases.

- the transition comes from $q_i \xrightarrow{\alpha(x)}\!\!\circ\; q'$, and $\beta_i\sigma = \alpha(r)$. This contradicts the assumption $r \notin \mathsf{R}(\eta\sigma)$.

- the transition comes from $q_i \xrightarrow{\alpha(\bar{x})}\!\!\circ\; q'$, and $\beta_i\sigma = \alpha(r')$ for some r' such that $r' \in (\mathsf{R}(\eta\sigma) \cup \{?\}) \setminus \{r\}$. Then, $\beta_i = \alpha(r'')$, for some $r'' \in \mathsf{R}(\eta)$ such that $r''\sigma = r'$. Since $r \notin \mathsf{R}(\eta)$ by the hypotheses of (c), then $r'' \neq r$. Therefore $r'' \in (\mathsf{R}(\eta) \cup \{?\}) \setminus \{r\}$.

- the transition comes from $q_i \xrightarrow{\alpha(r')}\!\!\circ\; q'$ for some $r' \in \mathsf{Res}_s$ and $\beta_i\sigma = \alpha(r')$. Since $\beta_i\sigma = \beta_i$, the thesis trivially holds.

If the transition is a self-loop, then $\nexists q'.\ q_i \xrightarrow{\beta_i\sigma} q' \in \ddot{\delta}_{r,\mathsf{R}(\eta)\sigma}$. To show that there is a corresponding self-loop $q_i \xrightarrow{\beta_i} q_{i+1}$ in $\dot{\delta}_{r,\mathsf{R}(\eta)}$, we show by contrapositive that $\nexists q'.\ q_i \xrightarrow{\beta_i} q' \in \ddot{\delta}_{r,\mathsf{R}(\eta)}$. We have three cases.

- the transition comes from $q_i \xrightarrow{\alpha(x)}\!\!\circ\; q'$, with $\beta_i = \alpha(r)$. This contradicts the assumption $r \notin \mathsf{R}(\eta)$.

- the transition comes from $q_i \xrightarrow{\alpha(\bar{x})}\!\!\circ\; q'$, and $\beta_i = \alpha(r')$ for some $r' \in (\mathsf{R}(\eta) \cup \{?\}) \setminus \{r\}$. This implies that $r'\sigma \in \mathsf{R}(\eta\sigma) \cup \{?\}$. Since $r \notin \mathsf{R}(\eta\sigma)$ by the hypotheses of (c), then $r'\sigma \neq r$. Thus, $r'\sigma \in (\mathsf{R}(\eta\sigma) \cup \{?\}) \setminus \{r\}$.

- the transition comes from $q_i \xrightarrow{\alpha(r')}\!\!\circ\; q'$ with $\beta_i = \alpha(r')$ and $r'\sigma = r' \in \mathsf{Res}_s$. The thesis trivially holds.

□

Definition 17. *Let $\mathcal{R}$ be a binary relation over BPAs. We say that $\mathcal{R}$ is a* simulation *if and only if, whenever $P\,\mathcal{R}\,Q$:*

- *if $Q \xrightarrow{a} Q'$, then there exists P' such that $P \xrightarrow{a} P'$ and $P'\,\mathcal{R}\,Q'$*
- *if $Q \not\rightarrow$, then $P \not\rightarrow$*

Let $\mathcal{R}$ be a binary relation over BPAs. We say that $\mathcal{R}$ is a bisimulation *if and only if, whenever $P\,\mathcal{R}\,Q$:*

- *if $P \xrightarrow{a} P'$, then there exists Q' such that $Q \xrightarrow{a} Q'$ and $P'\,\mathcal{R}\,Q'$, and*
- *if $Q \xrightarrow{a} Q'$, then there exists P' such that $P \xrightarrow{a} P'$ and $P'\,\mathcal{R}\,Q'$.*

We say that P simulates *Q ($P \succ Q$ in symbols) when there exists a simulation $\mathcal{R}$ such that $P\,\mathcal{R}\,Q$. We say that P* is bisimilar to *Q ($P \sim Q$ in symbols) when there exists a bisimulation $\mathcal{R}$ such that $P\,\mathcal{R}\,Q$.*

