A Static Approach to Secure Service Composition

Massimo Bartoletti
Pierpaolo Degano
Gian-Luigi Ferrari
Roberto Zunino

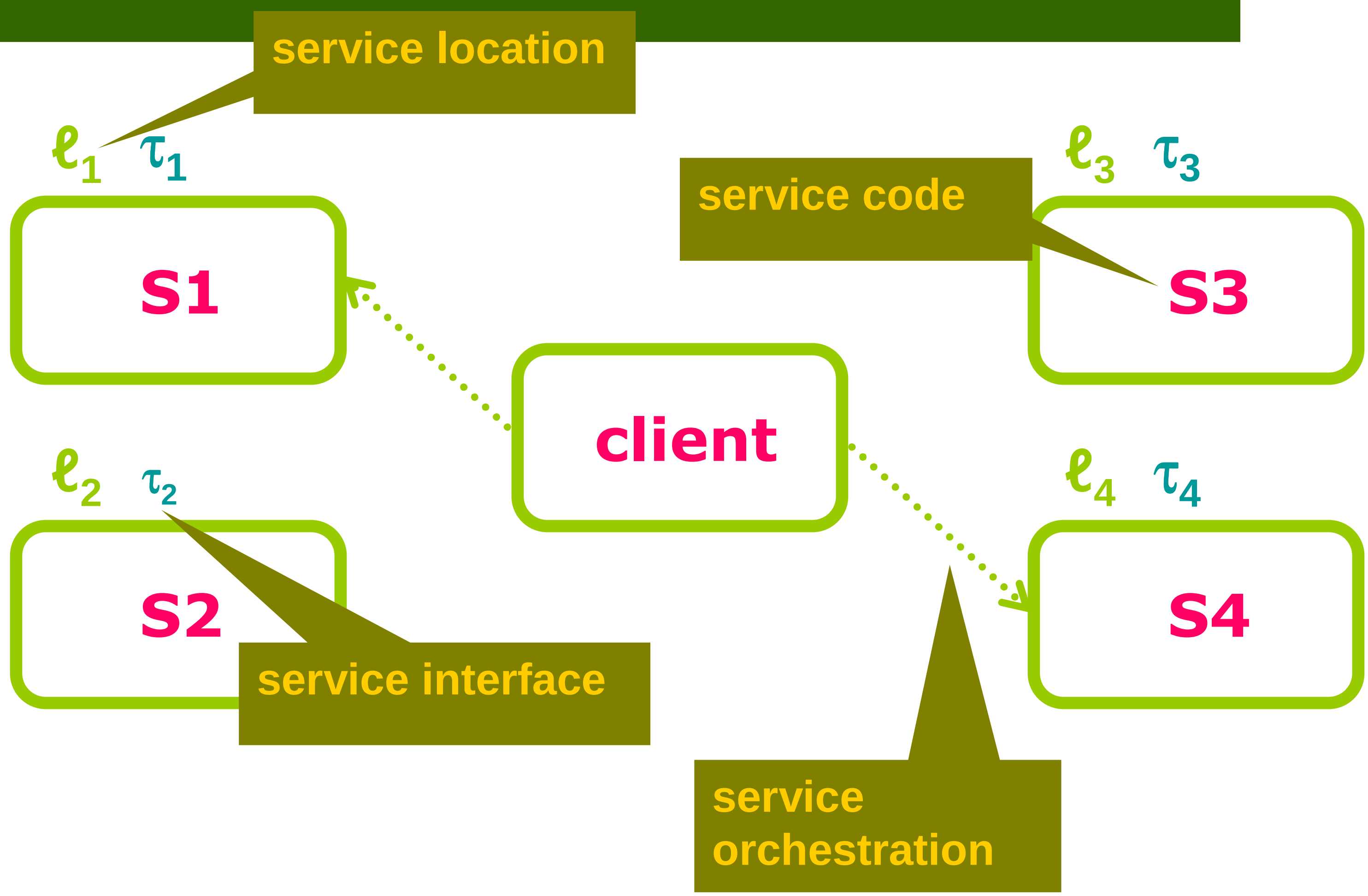
Dipartimento di Informatica
Università di Pisa

Summary

- **Overview**
  - **issues in secure service composition**
  - **safety framings and policies**
  - **req-by-contract for service request**
  - **plans for secure orchestration**
- A calculus for service composition
  - syntax and operational semantics
  - type & effect system
- Plans & Orchestration
  - constructions of plans and linearization
  - model-checking viable plans

Programming in a world of services

Programming in a world of services



Security and service composition

- two kinds of security concerns:
  - secrecy of transmitted data, authentication, etc (protocol analysis techniques)
  - control on computational resources (access control, resource usage analysis, information flow control, etc)
- need for linguistic mechanisms that:
  - work in a distributed setting
  - assume no trust relations among services
  - can also cope with *mobile code*

Security and service composition: safety framings

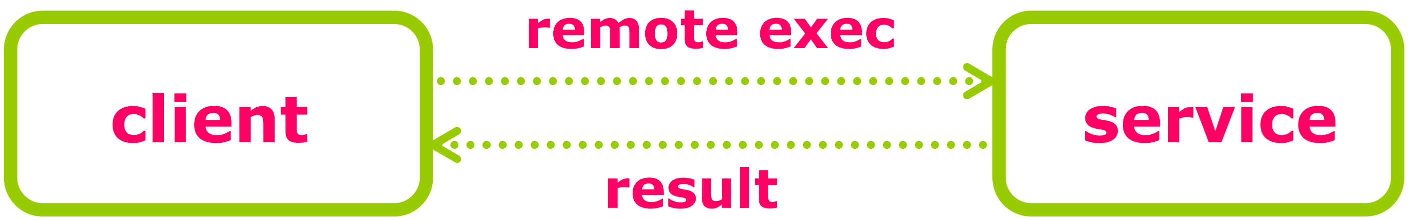
Client wants to protect from untrusted results

Linguistic mechanism: **safety framing**

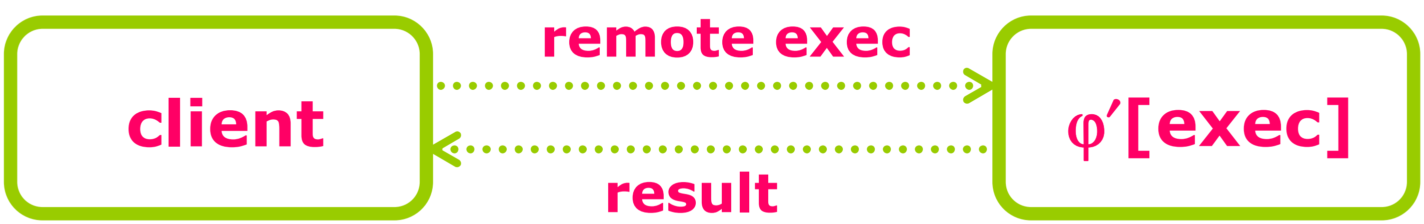
The **policy φ** is enforced stepwise within its **scope**

Security and service composition: safety framings

Similarly, services want to protect from clients



(now the **safety framing** belongs to the service)



Scoped policies check the **local execution histories**

Security and service composition: service selection

Req-by-name: request a *given* service among many

Why **S2** and not **S1** or **S3**, if all functionally equivalent ?

Security and service composition: service selection

Problems with “request by name”:

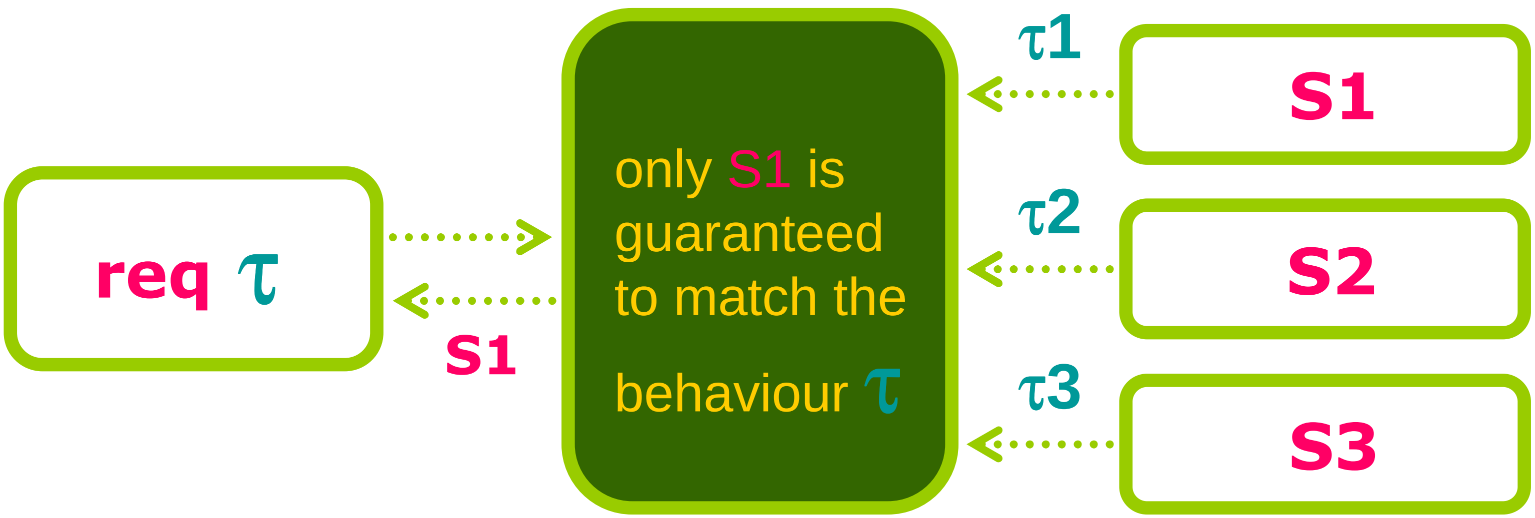
- what if named service **S2** becomes unavailable ?
- …and if **S2** is outperformed by **S1** or **S3** ?
- hard reasoning about non-functional properties of services (e.g. security)
- security level independent of the execution context (unless hard-wired in the service code)

From syntax-based to semantics-based invocation

Service names **ℓ,ℓ′,..** tell me nothing about the behaviour!

Security and service composition: service selection

Req-by-contract: request a service respecting **the desired behaviour**



τ imposes both functional and non-functional constraints

Use cases for Req-by-contract

Example: download an applet that obeys the policy φ

$$\mathbf{req}\ \ \tau_0 \longrightarrow (\tau_1 \xrightarrow{\varphi[\bullet]} \tau_2)$$

Example: a remote executer that obeys the policy φ'

$$\mathbf{req}\ \ (\tau_0 \longrightarrow \tau_1) \xrightarrow{\varphi'[\bullet]} \tau_2$$

Observable behaviour

- **access events** are the actions relevant for security (e.g. read/write local files, invoke/be invoked by a given service, etc)
  - mechanically inferred, or inserted by programmer.
  - their meaning is fixed globally.
  - access events cannot be hidden.
- the (abstract) behaviour observable by the orchestrator over-approximates the **run-time histories**, i.e. sequences of access events (via a type & effect system).

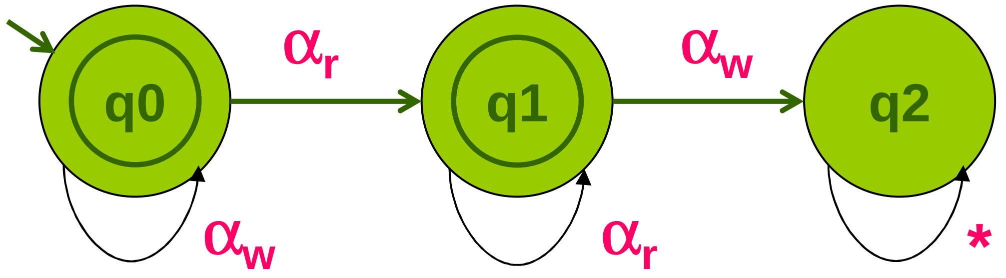
What kind of policies ?

