Plans for service composition

Massimo Bartoletti Pierpaolo Degano Gian Luigi Ferrari
Dipartimento di Informatica, Università di Pisa, Italy
{bartolet, degano, giangi}@di.unipi.it

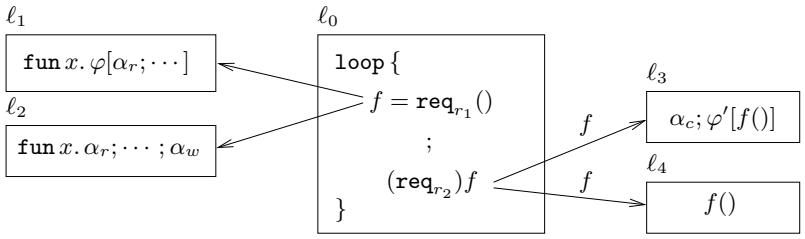
Abstract

We study how to compose services in the presence of security constraints. By analysing the abstract behaviour of a set of services, we are able to determine the plans that drive safe program executions. These plans can be of different kinds, and we study here three of them, with different expressive power. Simple plans choose a single service for each request; multi-choice plans instead can choose among a set of services; dependent plans exploit the knowledge of past choices to decide for the future ones.

1 Introduction

The ability of selecting and assembling together heterogeneous services is an important step towards the full development of service-oriented computing [7, 6, 4]. A *service* is a stand-alone component distributed over a network, and made available through standard interaction mechanisms. Composition of services may require peculiar mechanisms to handle complex interaction patterns (e.g. to implement transactions), while enforcing non-functional requirements on the system behaviour (e.g. security and service level agreement). Service composition heavily depends on which information about a service is made public, on how to choose those services that match the user's requirements, and on their actual run-time behaviour. Security makes service composition even harder. Services may be offered by different providers, which only partially trust each other. On the one hand, providers have to guarantee the delivered service to respect a given security policy, in any interaction with the operational environment, and regardless of who actually called the service. On the other hand, clients may want to protect themselves from the services invoked.

The *orchestration* of services concerns their composition and the coordination of their behaviour. The proposed standards for service orchestration, e.g. BPEL4WS [1, 5], offer an invocation mechanism based on service names and signatures. In the world of services this can be a limitation, because services can be created, removed and updated on-the-fly. In [3] we proposed an invocation-by-property mechanism: a service request $\mathtt{req}_r\varphi$ can be resolved by any service that obeys the property φ on demand. To protect themselves from callers, also



services can impose security policies on their own behaviour. Technically, in [3] we extracted from a program a static over-approximation of its behaviour, where each request is associated with all the services obeying the given property. This abstract behaviour was then model-checked for validity, so as to ensure that all the executions of the original program are safe, and guarantee that all the requests are served. However, this static technique may discard potentially safe programs, because the selection of a service, even though respecting the imposed property, may affect security later on in the program execution.

To overcome this problem, it is highly convenient to separately consider each different combination of service selections, namely a *plan*. If the overall abstract behaviour produced under a plan is model-checked valid, then *any* execution driven by that plan is safe.

In this paper we study three different kinds of plans: *simple plans*, that associate a single service with each request, *multi-choice plans*, that map requests into sets of services, and *dependent plans*, that also convey the dependence of a service selection with the choices made in the past. Given a program approximation, we extract from it the behaviour produced under a given plan, for all the three kinds considered. The techniques of [3] then suffice to verify validity, so providing us with the plans that drive safe program executions. For brevity, all the proofs are omitted here; the interested reader can find them in [2].

2 A motivating example

Suppose that a distributed application is based on web services, and that there are a number of locations offering services that may be combined to complete useful tasks. Services are functional units explicitly located at network sites, and have a published public interface. Unlike standard syntactic signatures, this interface includes an abstraction of the service behaviour. To obtain a service with a specific behaviour, a client queries the network for a published interface matching the requirements — a sort of *call-by-property* invocation.

To illustrate our approach, consider the scenario in the above figure, where we only display the control flow of services, the requests, and the aspects relevant to security. The main program at site ℓ_0 consists of a loop issuing two requests at each iteration. The request labelled r_1 asks for a piece of mobile code (e.g. an applet), and it can be served by two code providers at ℓ_1 and ℓ_2. The service

at ℓ_1 returns a function that protects itself with a policy φ, permitting its use in certified sites only (modelled by the event α_c). Within the function body, the only security-relevant operation is a read α_r on the file system where the delivered code is run. The code provided by ℓ_2 first reads some local data (α_r), and eventually writes them (α_w) through a network connection. Since ℓ_0 has a limited computational power, the code f obtained by the request r_1 is passed as a parameter to the service invoked by the request r_2. This request can be served by ℓ_3 and ℓ_4. The service at ℓ_3 is certified, and runs the provided code f under a "Chinese Wall" security policy φ', requiring that no data can be written (α_w) after reading them (α_r). The service at ℓ_4 is not certified, and it simply runs f.

The abstract behaviour of the whole system is rendered by the following *history expression* (it approximates the run-time system behaviour, and can be inferred by type and effect systems, e.g. as in [3]):

$$H = \mu h.\, \{r_2[\ell_3] \rhd \alpha_c \cdot \varphi'[\{r_1[\ell_1] \rhd \varphi[\alpha_r], r_1[\ell_2] \rhd \alpha_r \cdot \alpha_w\}] \\ r_2[\ell_4] \rhd \{r_1[\ell_1] \rhd \varphi[\alpha_r], r_1[\ell_2] \rhd \alpha_r \cdot \alpha_w\}\} \cdot h$$

The above means that, if r_2 is served by ℓ_3 (written as $r_2[\ell_3]$), then the event α_c occurs, followed by a *safety framing* φ'. This framing protects the behaviour it wraps, i.e. $\varphi[\alpha_r]$ if ℓ_1 is chosen for r_1, or $\alpha_r\alpha_w$ if ℓ_2 is chosen instead. Otherwise, if r_2 is served by ℓ_4, then the behaviour depends on the choice for r_1: if ℓ_1 is taken, then $\varphi[\alpha_r]$, otherwise $\alpha_r\alpha_w$. The outermost operator μ is the standard one used to model the loop in ℓ_0.

From the given history expression H, we wish to extract the *viable* plans that drive safe executions of service composition. To do that, it is convenient to "flatten" history expressions, by collecting all the possible combinations of service choices. In our example, we would obtain:

$$H' = \{r_1[\ell_1] \mid r_2[\ell_3] \rhd \mu h.\, \alpha_c \cdot \varphi'[\varphi[\alpha_r]] \cdot h, \\ r_1[\ell_2] \mid r_2[\ell_4] \rhd \mu h.\, \alpha_r \cdot \alpha_w \cdot h, \\ r_1[\ell_1] \mid r_2[\ell_4] \rhd \mu h.\, \varphi[\alpha_r] \cdot h \\ r_1[\ell_2] \mid r_2[\ell_3] \rhd \mu h.\, \alpha_c \cdot \varphi'[\alpha_r \cdot \alpha_w] \cdot h\}$$

Every element of H' clearly separates the plan from the associated abstract behaviour, which has no further plans within. For instance, under the plan that composes $r_1[\ell_1]$ with $r_2[\ell_3]$ (written as $r_1[\ell_1] \mid r_2[\ell_3]$), the overall abstract behaviour is $\mu h.\, \alpha_c \cdot \varphi'[\varphi[\alpha_r]] \cdot h$. The first two plans in H' are viable, while the others give rise to non-valid behaviour. The plan $r_1[\ell_2] \mid r_2[\ell_4]$ is not viable because the policy φ would be violated when the code f is run on a non certified site; instead, the plan $r_1[\ell_2] \mid r_2[\ell_3]$ would violate φ'. The plans considered so far are called *simple*, because they associate a single service with each request (so $r[\ell] \mid r[\ell']$ is well-formed only if $\ell = \ell'$).