Lemma B.3. *The similarity relation $\succ$ between BPAs is a simulation, and a preorder. The bisimilarity relation $\sim$ is a bisimulation, and an equivalence relation. Moreover, for all BPAs P, Q, R and substitution σ:*

$$P \sim Q \implies P\sigma \sim Q\sigma \tag{3a}$$

$$P \sim Q \implies P \cdot R \sim Q \cdot R \tag{3b}$$

$$P \succ Q \implies P \cdot R \succ Q \cdot R \wedge R \cdot P \succ R \cdot Q \tag{3c}$$

$$P \sim Q \iff P \succ Q \wedge Q \succ P \tag{3d}$$

Proof. These are well-known properties. Item (B.3a) is trivially true. To prove (B.3b), it suffices noting that if $P \sim 0$, then $P = 0$. □

Definition 18. *Let ζ be a substitution from BPA variables to BPA variables, and let $P = \langle p, \Delta \rangle$ be a BPA. We say that ζ is a* coherent renaming of P *when $\Delta\zeta$ is coherent.*

Lemma B.4. *Let ζ be a coherent renaming of P. Then, $P \sim P\zeta$.*

Proof. It is straightforward to check that the relation $\{ \langle P, P\zeta \rangle \mid P \text{ is a BPA} \}$ is a bisimulation. □

Note. In what follows, we compare the traces of U to the traces of $\mathsf{B}(U)$. While the similarity preorder $\succ$ is used in the auxiliary results, the main lemmata (7d and B.8b) only care about traces, as well as Theorem 13. Because of this, without loss of generality we can use here a slightly different definition of $\mathsf{B}(U)$, which does not affect the generated traces. More in detail, the equation $\mathsf{B}_{\#}(\nu n.U)_\Theta = \mathsf{B}_{\#}(U\{\_/n\})_\Theta$ of Def. 10 is changed to $\mathsf{B}_{\#}(\nu n.U)_\Theta = 0 \cdot \mathsf{B}_{\#}(U\{\_/n\})_\Theta$. This minor modification shall allow us to state Lemma B.6.

Lemma B.5. *For all d, U, U', h' and Θ, there exists a coherent renaming ζ of $\mathsf{B}_d(U\{U'/h'\})_\Theta$ such that ζ is the identity on the BPA variables in Θ, and:*

$$\mathsf{B}_d(U\{U'/h'\})_\Theta\, \zeta \;=\; \mathsf{B}_d(U)_{\Theta\{\mathsf{B}(U')_\Theta/h'\}}$$

where $\mathsf{B}_d(U)_{\Theta\{\langle p,\Delta\rangle/h\}}$ is a shorthand for $\langle p', \Delta + \Delta' \rangle$ if $\mathsf{B}_d(U)_{\Theta\{p/h\}} = \langle p', \Delta' \rangle$.

Proof. By induction on the structure of U. The base cases $U = \varepsilon, \alpha(r), h$ and the inductive cases $U = U_0 \cdot U_1$, $U = U_0 + U_1$, $U = \nu n.\, U''$ and $U = \varphi[U'']$ are straightforward. Consider $U = \mu h.\, U''$. If $h = h'$, trivial. Otherwise:

$$\mathsf{B}_d(U\{U'/h'\})_\Theta = \langle X, \{X \triangleq p\} + \Delta \rangle \quad \text{where } \langle p, \Delta \rangle = \mathsf{B}_d(U''\{U'/h'\})_{\Theta\{X/h\}}$$

$$\mathsf{B}_d(U)_{\Theta\{\mathsf{B}_d(U')_\Theta/h'\}} = \langle X, \{X \triangleq p'\} + \Delta' \rangle \quad \text{where } \langle p', \Delta' \rangle = \mathsf{B}_d(U'')_{\Theta\{X/h,\, \mathsf{B}_d(U')_\Theta/h'\}}$$

(note that we can choose the fresh variable X in both the equations). By the induction hypothesis, there exists a renaming ζ coherent with $\mathsf{B}_d(U''\{U'/h'\})_{\Theta\{X/h\}}$, such that ζ leaves the variables in $\Theta\{X/h\}$ untouched (thus $X\zeta = X$), and:

$$\mathsf{B}_d(U''\{U'/h'\})_{\Theta\{X/h\}}\, \zeta \;=\; \mathsf{B}_d(U'')_{\Theta\{X/h, \mathsf{B}_d(U')_\Theta/h'\}}$$

i.e. $p\zeta = p'$ and $\Delta\zeta = \Delta'$. Therefore, it follows that:

$$\begin{aligned}\mathsf{B}_d(U\{U'/h'\})_\Theta\,\zeta &= \langle X, \{X \triangleq p\} + \Delta\rangle\,\zeta \\ &= \langle X\zeta, \{X\zeta \triangleq p\zeta\} + \Delta\zeta\rangle \\ &= \langle X, \{X \triangleq p'\} + \Delta'\rangle \\ &= \mathsf{B}_d(U)_{\Theta\{\mathsf{B}_d(U')_\Theta/h'\}}\end{aligned}$$