- History-based security
- Policies φ are **regular** properties event histories
- Policies φ, φ' have a **local scope**, possibily **nested** $\varphi[\cdot\cdot\varphi'[\cdot\cdot]\cdot\cdot]$
- A policy can only control histories of a **single** site (no trust among services)
- Histories are **local** to **stateless** service sites (stateful easy)

Example: the Chinese Wall policy

φ Chinese Wall: cannot write (α_w) after read (α_r)

A_φ



q2 non-final sink

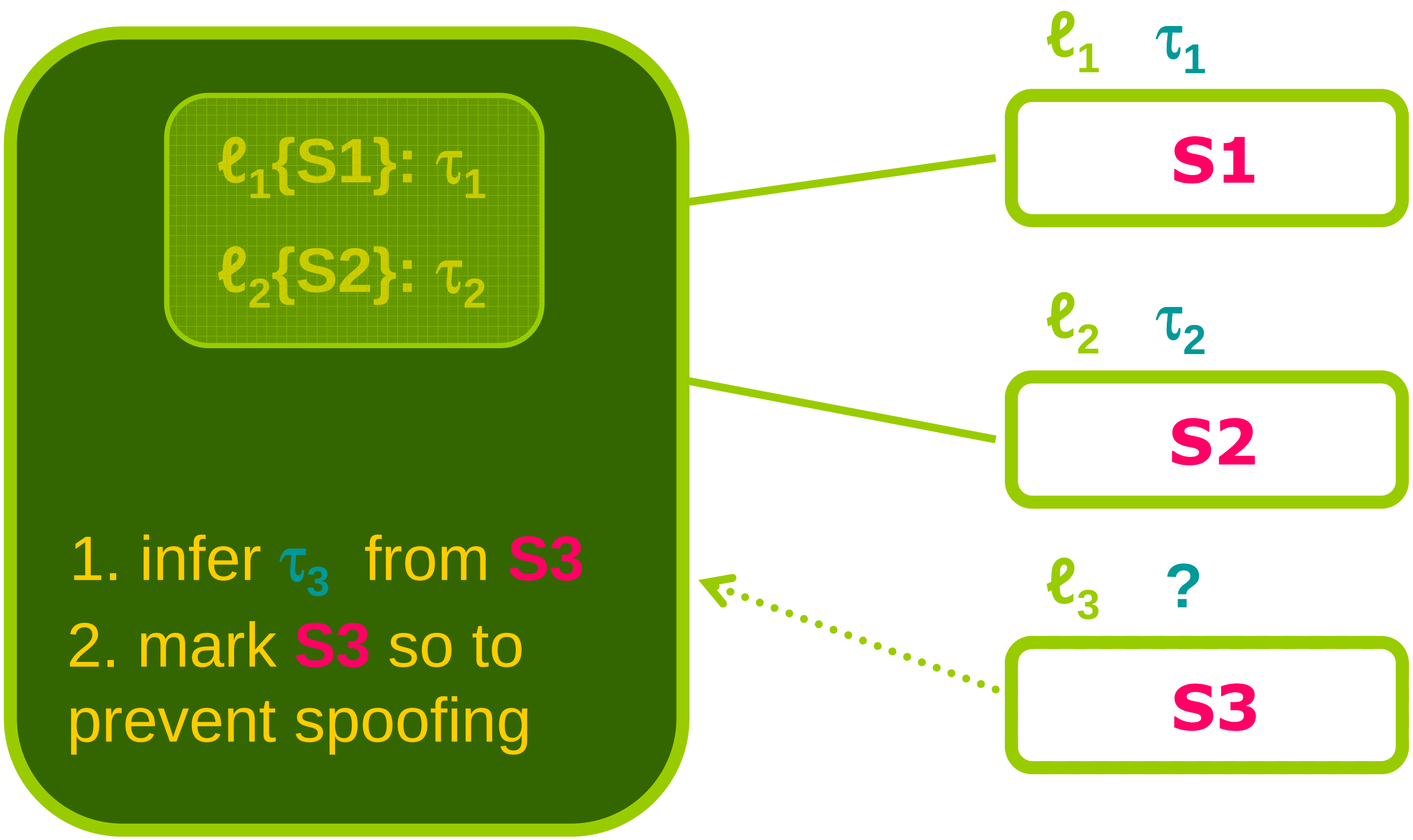
$\alpha_w \, \alpha_r \, \alpha_w \not\models \varphi$

Principle of Least Privilege

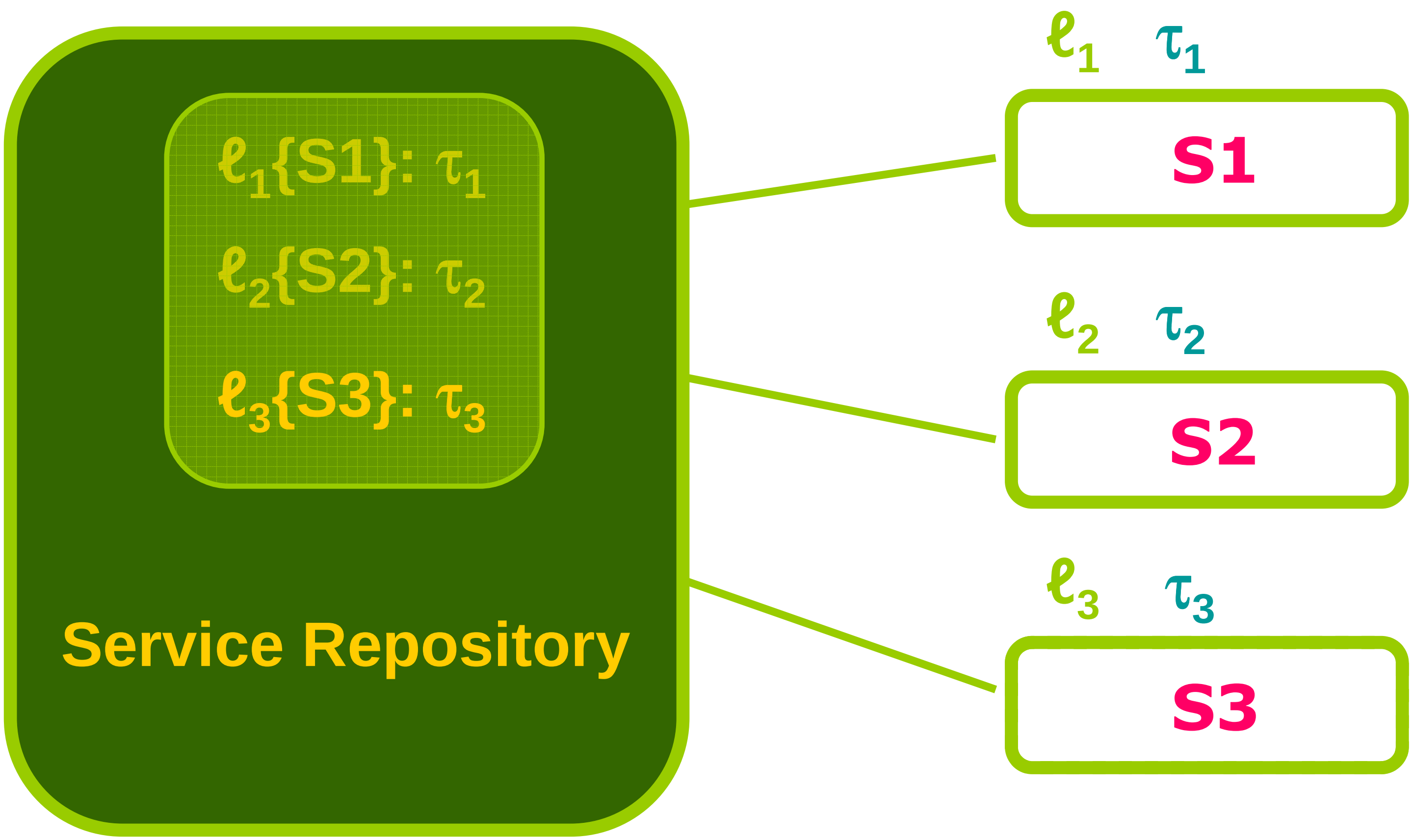
“Programs should be granted the minimum set of rights needed to accomplish their task”

- A service must always obey all the active policies (no policy override)
- Policies can always inspect the whole past history (no event can be discarded)
- “Privileged calls” implemented by policies that explicitly discard the past

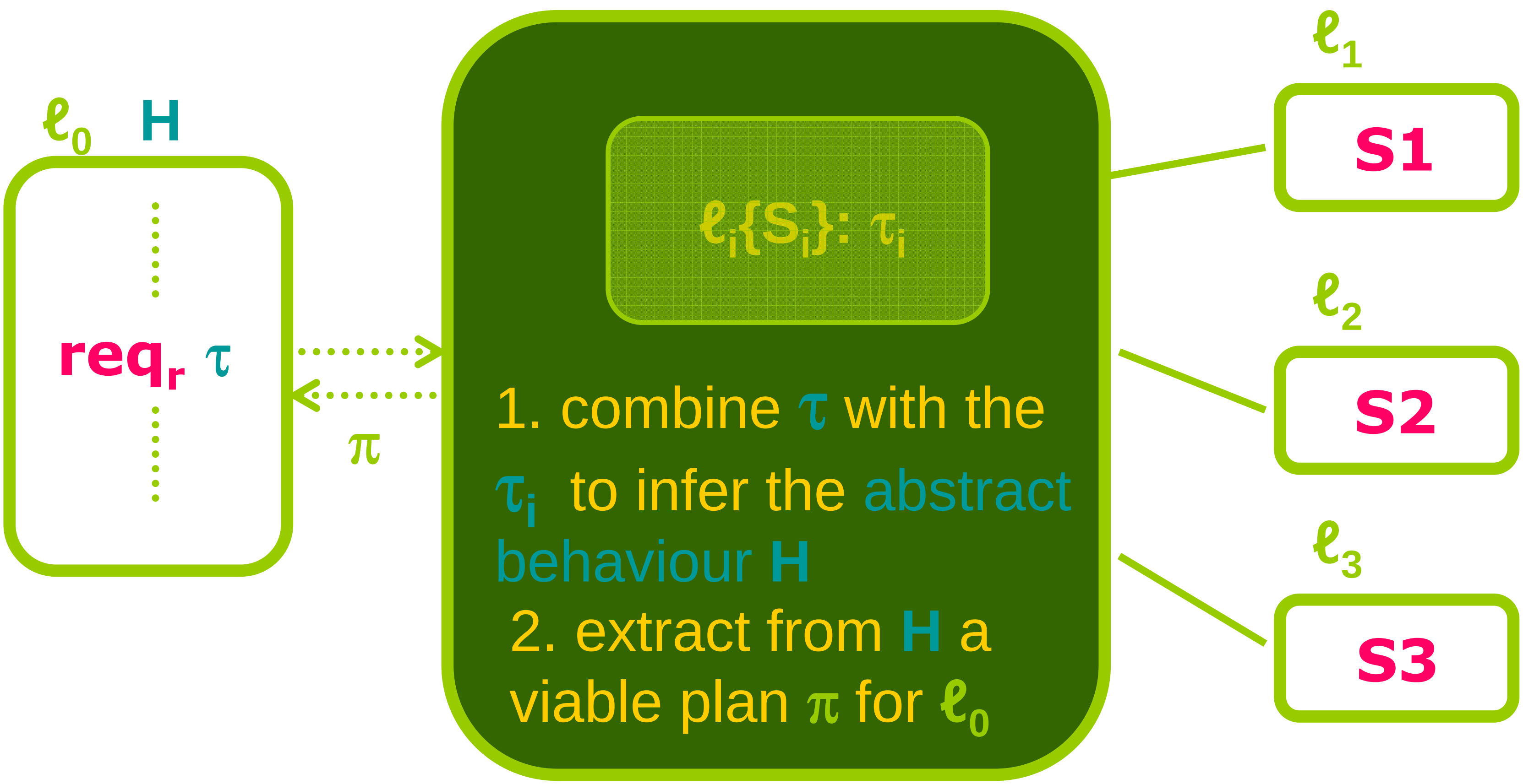
Service publication (1)