Assume now that a new service for r_2 is discovered at site ℓ_5, offering to run the code f without any constraints. Note that, if we stick to simple plans, then we must choose once and for all one among the viable plans, i.e. $r_1[\ell_1] \mid r_2[\ell_3]$, $r_1[\ell_2] \mid r_2[\ell_4]$, and $r_1[\ell_2] \mid r_2[\ell_5]$. Consequently, at each iteration of the loop the

same service is taken for the request r_2. To be more flexible (i.e. in case the service chosen for r_2 becomes unavailable), we would like to accept as valid also the plan $r_1[\ell_2] \mid r_2[\ell_4, \ell_5]$, where r_2 can be served by either ℓ_4 or ℓ_5. We call this kind of plans *multi-choice*, because a request can be resolved by a set of services. In our running example, this has the advantage of permitting to select for r_2 between ℓ_4 and ℓ_5 at each iteration of the loop.

Planning service composition can be even more complex. Assume for example that an argument g is passed to the request r_1, to invoke a billing service through a request r_3, and so let the code provider invoice the customer ℓ_0 for the service. The same function g is also passed later on to the service which will actually run the code f, to charge ℓ_0 for the cost of the execution. Clearly, the service which provides the code and the one which runs it can choose different billing services. However, neither simple nor multi-choice plans can render adequately this situation, as we will see in a while.

We therefore extend plans to keep track of dependencies among choices. The *dependent* plan $r_1[\ell_1.r_3[\ell_6]] \mid r_2[\ell_3.r_3[\ell_7]]$ is viable: the request r_3 is resolved with ℓ_6 within the service ℓ_1 chosen for r_1, while it is resolved with ℓ_7 within the service ℓ_3 chosen for r_2. The advantage of dependent plans is even more evident if we constrain the code providers to use ℓ_6, only, while code executors can only be paid through ℓ_7. The simple plan (yet not well-formed) that seems to solve the problem is $r_1[\ell_1] \mid r_2[\ell_3] \mid r_3[\ell_6] \mid r_3[\ell_7]$. However, this plan cannot express the linkage between the choice for r_3 within code providers, and that within code executers.

3 Planning service composition

We now set up a formal framework for determining the plans (if any) that safely drive the execution of a program.

3.1 Plans

Plans are intended to drive the selection of services upon request. Here we study three kinds of plans, the syntax of which follows:

Syntax of Plans

Simple plans :	π, π'	::=	0	$\mid$	$\pi \mid \pi'$	$\mid$	$r[\ell]$
Multi-choice plans :	π, π'	::=	0	$\mid$	$\pi \mid \pi'$	$\mid$	$r[\{\ell_1, \ldots, \ell_k\}]$
Dependent plans :	π, π'	::=	0	$\mid$	$\pi \mid \pi'$	$\mid$	$r[\ell.\pi]$

The empty plan 0 has no choices. The composite plan $\pi \mid \pi'$ offers the choices of π and π'. The operator $\mid$ is associative, commutative and idempotent, and its identity is 0 for simple and multi-choice plans only. For a request $\mathtt{req}_r\varphi$, the plan $r[\ell]$ chooses the service offered by site ℓ; analogously, $r[\{\ell_1, \ldots, \ell_k\}]$ can

choose any ℓ_i. Finally, the plan $r[\ell.\pi]$ resolves r with ℓ, and then π drives the execution of the service provided by ℓ.

We define a well-formedness condition on plans, by regarding them as injective mappings. For simple plans, $r[\ell] \mid r[\ell']$ is well-formed when $\ell = \ell'$. Similarly, given two sets S, S' of services, $r[S] \mid r[S']$ is well-formed for multi-choice plans if $S = S'$. For dependent plans, $r[\ell.\pi] \mid r[\ell'.\pi']$ is well-formed when $\ell = \ell'$ and $\pi = \pi'$. Without ambiguity, we abbreviate $r[\ell.0]$ with $r[\ell]$.

3.2 History expressions

We shall extend the history expressions of [3] that statically predict the histories generated by programs at run-time.

History expressions are much alike regular expressions, and include the empty history ε, access events α, sequencing $H \cdot H'$, non-deterministic choice $H + H'$, safety framings $\varphi[H]$, recursion $\mu h.H$ (μ binds the occurrences of the variable h in H), and planned selections $\{\pi_1 \triangleright H_1 \cdots \pi_k \triangleright H_k\}$.

History Expressions

H, H'	$::=$	
	ε	empty
	h	variable
	α	access event
	$H \cdot H'$	sequence
	$H + H'$	choice
	$\varphi[H]$	safety framing
	$\mu h.H$	recursion
	$\{\pi_1 \triangleright H_1 \cdots \pi_k \triangleright H_k\}$	planned selection

Intuitively, access events represent the program actions where sensible resources are accessed; the constructors $\cdot$ and $+$ correspond to sequentialization of code and conditionals, respectively; safety framings model blocks of code subject to security policies; recursion is for loops and recursive functions. More on these constructs is in [3]. The new construct of planned selection abstracts the behaviour of service requests. For instance, $\{r[\ell_1] \triangleright H_1 \cdots r[\ell_k] \triangleright H_k\}$ says that a request r can be resolved into one of the services provided by the sites $\ell_1, \ldots, \ell_k$, which may generate a history represented by $H_1, \ldots, H_k$, respectively.

To define the semantics of history expressions, we enrich the set of events with *framing* events $[_\varphi$, $]_\varphi$, that denote the opening and closing of a safety framing $\varphi[\cdots]$. Finite sequences of access events and *balanced* framing events, possibly ending with the marker !, are called *histories*. The history $\eta!$ approximates a non-terminating computation that generates the sequence of events η. We assume that histories are undistinguishable after truncation, i.e. $\eta!$ followed by η' equals to $\eta!$. The history $\eta = \alpha[_\varphi \alpha']_\varphi$ represents a computation that (i) generates an event α, (ii) enters the scope of φ, (iii) generates α' within the scope of φ, and (iv) leaves the scope of φ.