which concludes the proof. □

Lemma B.6. *For all closed usages U, $\mathsf{B}\_(U) \succ \mathsf{B}_\#(U)$.*

Proof. By induction on the size of U. Most cases follow directly by the induction hypothesis, since $\mathsf{B}\_(U)$ and $\mathsf{B}_\#(U)$ are defined through similar inductive equations. The only significant case is $U = \nu n.U'$, for which we have:

$$\mathsf{B}\_(\nu n.\,U') = \mathsf{B}_\#(U'\{\#/n\}) + \mathsf{B}\_(U'\{\_/n\}) \succ \varepsilon \cdot \mathsf{B}\_(U'\{\_/n\}) = \mathsf{B}_\#(\nu n.\,U')$$

□

Lemma B.7. *Let U be a closed usage, let $\mathcal{R}$ be a finite set of resources, and let σ be a uniquely-collapsing substitution, with witness $r \neq \#$ satisfying $\sigma(r) = \#$. Then, for all $a, U', \mathcal{R}'$ such that $U, \mathcal{R} \xrightarrow{a} U', \mathcal{R}'$:*

$$\text{(7a)} \quad \mathcal{R}' \setminus \mathcal{R} = \{r\} \implies \forall P \sim \mathsf{B}\_(U)\sigma.\, \exists P' \succ \mathsf{B}_\#(U')\sigma : P \xrightarrow{a\sigma} P'$$

$$\text{(7b)} \quad r \notin \mathcal{R}' \implies \forall P \sim \mathsf{B}\_(U)\sigma.\, \exists P' \sim \mathsf{B}\_(U')\sigma : P \xrightarrow{a\sigma} P'$$

$$\text{(7c)} \quad r \in \mathcal{R} \implies \forall P \sim \mathsf{B}_\#(U)\sigma.\, \exists P' \sim \mathsf{B}_\#(U')\sigma : P \xrightarrow{a\sigma} P'$$

Moreover, if U is initial, then, for all η and U':

$$\text{(7d)} \quad U, \emptyset \xrightarrow{\eta} U', \mathcal{R}' \implies \exists P' : \mathsf{B}\_(U) \xrightarrow{\eta\sigma} P'$$

Proof. For (B.7a-c), we proceed by induction on the structure of U.

- $U = \varepsilon$. Trivial case, since ε is a final configuration.
- $U = \alpha(r')$, with $r' \in \mathsf{Res} \cup \{?\}$. Then $a = \alpha(r')$, $U' = \varepsilon$, and $\mathcal{R}' = \mathcal{R}$.
  For (7a), the fact $\mathcal{R}' = \mathcal{R}$ contradicts the hypothesis.
  For (7b) and (7c), by hypothesis we have $P \sim \mathsf{B}_d(U)\sigma = \alpha(r)\sigma$, for all d. By Def. 17, $P \xrightarrow{\alpha(r)\sigma} P'$, with $P' \sim 0$. Then, $P' \sim \mathsf{B}_d(U')\sigma = 0$.
- $U = \nu n.U''$. Then $a = \varepsilon$, and $U' = U''\{r'/n\}$, $\mathcal{R}' = \mathcal{R} \cup \{r'\}$ with $r' \in \mathsf{Res}_d \setminus \mathcal{R}$. We have three cases.
  - For (7a), we have $r' = r$. By hypothesis, $P \sim \mathsf{B}\_(U)\sigma = \mathsf{B}\_(U''\{\_/n\})\sigma + \mathsf{B}_\#(U''\{\#/n\})\sigma \xrightarrow{\varepsilon} \mathsf{B}_\#(U''\{\#/n\})\sigma$. By Def. 17, we have $P \xrightarrow{\varepsilon} P' \sim \mathsf{B}_\#(U''\{\#/n\})\sigma = \mathsf{B}_\#(U''\{r/n\})\sigma = \mathsf{B}(U')\sigma$.