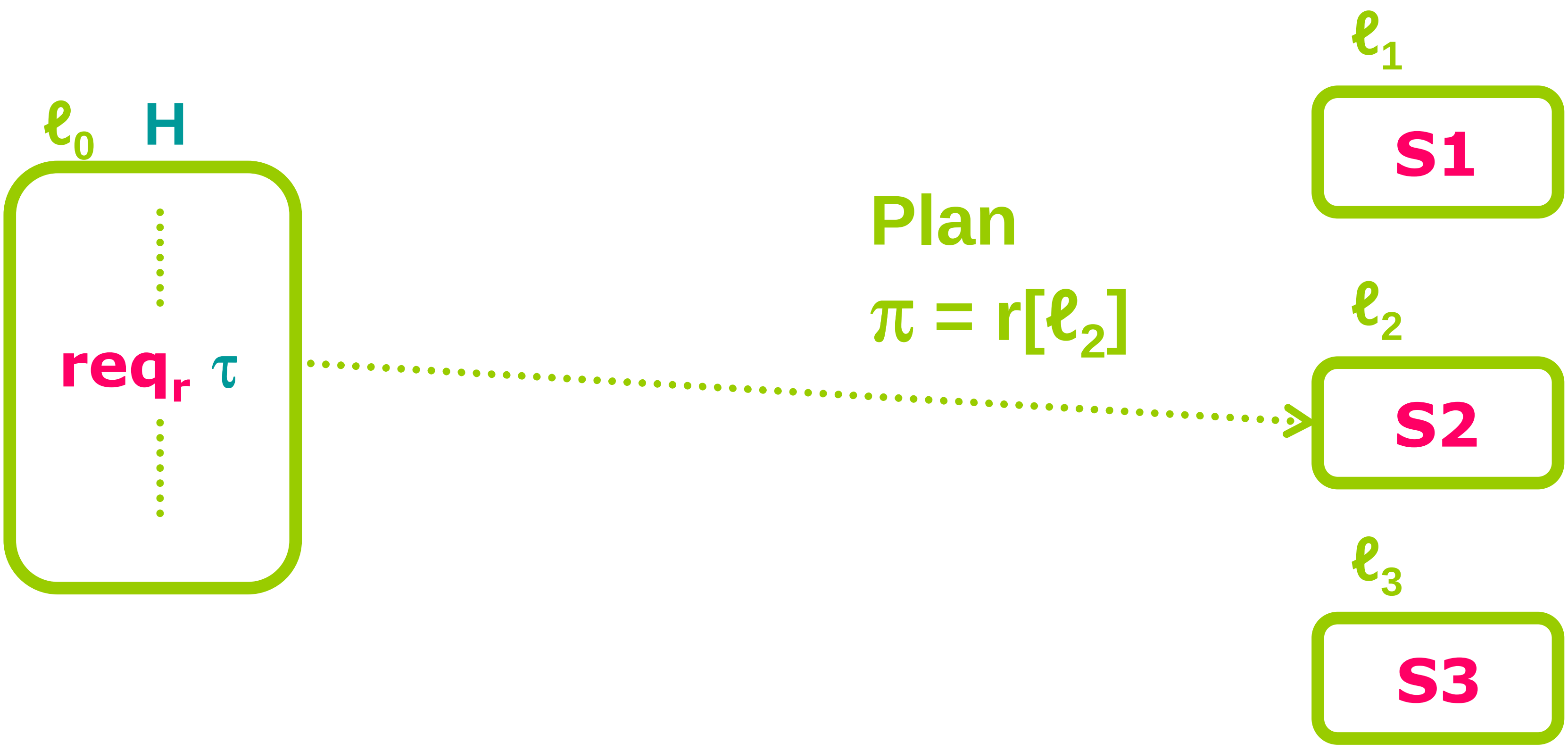
Service publication (2)



Service orchestration



Service orchestration



Names are only known by the orchestrator!

What is a plan ?

- A plan drives the execution of an application, by associating each service request with one (or more) appropriate services
- With a **viable plan**:
  - executions never violate policies
  - there are no unresolved requests
  - you can then dispose from any execution monitoring!
- Many kinds of plans:
  - **Simple:** one service for each request
  - **Multi-choice:** more services for each request
  - **Dependent:** one service, and a continuation plan
  - **...**

Who do we trust ?

The orchestrator, that:

- certifies the behavioural descriptions of services **(types annotated with effects H)**
- composes the descriptions, and ensures that selected services match the requested types
- extracts the **viable plans** (through model-checking)

Also, someone must ensure that services do not change their code on-the-fly

Summing up…

- a calculus for secure service composition:
  - **distributed** services
  - **safety framings** scoped policies on localized execution histories
  - **req-by-contract** service invocation
- static orchestrator:
  - certifies the **behavioural interfaces** of services
  - provides a client with the **viable plans** driving secure executions

What's next

- calculus: syntax and **operational semantics**
- static semantics: **type & effect system**
  - **types** carry annotations H about service behaviour
  - **effects** H are history expressions, which over-approximate the actual execution histories
- extracting viable plans:
  - **linearization:** unscrambling the structure of H
  - **model checking:** valid plans are viable

Services

Services	**e ::=**	**x**	variable
		α	access event
		if b then e else e′	conditional
		$\lambda_z x.e$	abstraction
		e e′	application
		φ**[e]**	safety framing
		req$_r$ τ	service request
(only in configs)		**wait** ℓ	wait reply

Networks

location

service code and published interface

$N ::= \ell\{e:\tau\}: \eta, e'$	published service
$N \parallel N'$	composition

running code

execution history

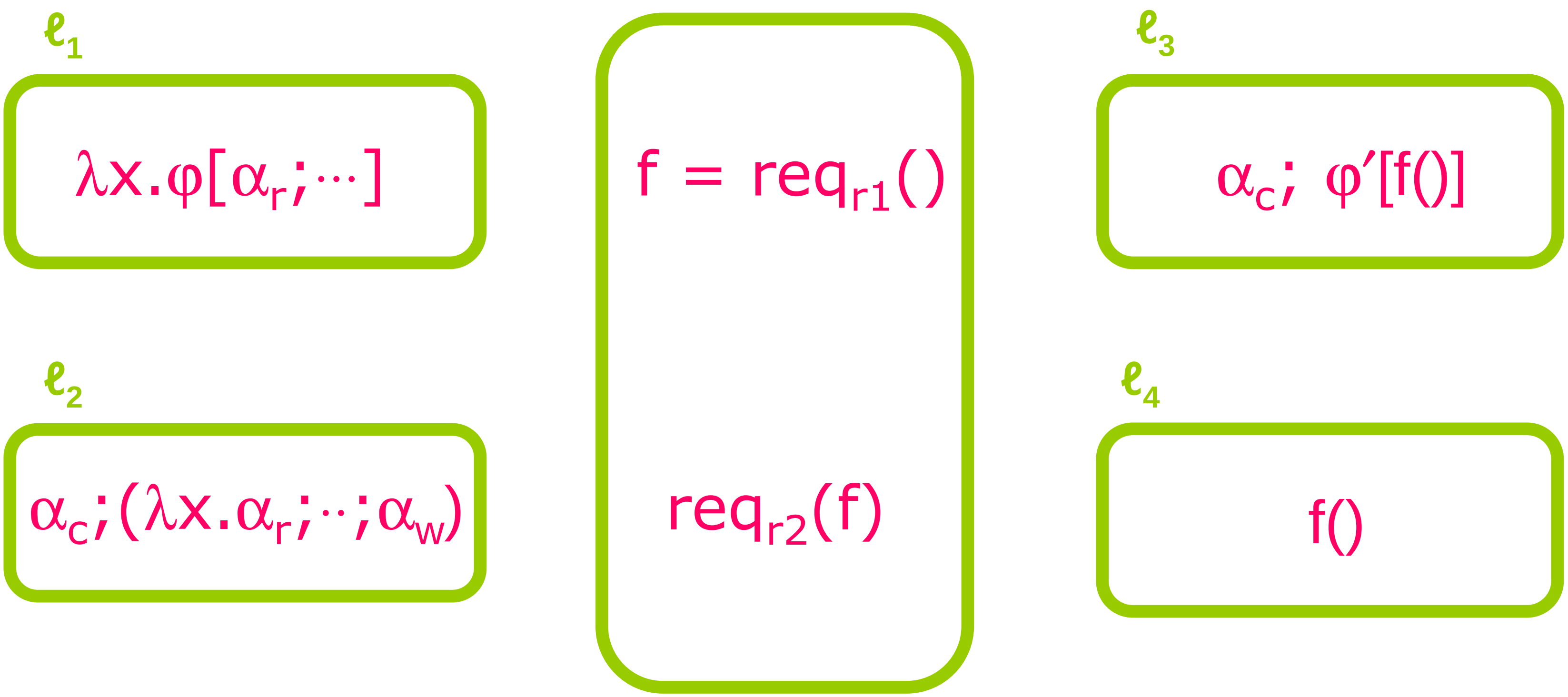
(Simple) Plans

A **plan** is a function from requests **r** to services **ℓ**

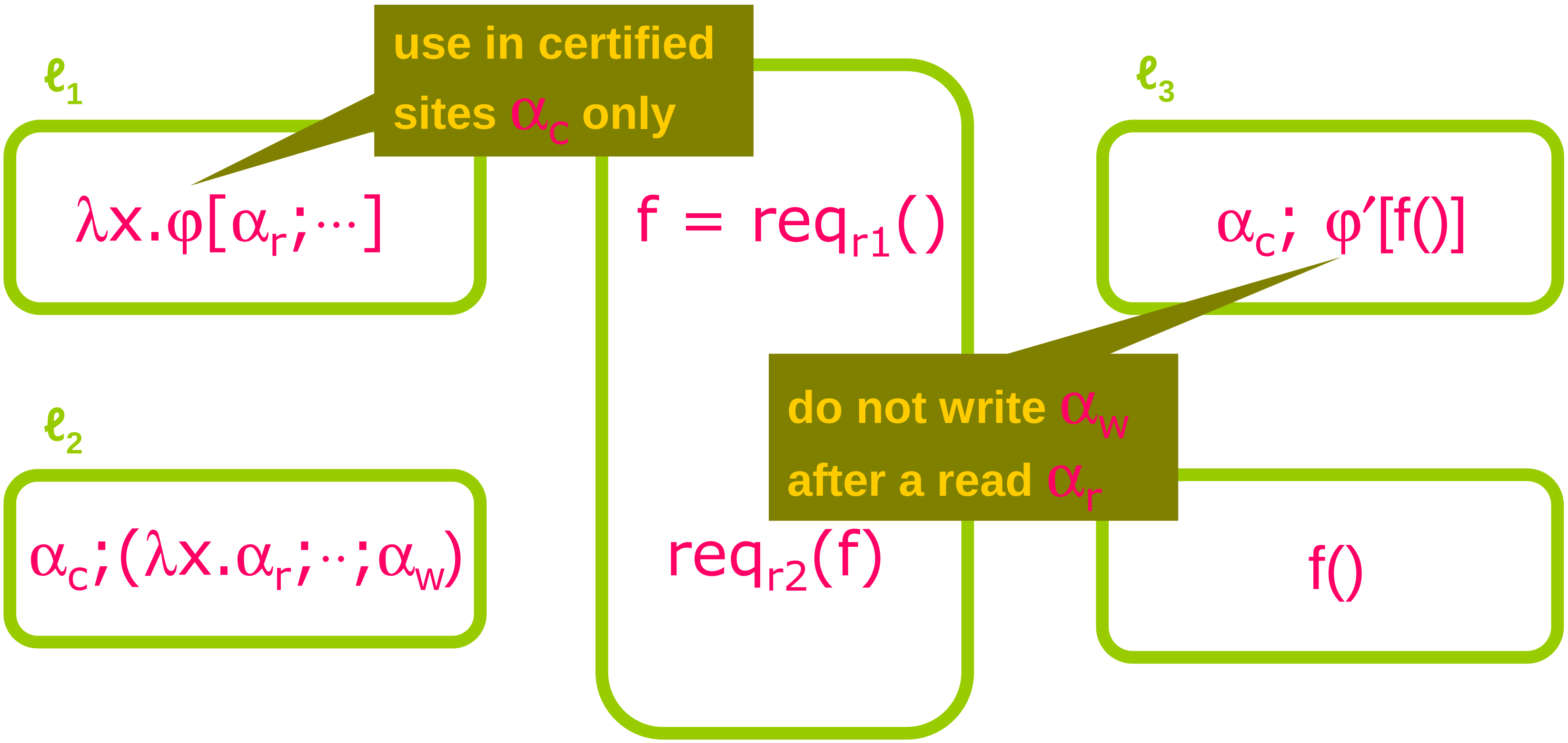
π	::=	**0**	empty
		r[ℓ]	service choice
		$\pi \mid \pi'$	composition