The *denotational semantics* of history expressions is defined over the lifted cpo of sets of histories [8], ordered by (lifted) set inclusion $\subseteq_\perp$ (where $\perp \subseteq_\perp \mathcal{H}$ for all $\mathcal{H}$, and $\mathcal{H} \subseteq_\perp \mathcal{H}'$ whenever $\mathcal{H} \subseteq \mathcal{H}'$). The least upper bound between two elements of the cpo is standard set union $\cup$, assuming that $\perp \cup \mathcal{H} = \mathcal{H}$. The *strict* least upper bound is denoted by $\cup_\perp$, and it is such that $\perp \cup_\perp \mathcal{H} = \perp$. We stipulate that concatenation of sets of histories is strict, i.e. it returns $\perp$ whenever one of its arguments is such.

The semantics $[\![H]\!]^\pi_\rho$ of a history expression H (in an environment ρ – omitted for closed expressions – mapping variables to sets of histories) is defined by the following rules, and it depends on a given evaluation plan π. The meaning of a planned selection $\{\}$ with no choices is the distinguished element $\perp$: no service is available, so an error occurs. The semantics of a set of choices is the union of the semantics of its components. All the other rules are standard; note that strict union is used in the rule for $+$, because both the branches of a conditional should be successful. Note that the rule for $[\![\{\pi' \rhd H\}]\!]^\pi_\rho$ is missing in the following table. Indeed, the actual semantics depends on the kind of plans we will consider. The next three subsections fill-in the gap.

Semantics of history expressions

$$[\![\varepsilon]\!]^\pi_\rho = \varepsilon \qquad [\![\alpha]\!]^\pi_\rho = \alpha \qquad [\![h]\!]^\pi_\rho = \rho(h) \qquad [\![\varphi[H]]\!]^\pi_\rho = \varphi[[\![H]\!]^\pi_\rho]$$

$$[\![H \cdot H']\!]^\pi_\rho = [\![H]\!]^\pi_\rho \, [\![H']\!]^\pi_\rho \qquad [\![H + H']\!]^\pi_\rho = [\![H]\!]^\pi_\rho \cup_\perp [\![H']\!]^\pi_\rho$$

$$[\![\mu h.H]\!]^\pi_\rho = \bigcup_{n\in\omega} f^n(!) \quad \text{where } f(X) = [\![H]\!]^\pi_{\rho\{X/h\}}$$

$$[\![\{\}]\!]^\pi_\rho = \perp \qquad [\![\{\pi_1 \rhd H_1 \cdots \pi_k \rhd H_k\}]\!]^\pi_\rho = \bigcup_{i\in 1..k} [\![\{\pi_i \rhd H_i\}]\!]^\pi_\rho$$

Example 1. Consider the following history expressions:

$$H_0 = \mu h.\, \alpha \cdot h \qquad H_1 = \mu h.\, h \cdot \alpha \qquad H_2 = \mu h.\, \alpha + h \cdot h + \varphi[h]$$

Then, $[\![H_0]\!]_\emptyset = \alpha^*! = \{!, \alpha!, \alpha\alpha!, \ldots\}$, i.e. H_0 generates histories with an arbitrary number of α, and never terminates. Instead, $[\![H_1]\!]_\emptyset = \{!\}$, i.e. H_1 loops forever, without generating any events. The semantics of H_2 comprises all the histories having an arbitrary number of occurrences of α, and arbitrarily nested, balanced framings of φ. For instance, $\varphi[\alpha]\varphi[\alpha\varphi[\alpha]] \in [\![H_2]\!]^\pi$, for all plans π. □

3.3 Simple plans

We first consider simple plans. The semantics (under an evaluation plan π) of a singleton planned selection $\{\pi' \rhd H\}$ requires that π resolves all the choices $r[\ell]$ that occur in π'. We write $\pi' \sqsubseteq_s \pi$ when π resolves π'; formally, the relation $\sqsubseteq_s$ is a partial order, defined in the table below.

Semantics of simple plans

$$[\![\{\pi' \rhd H\}]\!]^\pi_\rho = \begin{cases} [\![H]\!]^\pi_\rho & \text{if } \pi' \sqsubseteq_s \pi \\ \bot & \text{otherwise} \end{cases}$$

$$0 \sqsubseteq_s \pi \qquad r[\ell] \sqsubseteq_s \pi \ \text{ if } \pi = r[\ell] \mid \pi_1' \qquad \pi_0 \mid \pi_1 \sqsubseteq_s \pi \ \text{ if } \pi_0 \sqsubseteq_s \pi \text{ and } \pi_1 \sqsubseteq_s \pi$$

Example 2. Let $H = \{r[\ell] \rhd \{r'[\ell_1] \rhd \alpha_1, r'[\ell_2] \rhd \alpha_2\}, r[\ell'] \rhd \beta\}$, and let $\pi = r[\ell] \mid r'[\ell_2]$. Then, since $r[\ell] \sqsubseteq_s \pi$ and $r'[\ell_2] \sqsubseteq_s \pi$:

$$\begin{aligned} [\![H]\!]^\pi &= [\![\{r[\ell] \rhd \{r'[\ell_1] \rhd \alpha_1, r'[\ell_2] \rhd \alpha_2\}\}]\!]^\pi \cup [\![\{r[\ell'] \rhd \beta\}]\!]^\pi \\ &= [\![\{r'[\ell_1] \rhd \alpha_1, r'[\ell_2] \rhd \alpha_2\}]\!]^\pi \cup \bot = [\![\{r'[\ell_1] \rhd \alpha_1, r'[\ell_2] \rhd \alpha_2\}]\!]^\pi \\ &= [\![\{r'[\ell_1] \rhd \alpha_1\}]\!]^\pi \cup [\![\{r'[\ell_2] \rhd \alpha_2\}]\!]^\pi = \bot \cup [\![\alpha_2]\!]^\pi = \{\alpha_2\} \end{aligned}$$ □

Example 3. Let $H = \alpha + \{r[\ell] \rhd \beta\}$, and let $\pi = 0$. Then:

$$[\![H]\!]^\pi \ = \ [\![\alpha]\!]^\pi \cup_\bot [\![\{r[\ell] \rhd \beta\}]\!]^\pi \ = \ \{\alpha\} \cup_\bot \bot \ = \ \bot$$

This example models a situation where, e.g. H has been extracted from an expression $\texttt{if}\, b\, \texttt{then}\, \alpha\, \texttt{else}\, (\texttt{req}_r \tau)*$, and e_ℓ is the only service whose type is compatible with τ. The semantics of H is undefined (i.e. $\bot$) because, in case b is false, the plan π is not able to choose any of the proposed services. □

3.4 Multi-choice plans

The definition of the semantics is essentially the same as that for simple plans; it suffices to suitably extend the "resolves" relation.

Semantics of multi-choice plans

$$[\![\{\pi' \rhd H\}]\!]^\pi_\rho = \begin{cases} [\![H]\!]^\pi_\rho & \text{if } \pi' \sqsubseteq_m \pi \\ \bot & \text{otherwise} \end{cases}$$

$$0 \sqsubseteq_m \pi \qquad \pi_0 \mid \pi_1 \sqsubseteq_m \pi \quad \text{if} \quad \begin{matrix} \pi_0 \sqsubseteq_m \pi \\ \pi_1 \sqsubseteq_m \pi \end{matrix} \qquad r[S] \sqsubseteq_m \pi \quad \text{if} \quad \begin{matrix} \pi = r[S'] \mid \pi' \\ S \subseteq S' \end{matrix}$$

3.5 Dependent plans

The treatment of dependent plans requires a little more machinery, because also the evaluation plan must be "consumed". For example, the evaluation of $\{r[\ell] \rhd \{r'[\ell_1] \rhd \alpha_1, r'[\ell_2] \rhd \alpha_2\}\}$ under the plan $\pi = r[\ell.r'[\ell_1]] \mid r'[\ell_2]$ involves evaluating $\{r'[\ell_1] \rhd \alpha_1, r'[\ell_2] \rhd \alpha_2\}$ under the residual plan $r'[\ell_1]$, obtained by consuming the choice $r[\ell]$ offered by π.