- For (7b), we have $r' \neq r$. Since σ is uniquely-collapsing, $\sigma(r') = \_$. By hypothesis, $P \sim \mathsf{B}_\_(U)\sigma = \mathsf{B}_\_(U''\{\_/n\})\sigma + \mathsf{B}_\#(U''\{\#/n\})\sigma \xrightarrow{\varepsilon} \mathsf{B}_\_(U''\{\_/n\})\sigma$. By Def. 17, we have $P \xrightarrow{\varepsilon} P' \sim \mathsf{B}_\_(U''\{\_/n\})\sigma = \mathsf{B}_\_(U''\{r'/n\})\sigma = \mathsf{B}_\_(U')\sigma$.
- For (7c), we have $r' \neq r$. Since σ is uniquely-collapsing, $\sigma(r') = \_$. By hypothesis, $P \sim \mathsf{B}_\#(U)\sigma = \varepsilon \cdot \mathsf{B}_\#(U''\{\_/n\})\sigma \xrightarrow{\varepsilon} \mathsf{B}_\#(U''\{\#/n\})\sigma$. By Def. 17, we have $P \xrightarrow{\varepsilon} P' \sim \mathsf{B}_\#(U''\{\_/n\})\sigma = \mathsf{B}_\#(U''\{r'/n\})\sigma = \mathsf{B}_\#(U')\sigma$.

- $U = U_0 \cdot U_1$. There are two subcases, i.e. $U_0 = \varepsilon$ or $U_0 \neq \varepsilon$.

  If $U_0 = \varepsilon$, then $a = \varepsilon$, $U' = U_1$, and $\mathcal{R}' = \mathcal{R}$.

  For (7a), this contradicts the hypothesis $\mathcal{R}' \neq \mathcal{R}$.

  For (7b) and (7c), let $P \sim \mathsf{B}_d(U)\sigma$, for any d. By bisimilarity, since $\mathsf{B}_d(U)\sigma = 0 \cdot \mathsf{B}_d(U')\sigma \xrightarrow{\varepsilon} \mathsf{B}_d(U')\sigma$ then there exists $P' \sim \mathsf{B}_d(U')\sigma$ such that $P \xrightarrow{\varepsilon} P'$.

  If $U_0 \neq \varepsilon$, assume that $U_0, \mathcal{R} \xrightarrow{a'} U_0', \mathcal{R}'$. Then, $a = a'$, and $U' = U_0' \cdot U_1$. We have two further subcases:

  - For (7a) let $P \sim \mathsf{B}_\_(U)\sigma = \mathsf{B}_\_(U_0)\sigma \cdot \mathsf{B}_\_(U_1)\sigma$. Let $P_0 = \mathsf{B}_\_(U_0)\sigma$. By the induction hypothesis, there exists $P_0' \succ \mathsf{B}_\#(U_0')\sigma$ such that $P_0 \xrightarrow{a\sigma} P_0'$. Let $P' = P_0' \cdot \mathsf{B}_\_(U_1)\sigma$. Then, $P \sim P_0 \cdot \mathsf{B}_\_(U_1)\sigma \xrightarrow{a\sigma} P_0' \cdot \mathsf{B}_\_(U_1)\sigma = P'$. By Lemma B.3c and Lemma B.6, it follows that $P' = P_0' \cdot \mathsf{B}_\_(U_1)\sigma \succ \mathsf{B}_\#(U_0')\sigma \cdot \mathsf{B}_\_(U_1)\sigma \succ \mathsf{B}_\#(U_0')\sigma \cdot \mathsf{B}_\#(U_1)\sigma = \mathsf{B}_\#(U')\sigma$.
  - For (7b) and (7c), for any d, let $P \sim \mathsf{B}_d(U)\sigma = \mathsf{B}_d(U_0)\sigma \cdot \mathsf{B}_d(U_1)\sigma$. Let $P_0 = \mathsf{B}_d(U_0)\sigma$. By the induction hypothesis, there exists $P_0' \sim \mathsf{B}_d(U_0')\sigma$ such that $P_0 \xrightarrow{a\sigma} P_0'$. Let $P' = P_0' \cdot \mathsf{B}_d(U_1)\sigma$. Then, $P \sim P_0 \cdot \mathsf{B}_d(U_1)\sigma \xrightarrow{a\sigma} P_0' \cdot \mathsf{B}_d(U_1)\sigma = P'$. By Lemma B.3b, it follows that $P_0' \cdot \mathsf{B}_d(U_1)\sigma \sim \mathsf{B}_d(U_0')\sigma \cdot \mathsf{B}_d(U_1)\sigma = \mathsf{B}_d(U')\sigma$.

- $U = U_0 + U_1$. Then, $a = \varepsilon$ and $\mathcal{R}' = \mathcal{R}$. For (7a), this contradicts $\mathcal{R}' \neq \mathcal{R}$.