Plans respect the partial knowledge **ℓ < ℓ′** of services about the network (**<** is a partial ordering)

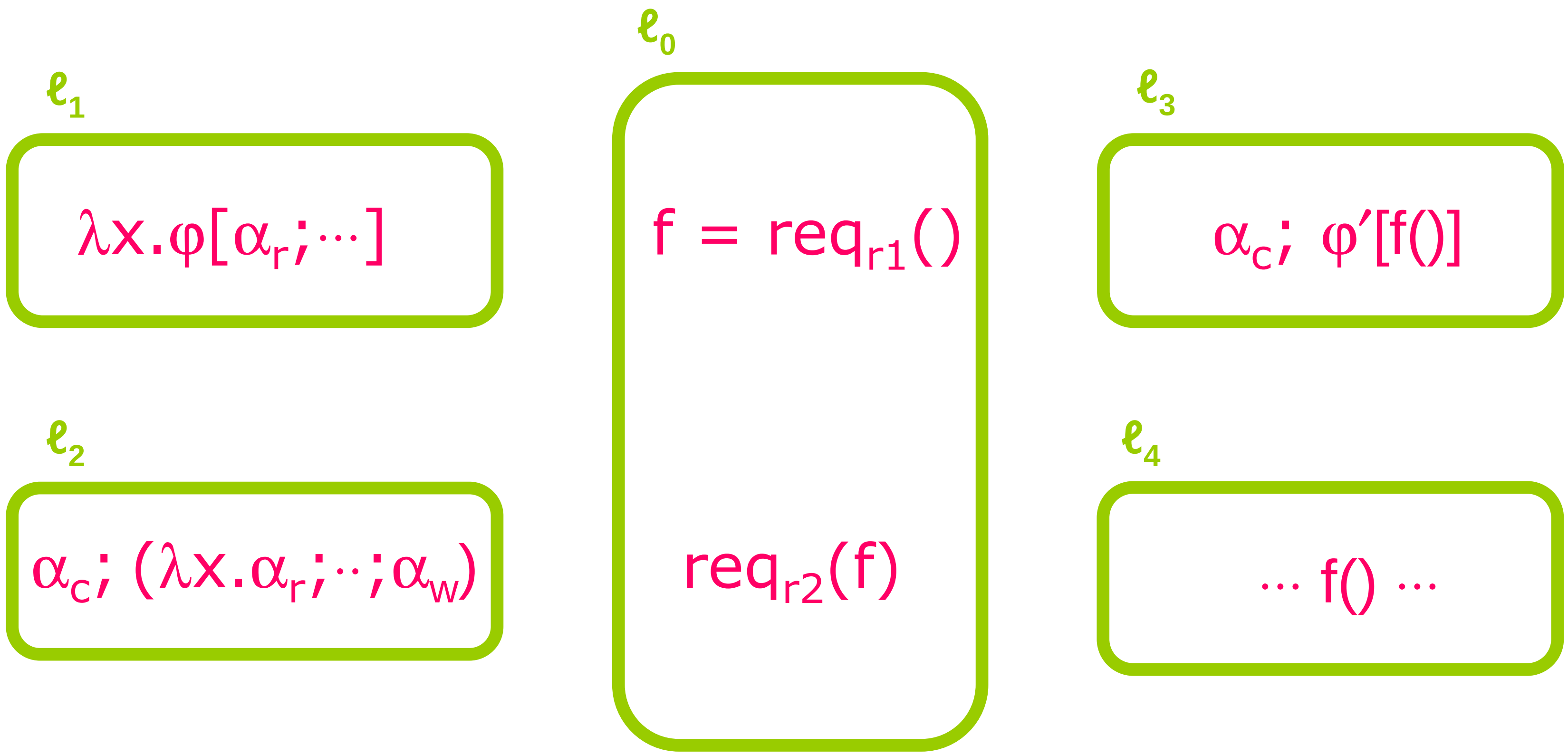
Example: delegating code execution



Example: delegating code execution

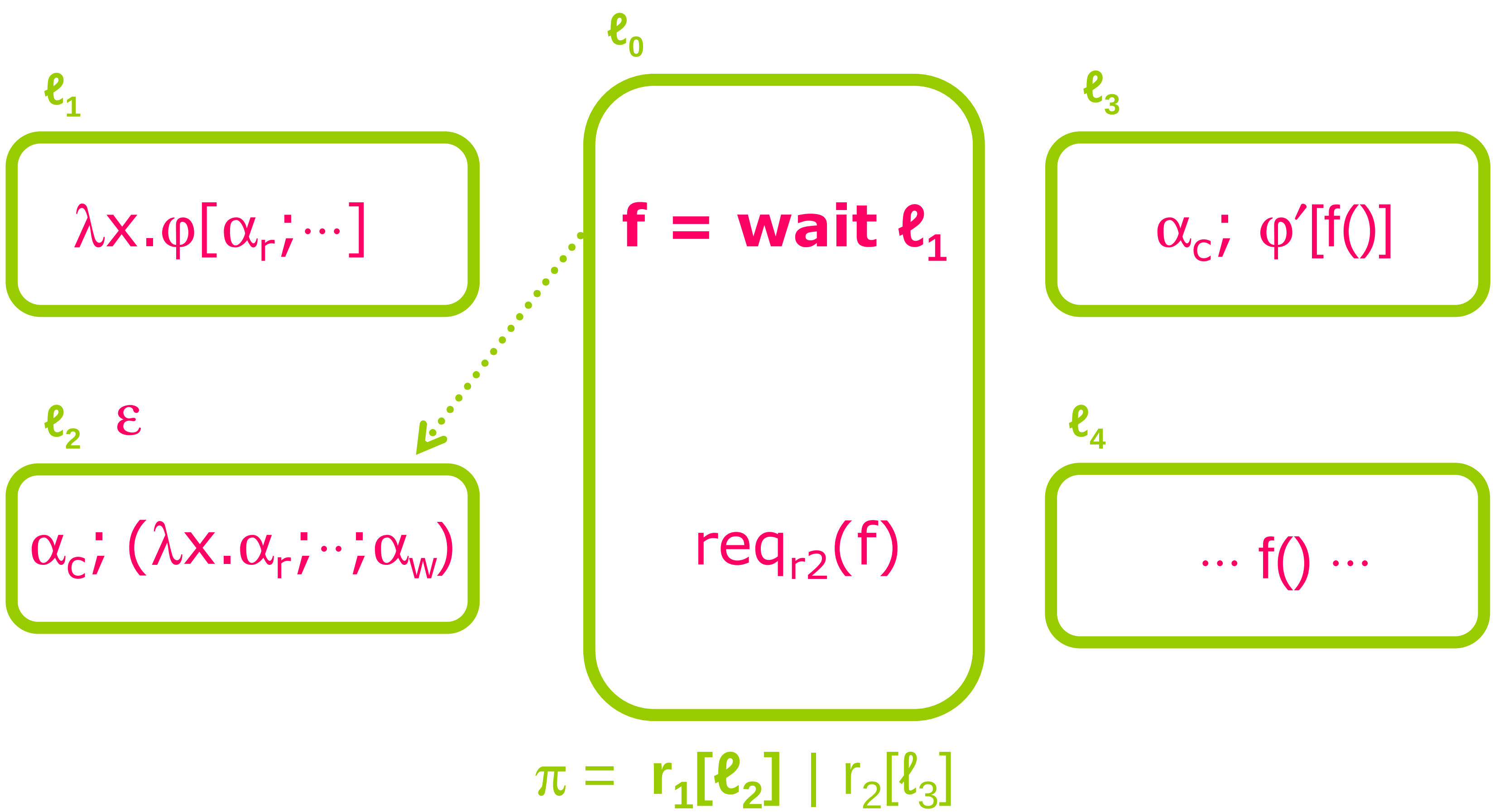


Executing a network of services

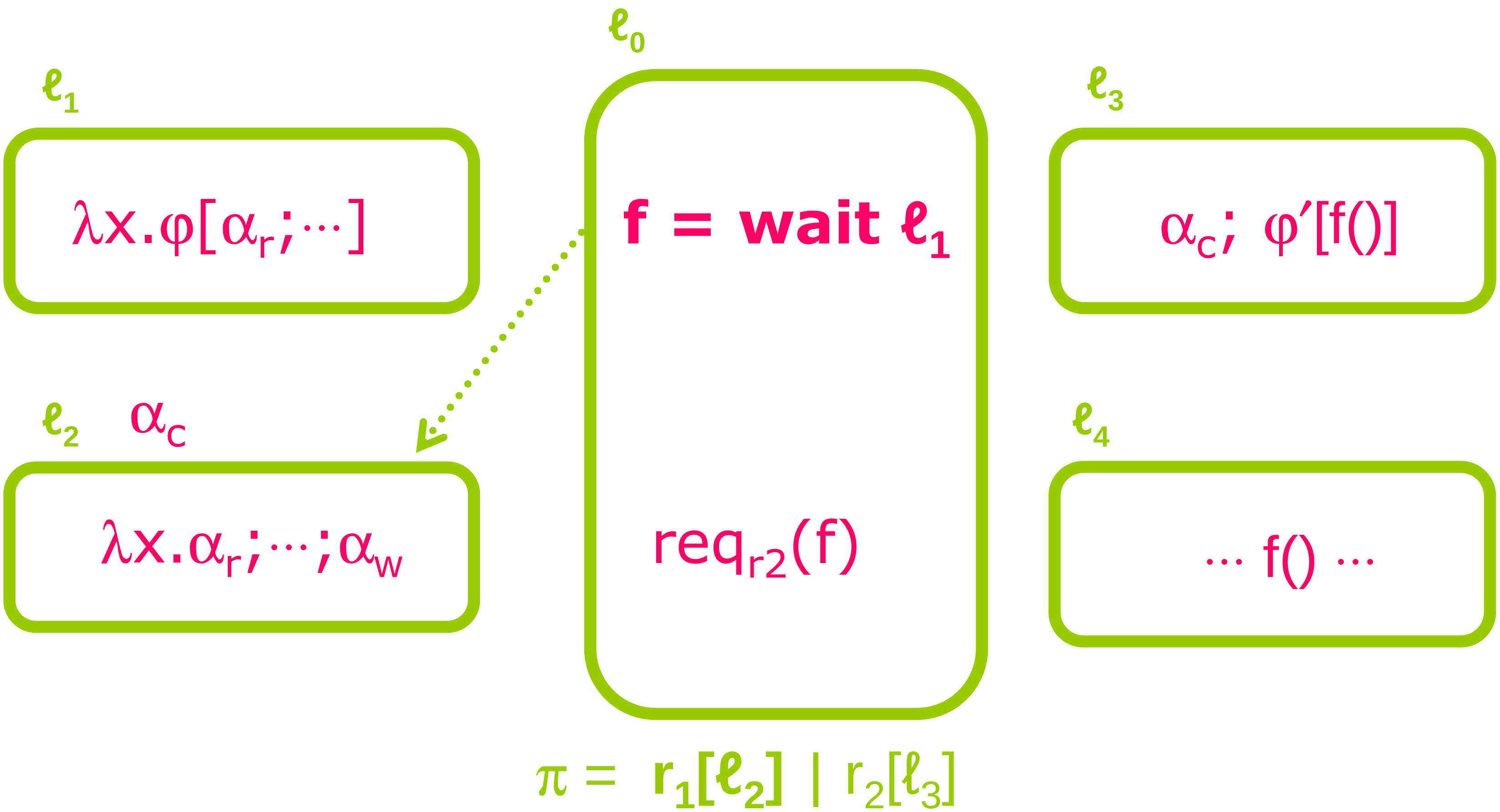


$\pi = r_1[\ell_2] \mid r_2[\ell_3]$

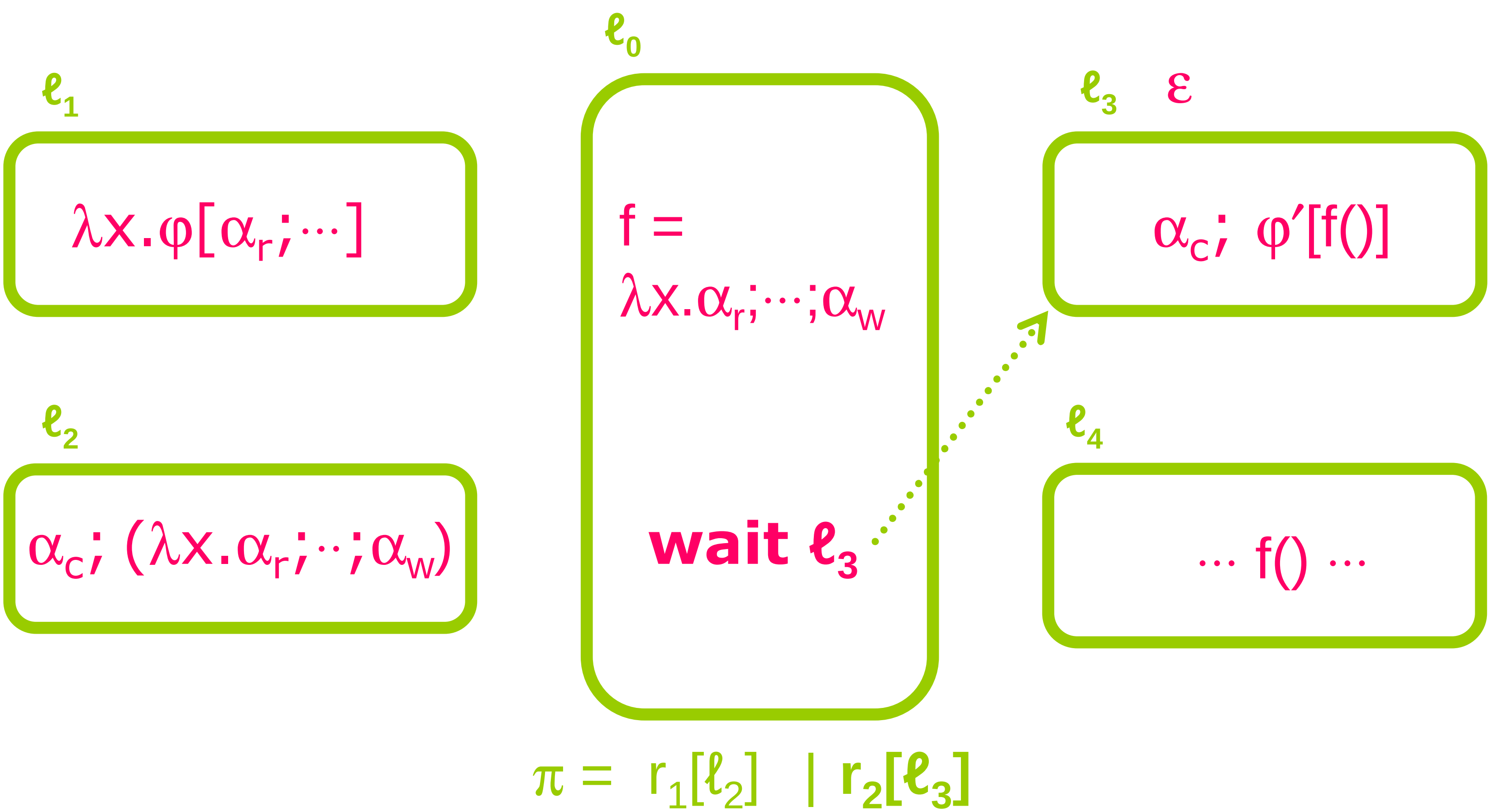
Executing a network of services



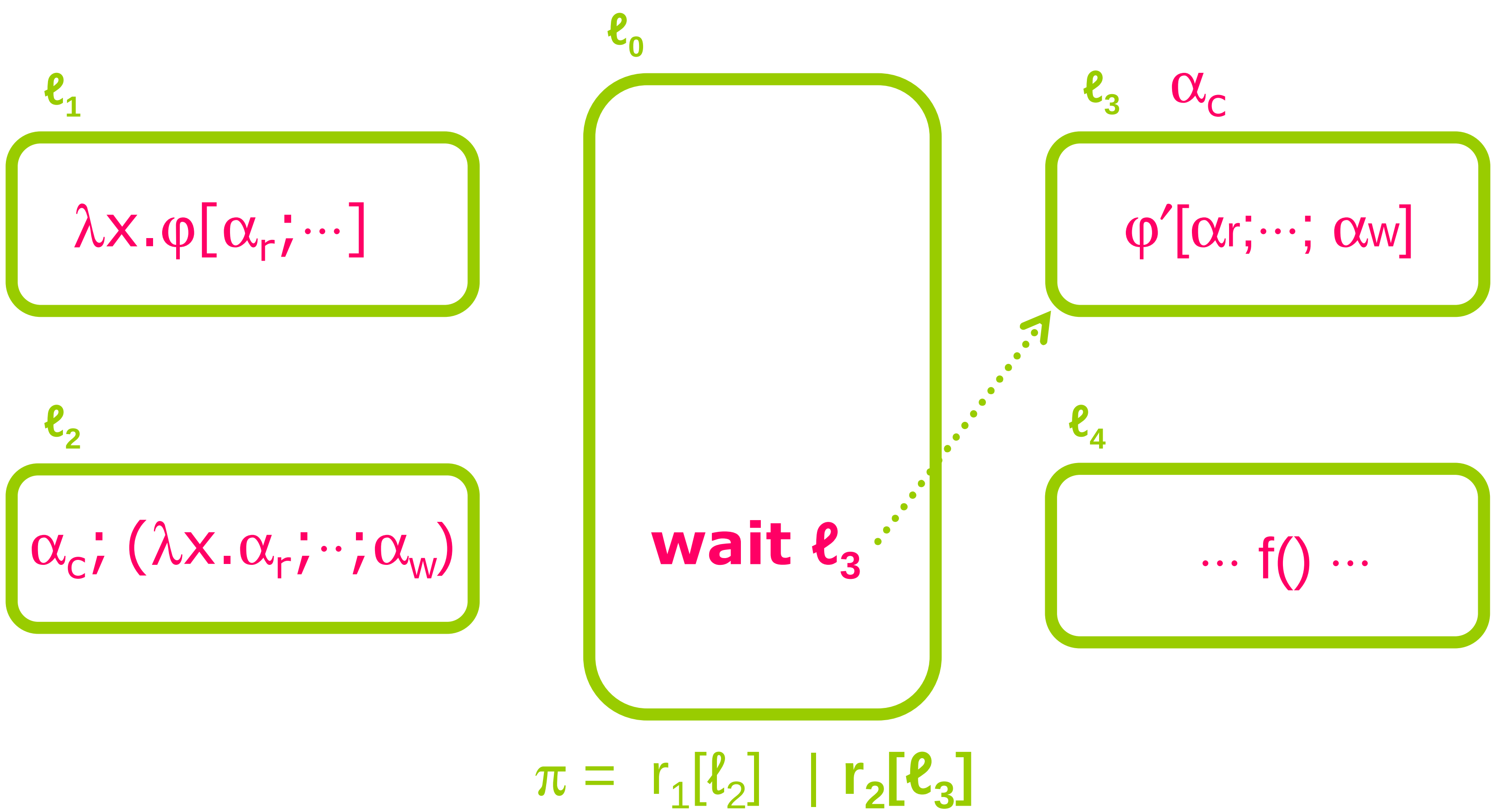
Executing a network of services



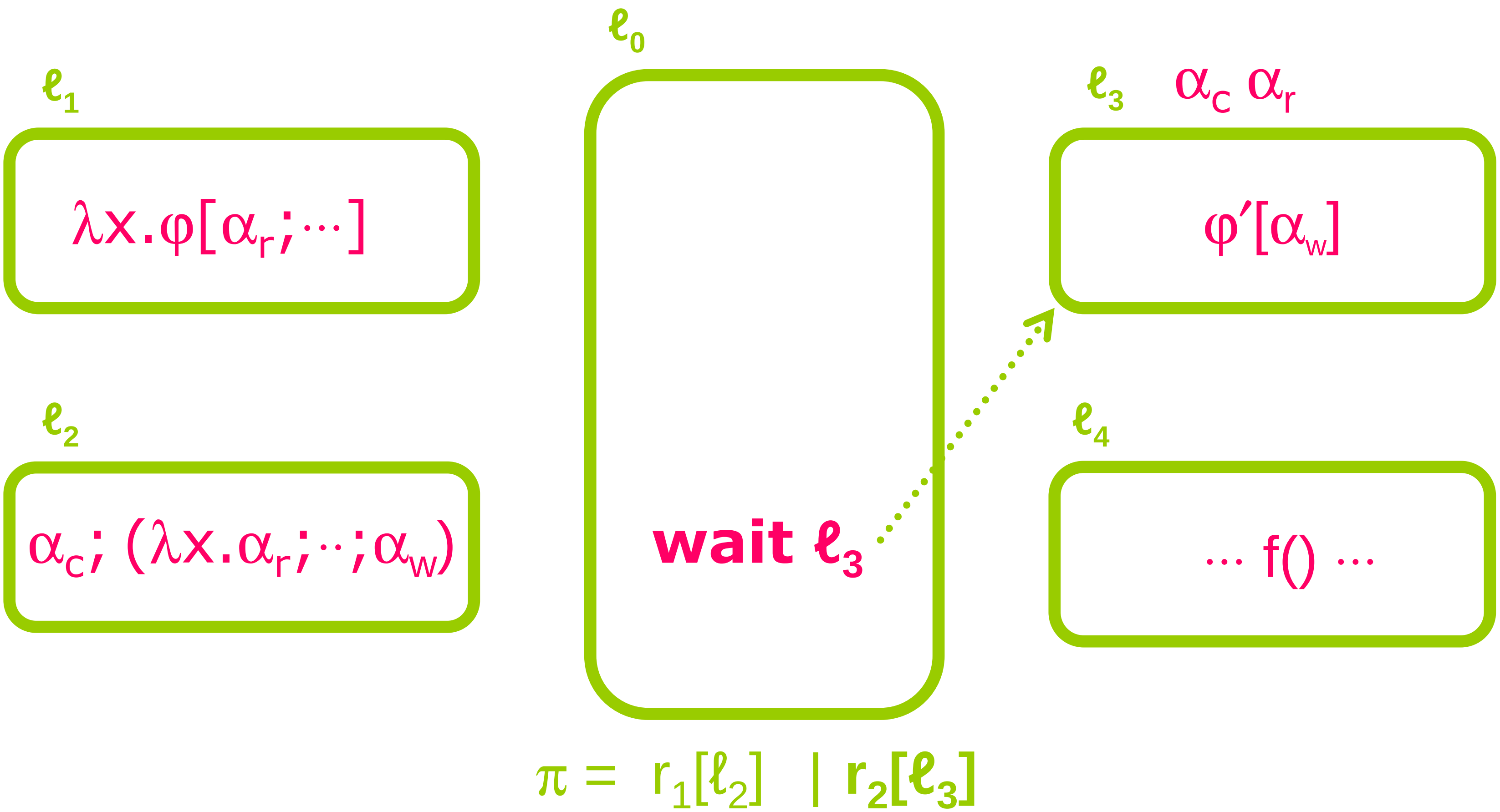
Executing a network of services



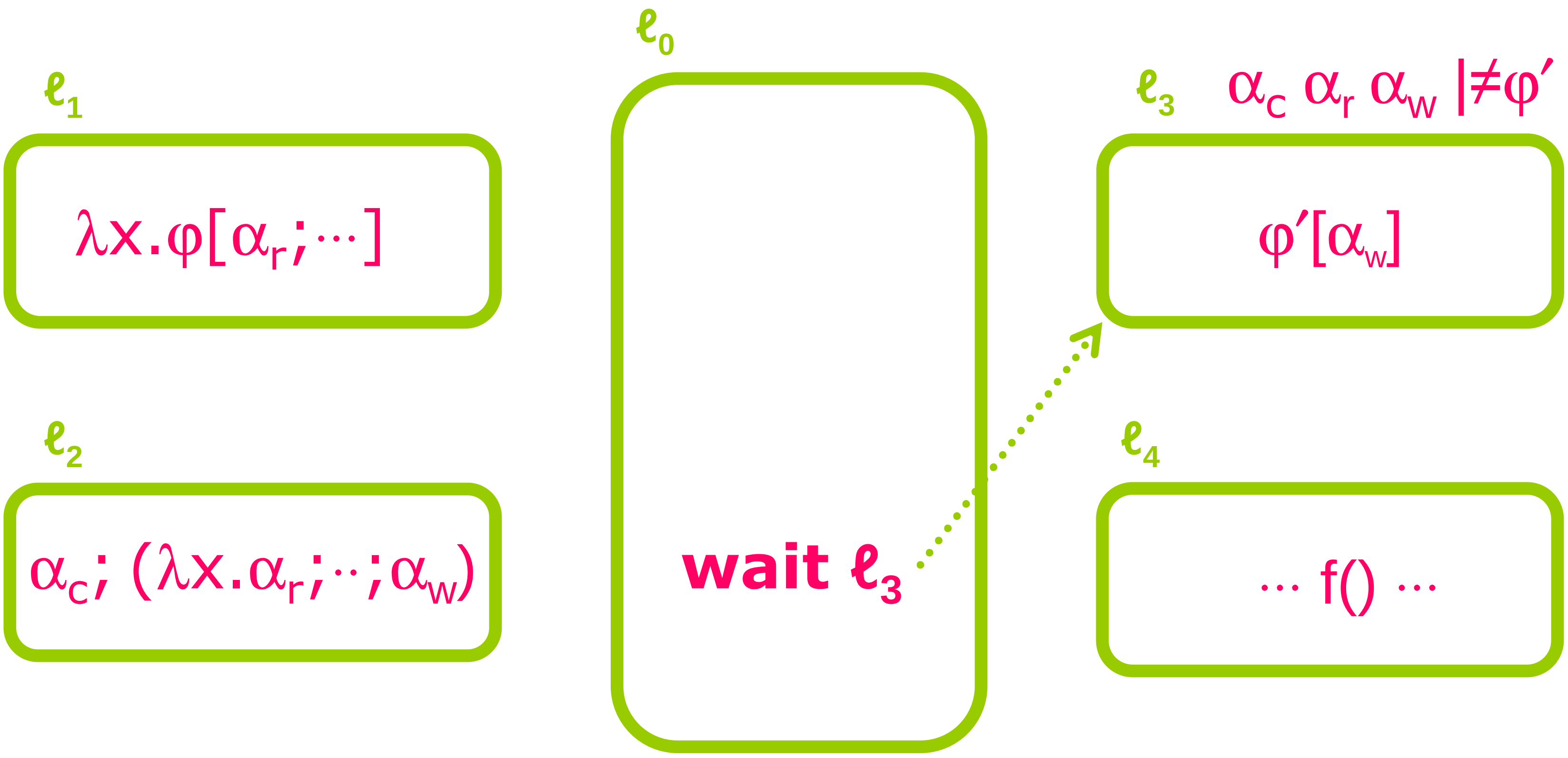
Executing a network of services



Executing a network of services



Executing a network of services



$\pi = r_1[\ell_2] \mid r_2[\ell_3]$ **not viable!**

Semantics of services (1)

[App1]

$$\frac{\eta, e_1 \rightarrow \eta', e_1'}{\eta,\ e_1\ e_2 \rightarrow \eta',\ e_1'\ e_2}$$

[App2]

$$\frac{\eta, e_2 \rightarrow \eta', e_2'}{\eta,\ v\ e_2 \rightarrow \eta',\ v\ e_2'}$$

[AbsApp]

$$\eta,\ (\lambda_z x.e)\ v \rightarrow \eta,\ e\{v/x,\ \lambda_z x.e/z\}$$

[If]

$$\eta,\ \text{if } b \text{ then } e_{true} \text{ else } e_{false} \rightarrow \eta,\ e_{\mathcal{B}(b)}$$

Semantics of services (2)

[Event]

$$\eta, \alpha \rightarrow \eta\alpha, ()$$

[Framing In]

$$\frac{\eta, e \rightarrow \eta', e' \qquad \eta' \models \varphi}{\eta, \varphi[e] \rightarrow \eta', \varphi[e']}$$

[Framing Out]

$$\frac{\eta \models \varphi}{\eta, \varphi[v] \rightarrow \eta, v}$$

Semantics of networks (1)

[Inject]

$$\frac{\eta, e \rightarrow \eta', e'}{\ell: \eta, e \rightarrow_\pi \ell: \eta', e'}$$

{e:τ} omitted

[Par]

$$\frac{N_1 \rightarrow_\pi N_1'}{N_1 \parallel N_2 \rightarrow_\pi N_1' \parallel N_2}$$

Semantics of networks (2)

[Request] $\pi = r[\ell'] \mid \pi'$ -- plan

$$\ell: \eta, \mathrm{req}_r\ v \ \| \ \ell'\{e'\}: \varepsilon, \star \ \to_\pi \ \ell: \eta, \mathrm{wait}\ \ell' \ \| \ \ell'\{e'\}: \varepsilon, e'v$$

[Reply]

$$\ell: \eta, \mathrm{wait}\ \ell' \ \| \ \ell'\{e'\}: \eta', v \ \to_\pi \ \ell: \eta, v \ \| \ \ell'\{e'\}: \varepsilon, \star$$