Consuming the evaluation plan π while selecting $\pi' \rhd H$ is rendered by π'/π. This operation computes the residual plan under which H must be evaluated – of course π must resolve π', otherwise the result is $\bot$. The definition of / relies on the fact that dependent plans ordered by $\sqsubseteq_d$ form a *meet semi-lattice*, and so the meet $\sqcap$ of any pair of elements always exists.

Semantics of dependent plans

$$[\![\{\pi' \rhd H\}]\!]^{\pi}_{\rho} = \begin{cases} [\![H]\!]^{\pi'/\pi}_{\rho} & \text{if } \pi' \sqsubseteq_d \pi \\ \bot & \text{otherwise} \end{cases}$$

$$0 \sqsubseteq_d \pi \qquad \pi_0 \mid \pi_1 \sqsubseteq_d \pi \quad \text{if} \quad \begin{array}{l} \pi_0 \sqsubseteq_d \pi \\ \pi_1 \sqsubseteq_d \pi \end{array} \qquad r[\ell.\pi_0] \sqsubseteq_d \pi \quad \text{if} \quad \begin{array}{l} \pi = r[\ell.\pi_1] \mid \pi_1' \\ \pi_0 \sqsubseteq_d \pi_1 \end{array}$$

$$0/\pi = \pi \qquad r[\ell.\pi_0]/(r[\ell.\pi_1] \mid \pi_1') = \pi_0/\pi_1 \qquad (\pi_0 \mid \pi_1)/\pi = (\pi_0/\pi) \sqcap (\pi_1/\pi)$$

Example 4. Note that 0 is no more the identity for | in dependent plans. For instance, let $\pi = r[\ell.r'[\ell']]$. Then:

$$[\![\{r[\ell] \rhd \{r'[\ell'] \rhd \alpha\}\}]\!]^{\pi} = [\![\{r'[\ell'] \rhd \alpha\}]\!]^{r'[\ell']} \;=\; [\![\alpha]\!]^{0} \;=\; \alpha$$
$$[\![\{0 \mid r[\ell] \rhd \{r'[\ell'] \rhd \alpha\}\}]\!]^{\pi} = [\![\{r'[\ell'] \rhd \alpha\}]\!]^{0} \;=\; \bot$$

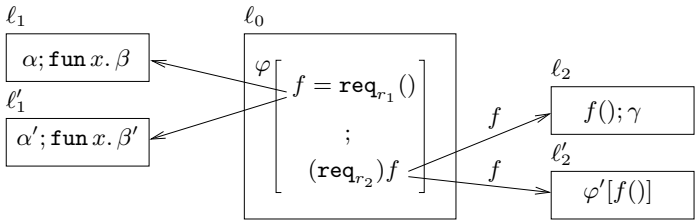
Note that in the second computation we have used the fact that $(0 \mid r[\ell])/\pi = (0/\pi) \sqcap (r[\ell]/\pi) = \pi \sqcap r'[\ell'] = 0$, and that $r'[\ell'] \not\sqsubseteq 0$. □

3.6 Validity

Valid histories represent viable computations, while invalid ones happen to violate some security constraints. For example, consider the history $\eta_0 = \alpha_w\alpha_r\varphi[\alpha_w]$, where φ requires that no write α_w occurs after a read α_r. Then, η_0 is *not* valid according to our intended meaning, because the rightmost α_w occurs within a safety framing enforcing φ, and $\alpha_w\alpha_r\alpha_w$ does not obey φ. We assume hereafter that security policies φ are regular properties of sequences of access events $\alpha_1 \cdots \alpha_k$, and we write $\alpha_1 \cdots \alpha_k \models \varphi$ when the given sequence obeys φ. To be valid, a history η must obey all the policies within their scopes, determined by the framing events in η.

To give a formal definition of validity, it is convenient to introduce the notion of *safe sets*. For example, the history η_0 above has one safe set $\varphi[\{\alpha_w\alpha_r, \alpha_w\alpha_r\alpha_w\}]$. Intuitively, this means that the scope of the framing $\varphi[\cdots]$ spans over the histories $\alpha_w\alpha_r$ and $\alpha_w\alpha_r\alpha_w$. For each safe set $\varphi[\mathcal{H}]$, validity requires that *all* the histories in $\mathcal{H}$ obey φ.

Some notation is now needed. Let $\eta^\flat$ be the history obtained from η by erasing all the framing events, and let η^∂ be the set of all the prefixes of η, including the empty history ε. For example, if $\eta_0 = \alpha_w\alpha_r\varphi[\alpha_w]$, then $(\eta_0^\flat)^\partial = ((\alpha_w\alpha_r[_\varphi\alpha_w]_\varphi)^\flat)^\partial = (\alpha_w\alpha_r\alpha_w)^\partial = \{\varepsilon, \alpha_w, \alpha_w\alpha_r, \alpha_w\alpha_r\alpha_w\}$. Then, the safe sets $S(\eta)$ and validity of histories and history expressions are defined as follows:



Safe sets and validity

$$S(\varepsilon) = \emptyset \qquad S(\eta\,\alpha) = S(\eta) \qquad S(\eta_0\,\varphi[\eta_1]) = S(\eta_0\,\eta_1) \cup \varphi[\eta_0^\flat\,(\eta_1^\flat)^\partial]$$

A history η is *valid* ($\models \eta$ in symbols) when:

$$\varphi[\mathcal{H}] \in S(\eta) \implies \forall \eta' \in \mathcal{H} \,:\, \eta' \models \varphi$$

A history expression H is π-*valid* when $[\![H]\!]^\pi \neq \bot$ and $\eta \in [\![H]\!]^\pi \implies \models \eta$

Note that validity of a history expression is parametric in the given evaluation plan π, and it is defined componentwise on its semantics, provided it is not $\bot$.