  For (7b) and (7c), there are two further subcases for U', i.e. either $U' = U_0$ or $U' = U_1$. By hypothesis, for any d, $P \sim \mathsf{B}_d(U)\sigma = \mathsf{B}_d(U_0)\sigma + \mathsf{B}_d(U_1)\sigma$. If $U' = U_0$, then $P \xrightarrow{\varepsilon} P' \sim \mathsf{B}_d(U_0)\sigma = \mathsf{B}_d(U')\sigma$. The second case is similar.

- $U = \mu h.\, U''$. We have $a = \varepsilon$, $U' = U''\{U/h\}$, and $\mathcal{R}' = \mathcal{R}$.

  For (7a), this contradicts $\mathcal{R}' \neq \mathcal{R}$.

  For (7b) and (7c), for any d, we proceed as follows.

  Let $P \sim \mathsf{B}_d(\mu h.\, U'')\sigma = \langle X, \{X \triangleq p\} + \Delta\rangle\sigma$, where $\langle p, \Delta\rangle = \mathsf{B}_d(U'')_{\{X/h\}}$. By Definition 17, since $\langle X, \{X \triangleq p\} + \Delta\rangle\sigma \xrightarrow{\varepsilon} \langle p, \{X \triangleq p\} + \Delta\rangle\sigma$, then, for some BPA P', $P \xrightarrow{\varepsilon} P' \sim \langle p, \{X \triangleq p\} + \Delta\rangle\sigma$. Note that:

$$\langle p, \{X \triangleq p\} + \Delta\rangle\sigma \;=\; \mathsf{B}_d(U'')_{\{X/h\}}\sigma \cup \{X \triangleq p\}\sigma \;=\; \mathsf{B}_d(U'')_{\{\mathsf{B}_d(U)/h\}}\sigma$$

By Lemma B.5, it follows that:

$$\mathsf{B}_d(U''\{U/h\})\zeta \;=\; \mathsf{B}_d(U'')_{\{\mathsf{B}_d(U)/h\}}$$

for some renaming ζ coherent with $\mathsf{B}_d(U''\{U/h\})$. Then, by Lemma B.4:

$$\mathsf{B}_d(U''\{U/h\})\zeta \;\sim\; \mathsf{B}_d(U''\{U/h\}) \;=\; \mathsf{B}_d(U')$$

Summing up, Lemmata B.3a and B.3d allow us to conclude that:

$$P' \;\sim\; \langle p, \{X \triangleq p\} + \Delta\rangle\sigma \;\sim\; \mathsf{B}_d(U')\sigma$$

For (7d) it suffices to use the previous statements in an inductive argument over the number of transitions. For the first step we pick $P = \mathsf{B}_\_(U)$: indeed, $\mathsf{B}_\_(U)\sigma = \mathsf{B}_\_(U)$ because U is initial. We start with $\mathcal{R} = \emptyset$, to be later pupulated with freshly created resources. We can split the transition steps in two phases. In the first phase, the witness r has not yet been generated, so we keep applying (7b). Let η' the trace produced in this phase. If r is never generated in η, then we are done, because $\eta = \eta'$ and (7b) gives us a P_0 such that $\mathsf{B}_\_(U) \xrightarrow{\eta\sigma} P_0$. Otherwise, r is eventually generated, say upon the transition $U_0, \mathcal{R}_0 \xrightarrow{\varepsilon} U_1, \mathcal{R}_1$. We then apply (7a) once, obtaining for some $P_1 \succ \mathsf{B}_\#(U_1)$:

$$\mathsf{B}_\_(U) \xrightarrow{\eta'\sigma} P_0 \xrightarrow{\varepsilon} P_1$$

After that, we consider $\mathsf{B}_\#(U_1)$, and apply (7c) to it, following every move of U_1 in this second phase: let η'' the trace generated in this way. Clearly, we have $\eta = \eta'\eta''$. We get $\mathsf{B}_\#(U_1) \xrightarrow{\eta''\sigma} P_2$ for some P_2. Since $P_1 \succ \mathsf{B}_\#(U_1)$, then for some P_2' we have $P_1 \xrightarrow{\eta''\sigma} P_2'$. Summing everything up:

$$\mathsf{B}_\_(U) \xrightarrow{\eta'\sigma} P_0 \xrightarrow{\varepsilon} P_1 \xrightarrow{\eta''\sigma} P_2'$$