Other kinds of plans

- **Simple plans** $\pi ::= 0 \mid \pi \,|\, \pi \mid r[\ell]$

  ℓ: req_r || ℓ': {P} $\rightarrow_{r[\ell']}$ ℓ: wait ℓ' || ℓ': P

- **Multi-choice plans** $\pi ::= 0 \mid \pi \,|\, \pi \mid r[\ell_1 \ldots \ell_k]$

  ℓ: req_r || ℓ': {P} $\rightarrow_{r[\ell', \ell'']}$ ℓ: wait ℓ' || ℓ': P

- **Dependent plans** $\pi ::= 0 \mid \pi \,|\, \pi \mid r[\ell.\ \pi]$

  ℓ: $r[\ell'.\pi] \triangleright req_r$ || ℓ': {P} $\rightarrow$ ℓ: $r[\ell'.\pi] \triangleright$ wait ℓ' || ℓ': $\pi \triangleright$ P

- …many others: multi+dependent, regular, dynamic,…

Static semantics

Type & effect system

- **types** carry annotations **H** about service abstract behaviour
- **effects H**, namely ***history expressions***, over-approximate the actual execution histories
- the type & effect inferred for a service depends on its **partial knowledge <** of the network

Types

(pretty standard)

$$\tau ::= \textbf{int} \mid \textbf{bool} \mid \mathbf{1} \mid \cdots \mid \tau \xrightarrow{H} \tau'$$

Effects (history expressions)

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

Semantics of history expressions

$[\![\ \alpha\ ]\!]^{\pi} = (?: \alpha)$

$[\![\ \ell : H\ ]\!]^{\pi} = [\![\ H\ ]\!]^{\pi}\{\ell\ /\ ?\}$

$[\![\{\pi_1 \triangleright H_1 \cdots \pi_k \triangleright H_k\}\ ]\!]^{\pi} = \bigcup_{i=1..k} [\![\ \{\pi_i \triangleright H_i\}\ ]\!]^{\pi}$

$[\![\ \{\pi' \triangleright H\}\ ]\!]^{\pi} = [\![\ H\ ]\!]^{\pi}$ if $\pi' \leq \pi$

plan π' resolves the requests as π

$0 \leq \pi$ $\quad r[\ell] \leq r[\ell] \mid \pi$ $\quad \pi_0 \mid \pi_1 \leq \pi$ if $\pi_0 \leq \pi$ & $\pi_1 \leq \pi$

Semantics of history expressions

$$[\![H \cdot H']\!]^{\pi} = [\![H]\!]^{\pi} \cdot [\![H']\!]^{\pi}$$

$$[\![H + H']\!]^{\pi} = [\![H]\!]^{\pi} + [\![H']\!]^{\pi}$$

$$[\![\mu h.H]\!]^{\pi} = \bigcup_{n>0} f^n(\bot)$$

$$\text{where } f(X) = [\![H]\!]^{\pi}_{\{X / h\}}$$

Example

$$H = \{ r[\ell] \triangleright \{ r'[\ell_1] \triangleright \alpha_1,\ r'[\ell_2] \triangleright \alpha_2 \},\ r[\ell'] \triangleright \beta \}$$