Example 5. Let $H = \varphi[\alpha_0 \cdot \{r[\ell_1] \rhd \alpha_1, r[\ell_2] \rhd \varphi'[\alpha_2]\}] \cdot \alpha_3$. Then:

$$S([\![H]\!]^{r[\ell_1]}) = S([_\varphi \alpha_0 \alpha_1]_\varphi \alpha_3) = \{\ \varphi[\{\varepsilon, \alpha_0, \alpha_0\alpha_1\}]\ \}$$
$$S([\![H]\!]^{r[\ell_2]}) = S([_\varphi \alpha_0 [_{\varphi'} \alpha_2]_{\varphi'}]_\varphi \alpha_3) = \{\ \varphi[\{\varepsilon, \alpha_0, \alpha_0\alpha_2\}], \varphi'[\{\alpha_0, \alpha_0\alpha_2\}]\ \}$$

If φ requires "never α_3" and φ' requires "never α_2", then H is $r[\ell_1]$-valid, because the histories ε, α_0, and $\alpha_0\alpha_1$ obey φ. Instead, H is not $r[\ell_2]$-valid, because the history $\alpha_0\alpha_2$ in the safe set $\varphi'[\{\alpha_0, \alpha_0\alpha_2\}]$ does not obey φ'. □

3.7 Planning

Given a correct approximation H to the behaviour of a program, we analyse H to find the plans that drive a correct and viable service composition. This issue is not trivial, because the effect of selecting a given service for a request is not confined to the execution of that service. For instance, the history generated while running a service may later on violate a policy that will become active after the service has returned (see Example 6 below). Since each service selection affects the *whole* execution of a program, we cannot devise a viable plan by selecting services that satisfy the constraints imposed by the requests, only.

Example 6. Consider the figure above, where ℓ_0 makes two requests within a safety framing φ, requiring "never γ after α". Intuitively, any service selected upon the request r_1 returns a function f, then passed as an argument to the request r_2. For instance, the service at ℓ_1 generates the event α, and then returns

a function that will produce β when applied; the service at ℓ'_2 applies f within the framing φ', requiring that "never β'". The associated history expression is:

$$H = \varphi[\{r_1[\ell_1] \rhd \alpha, r_1[\ell'_1] \rhd \alpha'\} \cdot \\ \{r_2[\ell_2] \rhd \{r_1[\ell_1] \rhd \beta, r_1[\ell'_1] \rhd \beta'\} \cdot \gamma,\ r_2[\ell'_2] \rhd \varphi'[\{r_1[\ell_1] \rhd \beta, r_1[\ell'_1] \rhd \beta'\}]\}]$$

Consider first the plan $\pi_1 = r_1[\ell_1] \mid r_2[\ell_2]$. Then, $[\![H]\!]^{\pi_1} = \varphi[\alpha\beta\gamma]$ is not valid, because the policy φ is violated. Consider now $\pi_2 = r_1[\ell'_1] \mid r_2[\ell'_2]$. Then, $[\![H]\!]^{\pi_2} = \varphi[\alpha'\varphi'[\beta']]$ is not valid, because the policy φ' is violated. Instead, the remaining two plans, $r_1[\ell_1] \mid r_2[\ell'_2]$ and $r_1[\ell'_1] \mid r_2[\ell_2]$ make H valid. □

As shown above, the tree-shaped structure of planned selections makes it difficult to determine the plans under which a history expression is valid. In the next section, we present a semantic-preserving transformation that puts history expressions in a special form, amenable to detect viable plans. This special form is called *linear*, and requires that H is a planned selection $\{\pi_1 \rhd H_1 \cdots \pi_k \rhd H_k\}$, where no H_i has further selections. For instance, in the case of simple plans, the linearization of the history expression H in Example 6 is:

$$\{r_1[\ell_1] \mid r_2[\ell_2] \rhd \varphi[\alpha \cdot \beta \cdot \gamma],\ r_1[\ell_1] \mid r_2[\ell'_2] \rhd \varphi[\alpha \cdot \varphi'[\beta]], \\ r_1[\ell'_1] \mid r_2[\ell_2] \rhd \varphi[\alpha' \cdot \beta' \cdot \gamma],\ r_1[\ell'_1] \mid r_2[\ell'_2] \rhd \varphi[\alpha' \cdot \varphi'[\beta']]\}$$

Our main result is that we can detect the viable plans for service composition: executions driven by any of them will never violate the security constraints.

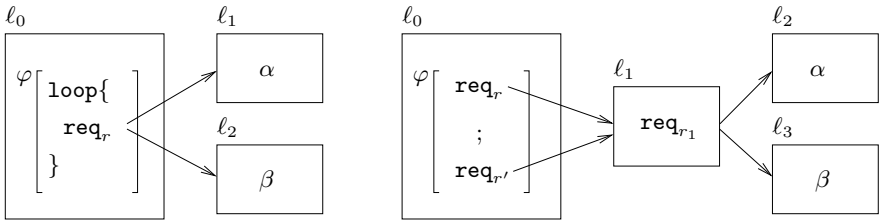
Theorem 1. If $H = \{\pi_1 \rhd H_1 \cdots \pi_k \rhd H_k\}$ is linear, and H_i is valid for some $i \in 1..k$, then H is π_i-valid.

Summing up, given a program P and a history expression H correctly approximating its behaviour, we linearize H into $\{\pi_1 \rhd H_1 \cdots \pi_k \rhd H_k\}$. If some H_i is valid, then we can deduce that H is π_i-valid. Consequently, the plan π_i safely drives the execution of $P\ e$, without resorting to any run-time monitor. To verify the validity of history expressions that, like the H_i above, have no planned selections, the model-checking techniques of [3] suffice.

4 Linearizing Plans

To put a history expression in linear form, for each kind of plan, we define a set of equations on history expressions. When oriented, these equations turn into a finitely terminating and confluent rewriting system – up to the equivalence rules of the three kinds of plans. Formally, we say that:

- $H \equiv H'$, when $[\![H]\!]^\pi_\rho = [\![H']\!]^\pi_\rho$, for each environment ρ and plan π.
- H is *linear* when $H = \{\pi_1 \rhd H_1 \cdots \pi_k \rhd H_k\}$, the plans are pairwise independent (i.e. $\pi_i \not\sqsubseteq \pi_j$ for all $i \neq j$) and no H_i has planned selections.



The relation $\equiv$ turns out to be a congruence for the three kinds of plans considered. Note that it is easy to fulfil the independency condition on plans. Given $\{\pi_1 \rhd H_1 \cdots \pi_k \rhd H_k\}$ and two non-independent plans $\pi_i \sqsubseteq \pi_j$, we substitute $\pi_i \rhd H_i + H_j$ for both $\pi_i \rhd H_i$ and $\pi_j \rhd H_j$.

We now proceed with the definition of linearization for the three kinds of plans. In the following, we shall write $\Pi_{i=1..k}\{\pi_i \rhd H_i\}$ for $\{\pi_1 \rhd H_1 \cdots \pi_k \rhd H_k\}$. Also, we shall always discard the components $\pi_i \rhd H_i$, if π_i is not well-formed.

4.1 Simple Plans

We linearize history expressions by applying from left to right the equations displayed in the following table. They are applicable in any context, because $\equiv$ is a congruence.