□

Lemma B.8. *Let U be a closed usage, and let σ be a name-collapsing substitution, mapping infinitely many resources to $\#$ and infinitely many to $\_$. Then, for each $d, a, \mathcal{R}$, each P and P':*

$$\begin{aligned} &P \sim \mathsf{B}_\_(U)\sigma \;\wedge\; P \xrightarrow{a} P' \\ &\implies \exists U', \mathcal{R}', b.\ (U, \mathcal{R} \xrightarrow{b} U', \mathcal{R}' \;\wedge\; P' \prec \mathsf{B}_\_(U')\sigma \;\wedge\; b\sigma = a) \end{aligned} \tag{B.8a}$$

Moreover, if U is initial, then for each $\bar\eta, P$ and $\mathcal{R}$:

$$\mathsf{B}_\_(U) \xrightarrow{\bar\eta} P \implies \exists U', \eta, \mathcal{R}'.\ (U, \mathcal{R} \xrightarrow{\eta} U', \mathcal{R}' \;\wedge\; \eta\sigma = \bar\eta) \tag{B.8b}$$

Proof. For (B.8a), we proceed by induction on the structure of U.

- If $U = \varepsilon$, then $\mathsf{B}_\_(U)\sigma = 0$ has no transitions.

- If $U = \alpha(r)$, then $a = \alpha(r)\sigma$ and $\mathsf{B}_{\_}(U)\sigma \xrightarrow{\alpha(r)\sigma} 0 \sim P'$. The thesis holds for $U' = \varepsilon$.

- If $U = \nu n.U''$, then $P \sim \mathsf{B}_{\_}(U)\sigma = \mathsf{B}_{\#}(U\{\#/n\})\sigma + \mathsf{B}_{\_}(U\{\_/n\})\sigma$. In both cases, we have $a = \varepsilon$. So, either $\mathsf{B}_{\_}(U)\sigma \xrightarrow{a} \mathsf{B}_{\#}(U''\{\#/n\})\sigma$ or $\mathsf{B}_{\_}(U)\sigma \xrightarrow{a} \mathsf{B}_{\_}(U''\{\_/n\})\sigma$.

  If $\mathsf{B}_{\_}(U)\sigma \xrightarrow{\varepsilon} \mathsf{B}_{\#}(U''\{\#/n\})\sigma$, we have $P' \sim \mathsf{B}_{\#}(U''\{\#/n\})\sigma$. By Lemma B.6, this implies $P' \prec \mathsf{B}_{\_}(U''\{\#/n\})\sigma$. Choose a fresh $r \in \mathsf{Res}_d \setminus \mathcal{R}$ such that $r\sigma = \#$: this is possible because there is an infinite number of such resources. Then, we let $U' = U''\{r/n\}$. Finally, we have $U, \mathcal{R} \xrightarrow{\varepsilon} U', \mathcal{R} \cup \{r\}$ and $P' \prec \mathsf{B}_{\_}(U''\{\#/n\})\sigma = \mathsf{B}_{\_}(U')\sigma$.

  Otherwise, if $\mathsf{B}_{\_}(U)\sigma \xrightarrow{\varepsilon} \mathsf{B}_{\_}(U''\{\_/n\})\sigma$, we proceed in a similar way, with $P' \sim \mathsf{B}_{\_}(U''\{\_/n\})\sigma$, and choosing a fresh r such that $r\sigma = \_$.

- If $U = U_0 \cdot U_1$, we have two further subcases:

  - If $U_0 = \varepsilon$, then $P \sim \mathsf{B}_{\_}(U_0)\sigma \cdot \mathsf{B}_{\_}(U_1)\sigma = 0 \cdot \mathsf{B}_{\_}(U_1)\sigma$. So, $a = \varepsilon$ and $P' \sim \mathsf{B}_{\_}(U_1)\sigma$. The thesis follows choosing $U' = U_1$.
  - If $U_0 \neq \varepsilon$, then $P \sim \mathsf{B}_{\_}(U_0)\sigma \cdot \mathsf{B}_{\_}(U_1)\sigma$, with $\mathsf{B}_{\_}(U_0)\sigma \neq 0$. So, we must have that $\mathsf{B}_{\_}(U_0)\sigma \cdot \mathsf{B}_{\_}(U_1)\sigma \xrightarrow{a} P'' \cdot \mathsf{B}_{\_}(U_1)\sigma$ with $\mathsf{B}_{\_}(U_0)\sigma \xrightarrow{a} P''$ and $P' \sim P'' \cdot \mathsf{B}_{\_}(U_1)\sigma$. The induction hypothesis (with $U = U_0$, and $P = \mathsf{B}_{\_}(U_0)\sigma$) implies that there exists some U_0' and $\mathcal{R}_0'$ such that $U_0, \mathcal{R} \xrightarrow{b} U_0', \mathcal{R}_0'$ and $P'' \prec \mathsf{B}_{\_}(U_0')\sigma$ with $b\sigma = a$. By Lemma B.3c, $P'' \cdot \mathsf{B}_{\_}(U_1)\sigma \prec \mathsf{B}_{\_}(U_0')\sigma \cdot \mathsf{B}_{\_}(U_1)\sigma$. Let $U' = U_0' \cdot U_1$: then $U = U_0 \cdot U_1, \mathcal{R} \xrightarrow{b} U', \mathcal{R}_0'$. Moreover $P' \sim P'' \cdot \mathsf{B}_{\_}(U_1)\sigma \prec \mathsf{B}_{\_}(U_0')\sigma \cdot \mathsf{B}_{\_}(U_1)\sigma = \mathsf{B}_{\_}(U')\sigma$.