$$\pi = r[\ell] \mid r'[\ell_2]$$

$$\begin{aligned}
[[\ H\ ]]^\pi &= [[\ \{ r[\ell] \triangleright \{ r'[\ell_1] \triangleright \alpha_1,\ r'[\ell_2] \triangleright \alpha_2 \}\}\ ]]^\pi \\
&\quad \cup [[\ \{ r[\ell'] \triangleright \beta \}\ ]]^\pi \\
&= [[\ \{ r'[\ell_1] \triangleright \alpha_1,\ r'[\ell_2] \triangleright \alpha_2 \}\ ]]^\pi \\
&= [[\ \{ r'[\ell_1] \triangleright \alpha_1 \}\ ]]^\pi \cup [[\ \{ r'[\ell_2] \triangleright \alpha_2 \}\ ]]^\pi \\
&= [[\ \alpha_2\ ]]^\pi = (?{:}\ \alpha_2)
\end{aligned}$$

Example

$H = \ell: \{r[\ell_1] \triangleright \ell_1: \alpha_1, r[\ell_2] \triangleright \ell_2: \alpha_2\} \cdot \beta$

$\pi = r[\ell_1]$

$$
\begin{aligned}
[[\ H\ ]]^{\pi} &= [[\{r[\ell_1] \triangleright \ell_1: \alpha_1, r[\ell_2] \triangleright \ell_2: \alpha_2\} \cdot \beta\ ]]^{\pi} \{\ell/?\} \\
&= [[\{r[\ell_1] \triangleright \ell_1: \alpha_1, r[\ell_2] \triangleright \ell_2: \alpha_2\}]]^{\pi} \cdot (\ell: \beta) \\
&= [[\ \ell_1: \alpha_1\ ]]^{\pi} \cdot (\ell: \beta) \\
&= (?: \alpha_1)\ \{\ell_1/?\} \cdot (\ell: \beta) \\
&= (\ell: \beta, \ell_1: \alpha_1)
\end{aligned}
$$

Typing rules (1)

[T-Ev]

$$\Gamma, \alpha \mid\!-_{\ell} \alpha : 1$$

[T-Var]

$$\Gamma, \varepsilon \mid\!-_{\ell} x : \Gamma(x)$$

[T-Loc]

$$\frac{\Gamma, H \mid\!-_{\ell} e : \tau}{\Gamma, \ell : H \mid\!- e : \tau}$$

[T-Wk]

$$\frac{\Gamma, H \mid\!-_{\ell} e : \tau}{\Gamma, H{+}H' \mid\!-_{\ell} e : \tau}$$

Typing rules (2)

[T-Fr]

$$\frac{\Gamma, H \mid\!-_{\ell} e : \tau}{\Gamma, \varphi[H] \mid\!-_{\ell} \varphi[e] : \tau}$$

[T-If]

$$\frac{\Gamma, H \mid\!-_{\ell} e : \tau \qquad \Gamma, H \mid\!-_{\ell} e' : \tau}{\Gamma, H \mid\!-_{\ell} \text{if } b \text{ then } e \text{ else } e' : \tau}$$

Typing rules (3)

[T-Abs]

$$\frac{\Gamma; x{:}\tau; z{:}\tau \xrightarrow{H} \tau', H \mid\!-_{\ell} e : \tau'}{\Gamma, \varepsilon \mid\!-_{\ell} \lambda_z x.e : \tau \xrightarrow{H} \tau'}$$

[T-App]

$$\frac{\Gamma, H \mid\!-_{\ell} e : \tau \xrightarrow{H''} \tau' \qquad \Gamma, H' \mid\!-_{\ell} e' : \tau}{\Gamma, H \cdot H' \cdot H'' \mid\!-_{\ell} e\ e' : \tau'}$$