Equational properties of planned selections with simple plans

$$H \equiv \{0 \rhd H\} \tag{1}$$

$$\Pi_{i\in 1..k}\{\pi_i \rhd H_i\} \cdot \Pi_{j\in 1..p}\{\pi'_j \rhd H'_j\} \equiv \Pi_{\substack{i\in 1..k \\ j\in 1..p}}\{\pi_i \mid \pi'_j \rhd H_i \cdot H'_j\} \tag{2}$$

$$\Pi_{i\in 1..k}\{\pi_i \rhd H_i\} + \Pi_{j\in 1..p}\{\pi'_j \rhd H'_j\} \equiv \Pi_{\substack{i\in 1..k \\ j\in 1..p}}\{\pi_i \mid \pi'_j \rhd H_i + H'_j\} \tag{3}$$

$$\varphi[\{\pi_1 \rhd H_1 \cdots \pi_k \rhd H_k\}] \equiv \{\pi_1 \rhd \varphi[H_1] \cdots \pi_k \rhd \varphi[H_k]\} \tag{4}$$

$$\mu h.\, \{\pi_1 \rhd H_1 \cdots \pi_k \rhd H_k\} \equiv \{\pi_1 \rhd \mu h.\, H_1 \cdots \pi_k \rhd \mu h.\, H_k\} \tag{5}$$

$$\Pi_{i\in 1..k}\{\pi_i \rhd \Pi_{j\in 1..p_i}\{\pi'_{i,j} \rhd H_{i,j}\}\} \equiv \Pi_{\substack{i\in 1..k \\ j\in 1..p_i}}\{\pi_i \mid \pi'_{i,j} \rhd H_{i,j}\} \tag{6}$$

Example 7. Consider the scenario displayed in the leftmost figure above, where φ asks "never β after α". The extracted history expression is:

$$H = \varphi[\mu h.\, \{r[\ell_1] \rhd \alpha,\ r[\ell_2] \rhd \beta\} \cdot h]$$

Then, using equations (1), (2), (4) and (5) and the identity of 0, we obtain:

$$\begin{aligned} H &\equiv \varphi[\mu h.\, \{r[\ell_1] \rhd \alpha,\ r[\ell_2] \rhd \beta\} \cdot \{0 \rhd h\}] \\ &\equiv \varphi[\mu h.\, \{r[\ell_1] \mid 0 \rhd \alpha \cdot h,\ r[\ell_2] \mid 0 \rhd \beta \cdot h\}] \\ &= \varphi[\mu h.\, \{r[\ell_1] \rhd \alpha \cdot h,\ r[\ell_2] \rhd \beta \cdot h\}] \\ &\equiv \varphi[\{r[\ell_1] \rhd \mu h.\, \alpha \cdot h,\ r[\ell_2] \rhd \mu h.\, \beta \cdot h\}] \\ &\equiv \{r[\ell_1] \rhd \varphi[\mu h.\, \alpha \cdot h],\ r[\ell_2] \rhd \varphi[\mu h.\, \beta \cdot h]\} \end{aligned}$$

Note that, in the linearization of H, the request r will be resolved into the *same* service at each iteration. Even though in the original H you can choose a service among ℓ_1 and ℓ_2 at each iteration of the loop, simple plans (injective by definition) force you to decide once and for all which service will be chosen for r. Thus, also in this case linearization preserves the semantics of H. □

Example 8. Consider the scenario displayed in the rightmost figure above, where φ asks "never $\alpha\alpha$ nor $\beta\beta$". The extracted history expression is:

$$H = \varphi[\{r[\ell_1] \rhd \{r_1[\ell_2] \rhd \alpha, r_1[\ell_3] \rhd \beta\}\} \cdot \{r'[\ell_1] \rhd \{r_1[\ell_2] \rhd \alpha, r_1[\ell_3] \rhd \beta\}\}]$$

The linearization of H with simple plans is obtained as follows:

$$\begin{aligned} H &\equiv \varphi[\{r[\ell_1] \mid r_1[\ell_2] \rhd \alpha, r[\ell_1] \mid r_1[\ell_3] \rhd \beta\} \cdot \\ &\qquad \{r'[\ell_1] \mid r_1[\ell_2] \rhd \alpha, r'[\ell_1] \mid r_1[\ell_3] \rhd \beta\}] \\ &\equiv \varphi[\{r[\ell_1] \mid r_1[\ell_2] \mid r'[\ell_1] \mid r_1[\ell_2] \rhd \alpha \cdot \alpha,\ r[\ell_1] \mid r_1[\ell_2] \mid r'[\ell_1] \mid r_1[\ell_3] \rhd \alpha \cdot \beta, \\ &\qquad\quad r[\ell_1] \mid r_1[\ell_3] \mid r'[\ell_1] \mid r_1[\ell_2] \rhd \beta \cdot \alpha,\ r[\ell_1] \mid r_1[\ell_3] \mid r'[\ell_1] \mid r_1[\ell_3] \rhd \beta \cdot \beta\}] \\ &= \varphi[\{r[\ell_1] \mid r_1[\ell_2] \mid r'[\ell_1] \rhd \alpha \cdot \alpha, r[\ell_1] \mid r_1[\ell_3] \mid r'[\ell_1] \rhd \beta \cdot \beta\}] \\ &\equiv \{r[\ell_1] \mid r_1[\ell_2] \mid r'[\ell_1] \rhd \varphi[\alpha \cdot \alpha], r[\ell_1] \mid r_1[\ell_3] \mid r'[\ell_1] \rhd \varphi[\beta \cdot \beta]\} \end{aligned}$$

Note the use of idempotency of $\mid$, and the removal of the two choices with non-injective plan $r[\ell_1] \mid r_1[\ell_2] \mid r'[\ell_1] \mid r_1[\ell_3]$. No simple plan is viable for H, because neither $\varphi[\alpha \cdot \alpha]$ nor $\varphi[\beta \cdot \beta]$ are valid. □

4.2 Multi-choice Plans

The situation here is slightly trickier than above. The intuition is that each equation saturates the plans, by merging all the services associated with the same request. As an example, the saturation of $\{r[\ell_1] \rhd H_1, r[\ell_2] \rhd H_2\}$ is $\{r[\ell_1] \rhd H_1, r[\ell_2] \rhd H_2, r[\ell_1, \ell_2] \rhd H_1 + H_2\}$. To merge plans, we use the auxiliary operator $\oplus$, defined as follows:

$$\pi \oplus \pi' \ = \ \begin{cases} r[S \cup S'] \mid (\pi_0 \oplus \pi'_0) & \text{if } \pi = r[S] \mid \pi_0 \text{ and } \pi' = r[S'] \mid \pi'_0 \\ \pi \mid \pi' & \text{otherwise} \end{cases}$$

Given a planned selection $\{\pi_1 \rhd H_1 \cdots \pi_k \rhd H_k\}$, its saturation is composed by $\bigoplus_{i \in I} \pi_i \rhd \Sigma_{i \in I} H_i$, for each non-empty $I \subseteq 1..k$. In spite of the apparent

complexity, the equations below are much alike those for simple plans, with the addition of saturation.