- If $U = U_0 + U_1$, then $P \sim \mathsf{B}_{\_}(U_0)\sigma + \mathsf{B}_{\_}(U_1)\sigma$. So we have $a = \varepsilon$, and either $P' \sim \mathsf{B}_{\_}(U_0)\sigma$ or $P' \sim \mathsf{B}_{\_}(U_1)\sigma$. In the first case, we choose $U' = U_0$, while in the second $U' = U_1$. Then the thesis trivially holds.

- If $U = \mu h.\, U''$, then $P \sim \mathsf{B}_{\_}(U)\sigma = \langle X, \{X \triangleq p\} + \Delta\rangle\sigma$, where $\langle p, \Delta\rangle = \mathsf{B}_{\_}(U'')_{\{X/h\}}$. From $\langle X, \{X \triangleq p\} + \Delta\rangle\sigma$ there is only one transition, going to $\langle p, \{X \triangleq p\} + \Delta\rangle\sigma$. We must then have $a = \varepsilon$, and $P' \sim \langle p, \{X \triangleq p\} + \Delta\rangle$. Let $U' = U''\{U/h\}$. Clearly, $U \xrightarrow{\varepsilon} U'$. By Lemma B.5 there is some coherent renaming ζ such that $\mathsf{B}_{\_}(U''\{U/h\})\zeta = \mathsf{B}_{\_}(U'')_{\{\mathsf{B}_{\_}(U)/h\}}$. So, by Lemma B.4, $\mathsf{B}_{\_}(U''\{U/h\}) \sim \mathsf{B}_{\_}(U''\{U/h\})\zeta$. Then:

$$\begin{aligned}
\mathsf{B}_{_}(U')\sigma &= \mathsf{B}_{_}(U''\{U/h\})\sigma \\
&\sim \mathsf{B}_{_}(U'')_{\{\mathsf{B}_{_}(U)/h\}}\sigma \\
&= \mathsf{B}_{_}(U'')_{\{\langle X, \{X \triangleq p\} + \Delta\rangle/h\}}\sigma \\
&= \langle p, \{X \triangleq p\} + \Delta + \{X \triangleq p\} + \Delta\rangle\sigma \\
&= \langle p, \{X \triangleq p\} + \Delta\rangle\sigma \sim P'
\end{aligned}$$

For B.8(b), we proceed by induction on the number of transitions. At the first step $\mathsf{B}\_(U)\sigma = \mathsf{B}\_(U)$, since U is initial. By applying B.8(a), we obtain $U, \mathcal{R} \xrightarrow{b} U', \mathcal{R}'$ with $a = b\sigma$ and $P' \prec \mathsf{B}\_(U')$. Further moves of P' are simulated by $\mathsf{B}\_(U')$, so we can apply B.8(a) again. That is, if $P' \xrightarrow{a'} P''$, we also have $\mathsf{B}\_(U') \xrightarrow{a'} \bar{P}''$ with $P'' \prec \bar{P}''$, so we have that $U', \mathcal{R}' \xrightarrow{b'} U'', \mathcal{R}''$ for some U'' such that $P'' \prec \bar{P}'' \prec \mathsf{B}\_(U'')$ and some b' such that $b'\sigma = a'$. By further repeating these steps, we can construct a trace η for U such that $\eta\sigma = \bar{\eta}$. □

Theorem 13. Let U be an initial usage without redundant framings. Then, U is valid if and only if:

$$\forall A_{\varphi_{[]}(r,\mathcal{R})} \in \Phi(U) \;\; : \;\; [\![\mathsf{B}(U)]\!] \lhd A_{\varphi_{[]}(r,\mathcal{R})} \;\mathsf{W}\; A_{\#}$$