Typing Example (1)

$$\frac{\alpha \;|\!-_{\ell}\; \alpha{:}1 \qquad\qquad ? \;|\!-_{\ell}\; (\lambda y.zx)\beta{:}1}{z{:}1 \xrightarrow{H} 1, \alpha + ? \;|\!-_{\ell}\; \text{if } b \text{ then } \alpha \text{ else } (\lambda y.zx)\beta{:}1}$$

Typing Example (2)

$$\frac{\alpha \mid\!-_{\ell}\ \alpha{:}1 \qquad \dfrac{\varepsilon \mid\!-_{\ell}\ (\lambda y.zx){:}1 \xrightarrow{H} 1 \qquad \beta \mid\!-_{\ell}\ \beta{:}1}{\varepsilon\cdot\beta\cdot H \mid\!-_{\ell}\ (\lambda y.zx)\beta{:}1}}{z{:}1 \xrightarrow{H} 1, \alpha+\beta\cdot H \mid\!-_{\ell}\ \text{if } b \text{ then } \alpha \text{ else } (\lambda y.zx)\beta{:}1}$$

Typing Example (3)

$$\frac{\dfrac{x{:}1;z{:}1 \xrightarrow{H} 1,\ H \;|\!-_{\ell}\; zx{:}1}{\varepsilon \;|\!-_{\ell}\; (\lambda y.zx){:}1 \xrightarrow{H} 1} \qquad \beta \;|\!-_{\ell}\; \boldsymbol{\beta}{:}1}{\alpha \;|\!-_{\ell}\; \alpha{:}1 \qquad \beta{\cdot}H \;|\!-_{\ell}\; (\lambda y.zx)\boldsymbol{\beta}{:}1}$$

$$\frac{\alpha \;|\!-_{\ell}\; \alpha{:}1 \qquad \beta{\cdot}H \;|\!-_{\ell}\; (\lambda y.zx)\boldsymbol{\beta}{:}1}{z{:}1 \xrightarrow{H} 1, \alpha + \beta{\cdot}H \;|\!-_{\ell}\; \text{if } b \text{ then } \alpha \text{ else } (\lambda y.zx)\boldsymbol{\beta}{:}1}$$

Typing Example (4)

$$\frac{\dfrac{z{:}1 \xrightarrow{H} 1,\ \varepsilon \mid\!-_{\ell}\ z{:}1 \xrightarrow{H} 1 \qquad x{:}1,\ \varepsilon \mid\!-_{\ell}\ x{:}1}{x{:}1;z{:}1 \xrightarrow{H} 1,\ \varepsilon \cdot \varepsilon \cdot H \mid\!-_{\ell}\ zx{:}1}}{\varepsilon \mid\!-_{\ell}\ (\lambda y.zx){:}1 \xrightarrow{H} 1}$$

$$\frac{\alpha \mid\!-_{\ell}\ \alpha{:}1 \qquad \dfrac{\varepsilon \mid\!-_{\ell}\ (\lambda y.zx){:}1 \xrightarrow{H} 1 \qquad \beta \mid\!-_{\ell}\ \beta{:}1}{\beta \cdot H \mid\!-_{\ell}\ (\lambda y.zx)\beta{:}1}}{z{:}1 \xrightarrow{H} 1, \alpha + \beta \cdot H \mid\!-_{\ell}\ \text{if } b \text{ then } \alpha \text{ else } (\lambda y.zx)\beta{:}1}$$

Typing Example

$$\frac{z:1 \xrightarrow{H} 1, \alpha + \beta \cdot H \;\vert\!-_{\ell}\; \text{if } b \text{ then } \alpha \text{ else } (\lambda y.zx)\beta : 1}{\varepsilon \;\vert\!-_{\ell}\; \lambda_z x.\text{if } b \text{ then } \alpha \text{ else } (\lambda y.zx)\beta : \tau \xrightarrow{H} \tau'}$$

To use rule **[T-Abs]** the latent and actual effects must be unified, i.e. $H = \alpha + \beta \cdot H$

A history expression that satisfies the above is:

$$H = \mu h.\, \alpha + \beta \cdot h$$

Typing rules (3)

[T-Req]

$$\tau = \cup \{ \rho +_{r[\ell]} \tau' \mid A \;\&\; B \;\&\; C \}$$

$$A \equiv \varnothing, \varepsilon \vdash_{\ell'} \mathbf{e} : \tau' \qquad B \equiv \rho \approx \tau'$$

$$C \equiv \ell < \ell' \{\mathbf{e} : \tau'\}$$

$$\overline{\Gamma, \varepsilon \vdash_{\ell} \mathbf{req}_r\ \rho : \tau}$$

Typing rules (3)

[T-Req]

certified interface

$$\tau = \bigcup \{\, \rho +_{r[\ell]} \tau' \mid A \;\&\; B \;\&\; C \,\}$$

$$A \equiv \emptyset, \varepsilon \mid\!-_{\ell'} \mathbf{e} : \tau' \qquad B \equiv \rho \approx \tau'$$

$$C \equiv \ell < \ell' \{\mathbf{e} : \tau'\}$$

$$\frac{}{\Gamma, \varepsilon \mid\!-_{\ell} \mathbf{req}_r\ \rho : \tau}$$

Typing rules (3)

[T-Req]

compatible types

$$\tau = \cup \{ \rho +_{r[\ell]} \tau' \mid A \,\&\, B \,\&\, C \}$$

$A \equiv \emptyset, \varepsilon \mid\!-_{\ell'} \mathbf{e} : \tau'$ $\qquad$ $B \equiv \rho \approx \tau'$

$C \equiv \ell < \ell' \{ \mathbf{e} : \tau' \}$

$$\frac{}{\Gamma, \varepsilon \mid\!-_{\ell} \mathrm{req}_r\ \rho : \tau}$$

Typing rules (3)

[T-Req]

$$\tau = \cup \{\, \rho +_{r[\ell]} \tau' \mid A \,\&\, B \,\&\, C \,\}$$

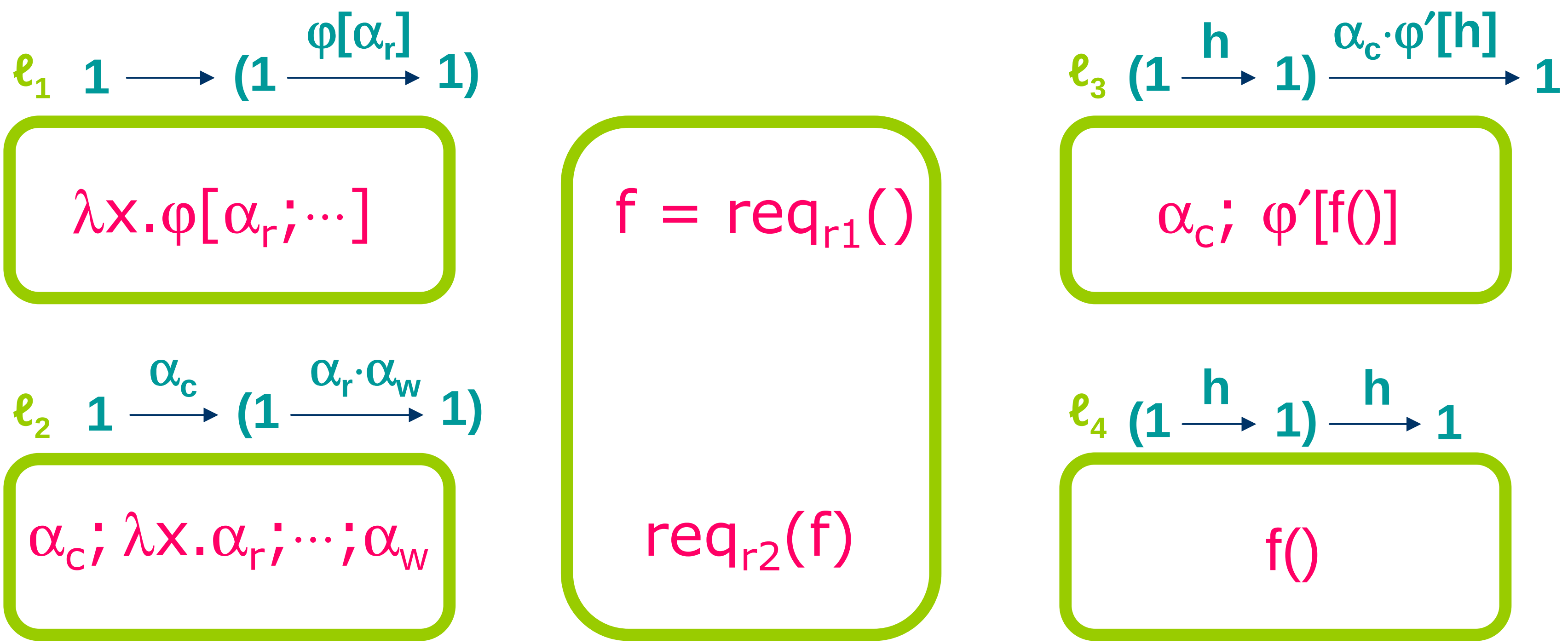
$$A \equiv \emptyset, \varepsilon \vdash_{\ell'} \mathbf{e} : \tau' \qquad B \equiv \rho \approx \tau'$$

$$C \equiv \ell < \ell' \{\mathbf{e} : \tau'\}$$

visibility

$$\overline{\Gamma, \varepsilon \vdash_{\ell} \mathbf{req}_r\ \rho : \tau}$$

Certified published interfaces



Abstracting client behaviour

$\ell_1 \quad 1 \xrightarrow{\varepsilon} (1 \xrightarrow{\varphi[\alpha_r]} 1)$

$\lambda x.\varphi[\alpha_r; \cdots]$

$\ell_2 \quad 1 \xrightarrow{\alpha_c} (1 \xrightarrow{\alpha_r \cdot \alpha_w} 1)$

$\alpha_c; \lambda x.\alpha_r; \cdots; \alpha_w$

$f = \mathbf{req}_{\mathbf{r1}}()$

$req_{r2}(f)$

$\ell_3 \quad (1 \xrightarrow{h} 1) \xrightarrow{\alpha_c \cdot \varphi'[h]} 1$

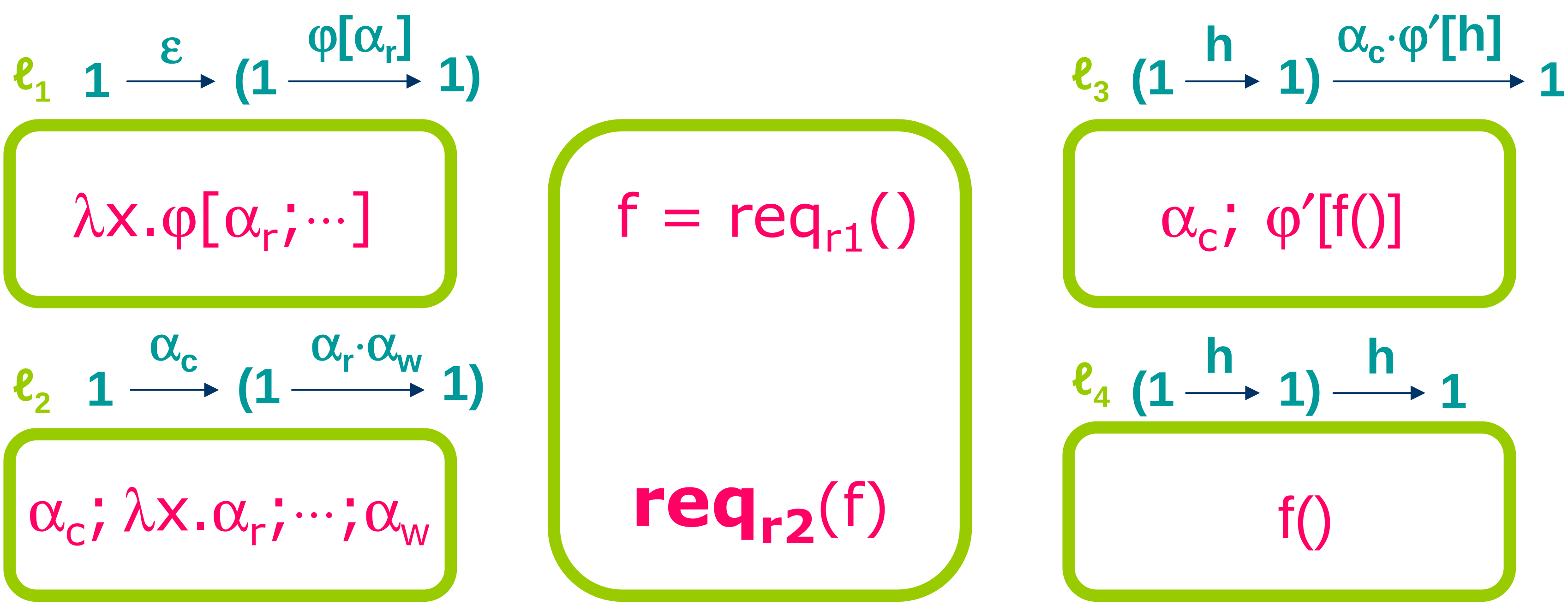
$\alpha_c;\ \varphi'[f()]$

$\ell_4 \quad (1 \xrightarrow{h} 1) \xrightarrow{h} 1$

$f()$

$\{\ r_1[\ell_1] \triangleright \ell_1: \varepsilon,\ r_1[\ell_2] \triangleright \ell_2: \alpha_c\ \} \cdot$