Equational properties of planned selections with multi-choice plans

$$H \equiv \{0 \rhd H\} \tag{7}$$

$$\Pi_{i\in 1..k}\{\pi_i \rhd H_i\} \cdot \Pi_{j\in 1..p}\{\pi'_j \rhd H'_j\} \equiv \Pi_{\substack{\emptyset\subset I\subseteq 1..k\\ \emptyset\subset J\subseteq 1..p}}\{\bigoplus_{\substack{i\in I\\ j\in J}}(\pi_i \mid \pi'_j) \rhd \Sigma_{\substack{i\in I\\ j\in J}}(H_i \cdot H'_j)\} \tag{8}$$

$$\Pi_{i\in 1..k}\{\pi_i \rhd H_i\} + \Pi_{j\in 1..p}\{\pi'_j \rhd H'_j\} \equiv \Pi_{\substack{\emptyset\subset I\subseteq 1..k\\ \emptyset\subset J\subseteq 1..p}}\{\bigoplus_{\substack{i\in I\\ j\in J}}(\pi_i \mid \pi'_j) \rhd \Sigma_{\substack{i\in I\\ j\in J}}(H_i + H'_j)\} \tag{9}$$

$$\varphi[\Pi_{i\in 1..k}\{\pi_i \rhd H_i\}] \equiv \Pi_{\emptyset\subset I\subseteq 1..k}\{\bigoplus_{i\in I}\pi_i \rhd \varphi[\Sigma_{i\in I}H_i]\} \tag{10}$$

$$\mu h.\, \Pi_{i\in 1..k}\{\pi_i \rhd H_i\} \equiv \Pi_{\emptyset\subset I\subseteq 1..k}\{\bigoplus_{i\in I}\pi_i \rhd \mu h.\, \Sigma_{i\in I}H_i\} \tag{11}$$

$$\Pi_{i\in 1..k}\{\pi_i \rhd \Pi_{j\in 1..p_i}\{\pi'_{i,j} \rhd H_{i,j}\}\} \equiv \Pi_{\substack{\emptyset\subset I\subseteq 1..k\\ \emptyset\subset J\subseteq 1..p_i}}\{\bigoplus_{i\in I}\pi_i \mid \bigoplus_{j\in J}\pi'_{i,j} \rhd \Sigma_{\substack{i\in I\\ j\in J}}H_{i,j}\} \tag{12}$$

Example 9. Let $H = \varphi[\mu h.\, \{r[\ell_1] \rhd \alpha,\ r[\ell_2] \rhd \beta\} \cdot h]$. Then, using equations (7), (8), (10) and (11) and the identity of the plan 0, we obtain:

$$\begin{aligned}
H &\equiv \varphi[\mu h.\, \{r[\ell_1] \rhd \alpha,\ r[\ell_2] \rhd \beta\} \cdot \{0 \rhd h\}] \\
&\equiv \varphi[\mu h.\, \{r[\ell_1] \mid 0 \rhd \alpha \cdot h,\ r[\ell_2] \mid 0 \rhd \beta \cdot h, (r[\ell_1] \oplus r[\ell_2]) \mid 0 \rhd (\alpha+\beta)\cdot h\}] \\
&= \varphi[\mu h.\, \{r[\ell_1] \rhd \alpha \cdot h,\ r[\ell_2] \rhd \beta \cdot h, r[\ell_1,\ell_2] \rhd (\alpha+\beta)\cdot h\}] \\
&\equiv \varphi[\{r[\ell_1] \rhd \mu h.\, \alpha \cdot h,\ r[\ell_2] \rhd \mu h.\, \beta \cdot h,\ r[\ell_1,\ell_2] \rhd \mu h.\, (\alpha+\beta)\cdot h\}] \\
&\equiv \{r[\ell_1] \rhd \varphi[\mu h.\, \alpha \cdot h],\ r[\ell_2] \rhd \varphi[\mu h.\, \beta \cdot h],\ r[\ell_1,\ell_2] \rhd \varphi[\mu h.\, (\alpha+\beta)\cdot h]\}
\end{aligned}$$

Note that, unlike Example 7, if $\varphi[\mu h.\, (\alpha+\beta)\cdot h]$ is valid, then you can resolve the request r into either ℓ_1 or ℓ_2 at *each* iteration of the loop. □

4.3 Dependent Plans

The main difference with the previous cases is revealed by the history expression $\{r[\ell] \rhd \{r'[\ell_1] \rhd \alpha_1, r'[\ell_2] \rhd \alpha_2\}\}$ (taken from Section 3.5), whose linearization $\{r[\ell.r'[\ell_1]] \rhd \alpha_1, r[\ell.r'[\ell_2]] \rhd \alpha_2\}$ differs e.g. from the linearization for simple plans, i.e. $\{r[\ell] \mid r'[\ell_1] \rhd \alpha_1, r[\ell] \mid r'[\ell_2] \rhd \alpha_2\}$. To account for that, we introduce the operator $\odot$ between depenend plans as follows:

$$0 \odot \pi = \pi \qquad r[\ell.\pi'] \odot \pi = r[\ell.\pi' \odot \pi] \qquad (\pi_0 \mid \pi_1) \odot \pi = (\pi_0 \odot \pi) \mid (\pi_1 \odot \pi)$$

$$H \equiv \{0 \rhd H\} \tag{13}$$
$$\Pi_{i\in 1..k}\{\pi_i \rhd H_i\} \cdot \Pi_{j\in 1..p}\{\pi'_j \rhd H'_j\} \equiv \Pi_{\substack{i\in 1..k\\ j\in 1..p}}\{\pi_i \mid \pi'_j \rhd H_i \cdot H'_j\} \tag{14}$$
$$\Pi_{i\in 1..k}\{\pi_i \rhd H_i\} + \Pi_{j\in 1..p}\{\pi'_j \rhd H'_j\} \equiv \Pi_{\substack{i\in 1..k\\ j\in 1..p}}\{\pi_i \mid \pi'_j \rhd H_i + H'_j\} \tag{15}$$
$$\varphi[\{\pi_1 \rhd H_1 \cdots \pi_k \rhd H_k\}] \equiv \{\pi_1 \rhd \varphi[H_1] \cdots \pi_k \rhd \varphi[H_k]\} \tag{16}$$
$$\mu h.\, \{\pi_1 \rhd H_1 \cdots \pi_k \rhd H_k\} \equiv \{\pi_1 \rhd \mu h.\, H_1 \cdots \pi_k \rhd \mu h.\, H_k\} \tag{17}$$
$$\Pi_{i\in 1..k}\{\pi_i \rhd \Pi_{j\in 1..p_i}\{\pi'_{i,j} \rhd H_{i,j}\}\} \equiv \Pi_{\substack{i\in 1..k\\ j\in 1..p_i}}\{\pi_i \odot \pi'_{i,j} \rhd H_{i,j}\} \tag{18}$$

where both operands of (14) and (15) are in linear form.

The side condition in equations (14) and (15) is easily fulfilled: it suffices to apply the (oriented) equations with the leftmost-innermost evaluation rule.