Proof. For the "if" case (soundness), we prove the contrapositive. Assume U is *not* valid. By Lemma 9, let $\eta \in [\![U]\!]$ be such that $\eta \not\lhd A_{\varphi_{[]}(r,\mathcal{R})}$ for some $A_{\varphi_{[]}(r,\mathcal{R})}$ in $\Phi(U)$. Let σ be a name-collapsing substitution such that $\sigma(\#) = \#$, $\sigma(r) = \#$ and $\sigma(r') = \_$ for all the other dynamic resources r'. Note that this σ satisfies the hypothesis of Lemma B.2(a). Since U is initial, then $\#$ does not occur in U, so $\# \notin \mathsf{R}(\eta)$. By Lemma B.7(b), $\mathsf{B}(U) \xrightarrow{\eta\sigma} P'$, for some P'. By Lemma B.2(a), $\eta \not\lhd A_{\varphi_{[]}(r,\mathsf{R}(\eta))}$ implies that $\eta\sigma \not\lhd A_{\varphi_{[]}(\#,\mathsf{R}(\eta\sigma))}$. Since $\mathsf{R}(\eta\sigma) \subseteq \mathsf{R}(U)$, by Lemma B.1 it follows that $\eta\sigma \not\lhd A_{\varphi_{[]}(\#,\mathsf{R}(U))}$. By Definition 10, $A_{\varphi_{[]}(\#,\mathsf{R}(U))} \in \Phi(U)$. The "unique witness" policy $A_{\#}$ is satisfied by η, because U can generate only well-formed traces. Therefore, $\eta\sigma \not\lhd A_{\varphi_{[]}(\#,\mathsf{R}(U))}\, \mathsf{W}\, A_{\#}$.

For the "only if" part (completeness) we prove the contrapositive. Assume some $\bar{\eta} \in [\![\mathsf{B}(U)]\!]$ such that $\bar{\eta} \lhd A_{\#}$ and $\bar{\eta} \not\lhd A_{\varphi_{[]}(r,\mathsf{R}(U))}$ for some $r \in U$ or $r = \#$. Let σ be a name-collapsing substitution such that $\sigma(r) \neq \_$ and such that there are infinitely many resources mapped to $\#$ and infinitely many mapped to $\_$. By Lemma B.8(b), $U \xrightarrow{\eta} U'$, for some U' and η such that $\eta\sigma = \bar{\eta}$. Since U is initial, then $r\sigma = r$ and $\mathsf{R}(U)\sigma = \mathsf{R}(U)$. We can then rephrase $\bar{\eta} \not\lhd A_{\varphi_{[]}(r,\mathsf{R}(U))}$ as $\eta\sigma \not\lhd A_{\varphi_{[]}(r\sigma,\mathsf{R}(U)\sigma)}$. Since $\mathsf{R}(\eta\sigma) \subseteq \{r \in \mathsf{R}(\eta) \cap \mathsf{Res}_s\} \cup \{\#, \_\} \subseteq \mathsf{R}(U) = \mathsf{R}(U)\sigma$, by Lemma B.1 it follows that $\eta\sigma \not\lhd A_{\varphi_{[]}(r\sigma,\mathsf{R}(\eta\sigma))}$. We have the following three cases:

- If $r = \# \notin \eta\sigma$, by Lemma B.2(c) then $\eta \not\lhd A_{\varphi_{[]}(\#,\mathsf{R}(\eta))}$.
- If $r = \# \in \eta\sigma$, then $\alpha(\#)$ occurs in $\eta\sigma$, for some action α. This implies that, for some dynamic r', $\alpha(r')$ occurs in η and $r'\sigma = \#$. By Lemma B.2(b), $\eta \not\lhd A_{\varphi_{[]}(r',\mathsf{R}(\eta))}$.
- Otherwise, $\# \neq r$, then $r \in \mathsf{Res}_s$ and we have two further subcases.

  If $r \in \mathsf{R}(\eta)$, by Lemma B.2(b) $\eta \not\lhd A_{\varphi_{[]}(r,\mathsf{R}(\eta))}$.

  Otherwise $r \notin \mathsf{R}(\eta)$, which implies $r \notin \mathsf{R}(\eta\sigma)$ and then by Lemma B.2(c), $\eta \not\lhd A_{\varphi_{[]}(r,\mathsf{R}(\eta))}$.

In all the cases above, we have found an r' such that $\eta \not\lhd A_{\varphi_{[]}(r',\mathsf{R}(\eta))}$. Therefore, from Lemma 9 it follows that U is not valid. □