Example 10. Let $H = \{r[\ell_0] \rhd \alpha_0\} \cdot \{0 \rhd \{r'[\ell_1] \rhd \alpha_1, r'[\ell_2] \rhd \alpha_2\}\}$, and let $H' = \{r[\ell] \mid 0 \rhd \{\alpha_0 \cdot \{r'[\ell_1] \rhd \alpha_1, r'[\ell_2] \rhd \alpha_2\}\}\}$. Then, $H \not\equiv H'$. For instance, with $\pi = r[\ell_0] \mid r'[\ell_1]$, we have that:

$$\begin{aligned}
[\![H]\!]^\pi &= [\![\{r[\ell_0] \rhd \alpha_0\}]\!]^\pi [\![\{0 \rhd \{r'[\ell_1] \rhd \alpha_1, r'[\ell_2] \rhd \alpha_2\}\}]\!]^\pi \\
&= [\![\alpha_0]\!]^0 [\![\{r'[\ell_1] \rhd \alpha_1, r'[\ell_2] \rhd \alpha_2\}]\!]^\pi \\
&= \alpha_0\big([\![\{r'[\ell_1] \rhd \alpha_1\}]\!]^\pi \cup [\![\{r'[\ell_2] \rhd \alpha_2\}]\!]^\pi\big) \\
&= \alpha_0([\![\alpha_1]\!]^0 \cup \bot) \;=\; \alpha_0\alpha_1 \\
[\![H']\!]^\pi &= [\![\{r[\ell] \mid 0 \rhd \{\alpha_0 \cdot \{r'[\ell_1] \rhd \alpha_1, r'[\ell_2] \rhd \alpha_2\}\}\}]\!]^\pi \\
&= [\![\alpha_0 \cdot \{r'[\ell_1] \rhd \alpha_1, r'[\ell_2] \rhd \alpha_2\}]\!]^0 \\
&= [\![\alpha_0]\!]^0 [\![\{r'[\ell_1] \rhd \alpha_1, r'[\ell_2] \rhd \alpha_2\}]\!]^0 \;=\; \bot
\end{aligned}$$

where we have used the fact that $(r[\ell_0] \mid 0)/\pi = (r[\ell_0]/\pi) \sqcap (0/\pi) = 0 \sqcap \pi = 0$.

Example 11. Consider again, from Example 8, the history expression:

$$H = \varphi[\{r[\ell_1] \rhd \{r_1[\ell_2] \rhd \alpha, r_1[\ell_3] \rhd \beta\}\} \cdot \{r'[\ell_1] \rhd \{r_1[\ell_2] \rhd \alpha, r_1[\ell_3] \rhd \beta\}\}]$$

The linearization of H with dependent plans is obtained as follows:

$$\begin{aligned}
H &\equiv \varphi[\{r[\ell_1] \odot r_1[\ell_2] \rhd \alpha, r[\ell_1] \odot r_1[\ell_3] \rhd \beta\} \cdot \\
&\qquad \{r'[\ell_1] \odot r_1[\ell_2] \rhd \alpha, r'[\ell_1] \odot r_1[\ell_3] \rhd \beta\}] \\
&\equiv \varphi[\{r[\ell_1.r_1[\ell_2]] \rhd \alpha, r[\ell_1.r_1[\ell_3]] \rhd \beta\} \cdot \{r'[\ell_1.r_1[\ell_2]] \rhd \alpha, r'[\ell_1.r_1[\ell_3]] \rhd \beta\}] \\
&\equiv \varphi[\{r[\ell_1.r_1[\ell_2]] \mid r'[\ell_1.r_1[\ell_2]] \rhd \alpha \cdot \alpha,\ r[\ell_1.r_1[\ell_2]] \mid r'[\ell_1.r_1[\ell_3]] \rhd \alpha \cdot \beta \\
&\qquad r[\ell_1.r_1[\ell_3]] \mid r'[\ell_1.r_1[\ell_2]] \rhd \beta \cdot \alpha,\ r[\ell_1.r_1[\ell_3]] \mid r'[\ell_1.r_1[\ell_3]] \rhd \beta \cdot \beta\}] \\
&\equiv \{r[\ell_1.r_1[\ell_2]] \mid r'[\ell_1.r_1[\ell_2]] \rhd \varphi[\alpha \cdot \alpha],\ r[\ell_1.r_1[\ell_2]] \mid r'[\ell_1.r_1[\ell_3]] \rhd \varphi[\alpha \cdot \beta], \\
&\qquad r[\ell_1.r_1[\ell_3]] \mid r'[\ell_1.r_1[\ell_2]] \rhd \varphi[\beta \cdot \alpha],\ r[\ell_1.r_1[\ell_3]] \mid r'[\ell_1.r_1[\ell_3]] \rhd \varphi[\beta \cdot \beta]\}
\end{aligned}$$

If φ asks "never $\alpha\alpha$ nor $\beta\beta$", then both $\varphi[\alpha \cdot \beta]$ and $\varphi[\beta \cdot \alpha]$ are valid, and so the plans $r[\ell_1.r_1[\ell_2]] \mid r'[\ell_1.r_1[\ell_3]]$ and $r[\ell_1.r_1[\ell_3]] \mid r'[\ell_1.r_1[\ell_2]]$ are viable for H. □

5 Conclusions

We have studied how to statically devise plans for service composition. Three kinds of plans have been considered: simple plans, that choose exactly one service for each request, multi-choice plans, that choose among a set of services for each request, and dependent plans, that also exploit the knowledge of past choices. Given a correct static approximation of a program behaviour, we have been able to determine the plans under which *all* the executions of that program obey the security constraints imposed by the user, so dispensing from an execution monitor.

A possible direction for future work is to consider other kinds of plans. Mixing multi-choice and dependent plans should be straightforward; a more ambitious goal is to study plans that can be updated at run-time. This would allow to overcome some limitations of statically-determined plans. For instance, consider a service environment $\{\ell_1 : \alpha,\ \ell_2 : \beta\}$, and a policy φ asking "never $\alpha\alpha$ nor $\beta\beta$". Then, the program $\varphi[\mathtt{let}\, f\, =\, \mathtt{req}_r\, \mathtt{in}\, (\mathtt{if}\, b\, \mathtt{then}\, \alpha; f()\, \mathtt{else}\, \beta; f())]$ admits no viable plan. Note however that we could easily devise a viable plan at run-time, once the boolean guard b has been evaluated.

Another way to extend our work is to design an orchestration language, building upon [3]. The selection mechanism would match properties of service *behaviour*, rather than service signatures, as happens in standard orchestration languages. Also, the machinery deployed here should be refined to cope with distribution: indeed, in real orchestration languages global histories make little sense, since histories are local to the various sites.

Acknowledgments

We wish to thank Roberto Zunino and the anonymous referees for their insightful comments. Research partially supported by the Project FET-GC II Sensoria.

References

[1] T. Andrews et al. *Business Process Execution Language for Web Services (BPEL4WS), Version 1.1*, 2003.

[2] Massimo Bartoletti, Pierpaolo Degano, and Gian Luigi Ferrari. Plans for service composition. Available at `www.di.unipi.it/~bartolet/plans.pdf`.

[3] Massimo Bartoletti, Pierpaolo Degano, and Gian Luigi Ferrari. Enforcing secure service composition. In *Proc. 18th Computer Security Foundations Workshop (CSFW)*, 2005.

[4] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarane. The next step in web services. *Communications of the ACM*, 46(10), 2003.

[5] R. Khalaf, N. Mukhi, and S. Weerawarana. Service oriented composition in BPEL4WS. In *Proc. WWW*, 2003.

[6] Mike P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *WISE*, 2003.

[7] M. Papazouglou and D. Georgakopoulos. Special issue on service oriented computing. *Communications of the ACM*, 46(10), 2003.

[8] Glynn Winskel. *The Formal Semantics of Programming Languages*. The MIT Press, 1993.