Abstracting client behaviour



$\{\ r_2[\ell_3] \triangleright \ell_3\!: \alpha_c \cdot\ \varphi'[\{\ r_1[\ell_1] \triangleright \varphi[\alpha_r],\ r_1[\ell_2] \triangleright \alpha_r \cdot \alpha_w\ \}],$

$r_2[\ell_4] \triangleright \ell_4\!: \{\ r_1[\ell_1] \triangleright \varphi[\alpha_r],\ r_1[\ell_2] \triangleright \alpha_r \cdot \alpha_w\ \}\}$

Summing up …

Calculus: **operational semantics** and

type & effect system

- **effects** are history expressions, and over-approximate the actual execution histories
- **planned selections** therein hinder information about which plans to choose for secure compositions

What's next: the road to viable plans

- **linearization:** extracting plans and their "pure" effects by unscrambling the structure of history expressions
- **validity:** defining when an effect denotes histories that *"never go wrong"*
- **model checking:** valid plans are viable
  - transform **history expression** into BPAs
  - transform **policies** into FSAs
- **orchestrator:** uses viable plans to drive safe service composition

Which are the viable plans ?

$\{ r_1[\ell_1] \triangleright \ell_1 : \varepsilon,\ r_1[\ell_2] \triangleright \ell_2 : \alpha_c] \} \cdot$

$\{ r_2[\ell_3] \triangleright \ell_3 : \alpha_c \cdot \varphi'[\{ r_1[\ell_1] \triangleright \varphi[\alpha_r],\ r_1[\ell_2] \triangleright \alpha_r \cdot \alpha_w \}],$

$r_2[\ell_4] \triangleright \ell_4 : \{ r_1[\ell_1] \triangleright \varphi[\alpha_r],\ r_1[\ell_2] \triangleright \alpha_r \cdot \alpha_w \}\}$

Difficult to tell: the planned selections are nested!

$\{ r_1[\ell_1] \mid r_2[\ell_3] \triangleright \ell_1 : \varepsilon,\ \ell_3 : \alpha_c \cdot \varphi'[\varphi[\alpha_r]],$ viable

$\{ r_1[\ell_2] \mid r_2[\ell_4] \triangleright \ell_2 : \alpha_c,\ \ell_4 : \alpha_r \cdot \alpha_w,$ viable

$\{ r_1[\ell_1] \mid r_2[\ell_4] \triangleright \ell_1 : \varepsilon,\ \ell_4 : \varphi[\alpha_r],$ not viable

$\{ r_1[\ell_2] \mid r_2[\ell_3] \triangleright \ell_2 : \alpha_c,\ \ell_3 : \alpha_c \cdot \varphi'[\alpha_r \cdot \alpha_w]\}$ not viable

Linearization

- transform H into a *semantically equivalent* $H' \equiv H$ such that H′ is in linear form, i.e.:

$$H' = \{\pi_1 \triangleright H_1 \cdots \pi_k \triangleright H_k\}$$

and the H_i have no planned selections.

- defined through oriented equations $\equiv$ that groups $r[\ell]$ in topmost position

Linearization

$$H \equiv \{0 \triangleright H\}$$

$$\{\pi_i \triangleright H_i\}_i \cdot \{\pi'_j \triangleright H'_j\}_j \equiv \{\pi_i \mid \pi'_j \triangleright H_i \cdot H'_j\}_{i,j}$$

$$\{\pi_i \triangleright H_i\}_i + \{\pi'_j \triangleright H'_j\}_j \equiv \{\pi_i \mid \pi_j \triangleright H_i + H'_j\}_{i,j}$$

$$\varphi[\,\{\pi_i \triangleright H_i\}_i\,] \equiv \{\pi_i \triangleright \varphi[H_i]\,\}_i$$

$$\mu h.\{\pi_i \triangleright H_i\}_i \equiv \{\pi_i \triangleright \mu h.\, H_i\,\}_i$$

$$\{\pi_i \triangleright \{\pi'_{i,j} \triangleright H_{i,j}\}_j\,\}_i \equiv \{\pi_i \mid \pi'_{i,j} \triangleright H_{i,j}\,\}_{i,j}$$

Example

H

$\varphi[\ \lambda_z x.\ \mathbf{req}_r\ \rho;\ z\ x\ ]$

ℓ_1

α

ℓ_2

β

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

Example

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

Simple vs multi-choice plans

With simple plans:

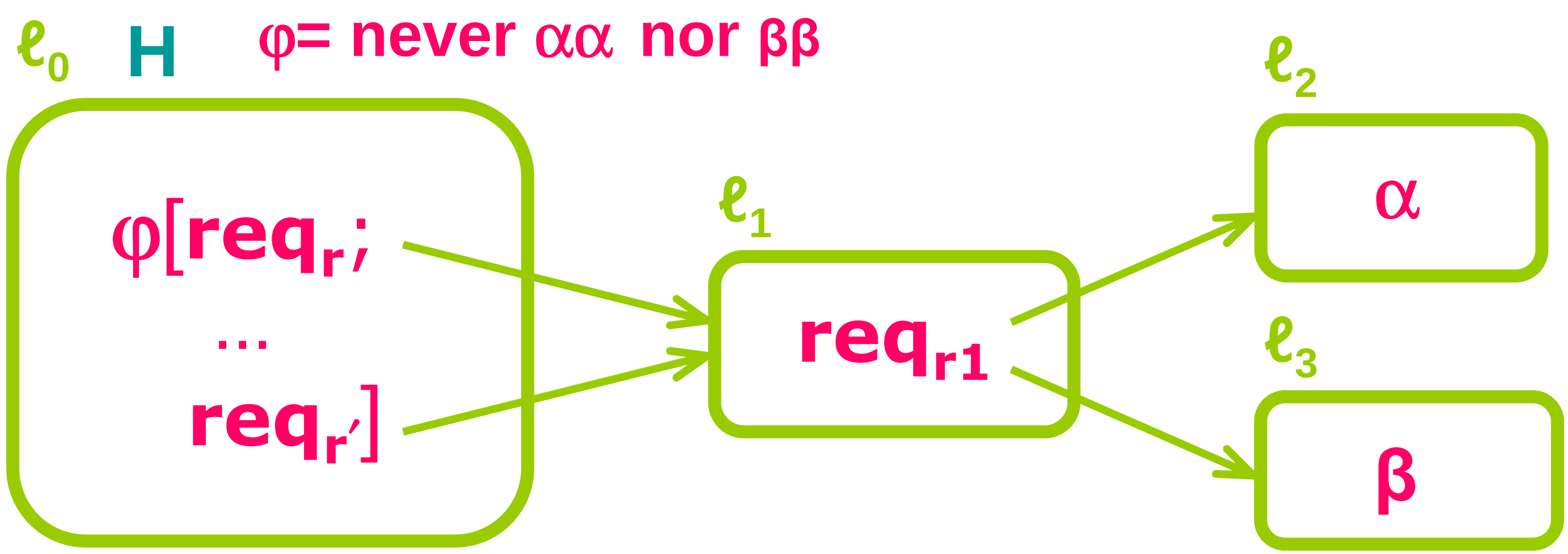
$$H \equiv \{\, r[\ell_1] \triangleright \varphi[\,\mu h.\ \alpha \cdot h],\ r[\ell_2] \triangleright \varphi[\,\mu h.\ \beta \cdot h]\,\}$$

With multi-choice plans:

$$H \equiv \{\, r[\ell_1] \triangleright \varphi[\,\mu h.\ \alpha \cdot h],\ r[\ell_2] \triangleright \varphi[\,\mu h.\ \beta \cdot h],$$
$$r[\ell_1,\ell_2] \triangleright \varphi[\,\mu h.\ (\alpha + \beta) \cdot h]\}$$

Plan $r[\ell_1,\ell_2]$ useful when ℓ_1 or ℓ_2 unavailable

Example: bottleneck service



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

Simple vs dependent plans

With simple plans:

$H \equiv \{\ r[\ell_1] \mid r_1[\ell_2] \mid r'[\ell_1] \triangleright \varphi[\alpha \cdot \alpha],$	not viable
$r[\ell_1] \mid r_1[\ell_3] \mid r'[\ell_1] \triangleright \varphi[\beta \cdot \beta]\ \}$	not viable

With dependent plans:

$H \equiv \{\ r[\ell_1.\ r_1[\ell_2]] \mid r'[\ell_1.\ r_1[\ell_2]] \triangleright \varphi[\alpha \cdot \alpha],$	not viable
$r[\ell_1.\ r_1[\ell_2]] \mid r'[\ell_1.\ r_1[\ell_3]] \triangleright \varphi[\alpha \cdot \beta],$	viable
$r[\ell_1.\ r_1[\ell_3]] \mid r'[\ell_1.\ r_1[\ell_2]] \triangleright \varphi[\beta \cdot \alpha],$	viable
$r[\ell_1.\ r_1[\ell_3]] \mid r'[\ell_1.\ r_1[\ell_3]] \triangleright \varphi[\beta \cdot \beta]\ \}$	not viable

Validity

- histories are enriched with $[_\varphi$ and $]_\varphi$ to point out the scope of policies.
- a history is **valid** when all the policies are respected, within their scopes
  - ex: $\alpha_w \; \alpha_r \; [_\varphi \; \alpha_w \;]_\varphi$ not valid (write after read)
  - ex: $\alpha_w \; [_\varphi \; \alpha_r \;]_\varphi \; \alpha_w$ valid (write outside scope of φ)
- a history expression **H** is π-valid when all the histories in **[[H]]**$^\pi$ are valid.

Validity, formally

- **Safe sets**:
  - $S(\varepsilon) = 0$ $\quad S(\eta\ \alpha) = S(\eta)$
  - $S(\eta_0\ [_\varphi\ \eta_1\ ]_\varphi) = S(\eta_0\ \eta_1) \cup \varphi[\ flat(\eta_0)\ flat\ pref(\eta_1)\ ]$
- **Example:**

$$S([_\varphi\ \alpha\ [_\psi\ \beta\ ]_\psi\ \gamma\ ]_\varphi) = S(\alpha\ [_\psi\ \beta\ ]_\psi\ \gamma\ ) \cup \varphi[\{\varepsilon, \alpha, \alpha\beta, \alpha\beta\gamma\}]$$

$$= \Psi[\{\alpha, \alpha\beta\}], \varphi[\{\varepsilon, \alpha, \alpha\beta, \alpha\beta\gamma\}]$$

- η **is *valid* if, for each** $\varphi[\{\eta_1,\ldots,\eta_k\}]$ **in** $S(\eta)$**:**

$$\eta_i \models \varphi \quad \text{for } 1 \leq i \leq k$$

Verifying validity

Model checking: valid plans are viable (drive executions that *never go wrong*)

- transform linearized **history expression** into **BPAs** (Basic Process Algebras)
- transform **policies** into **scoped policies** (in the form of Finite State Automata)

From history expressions to BPAs

Example

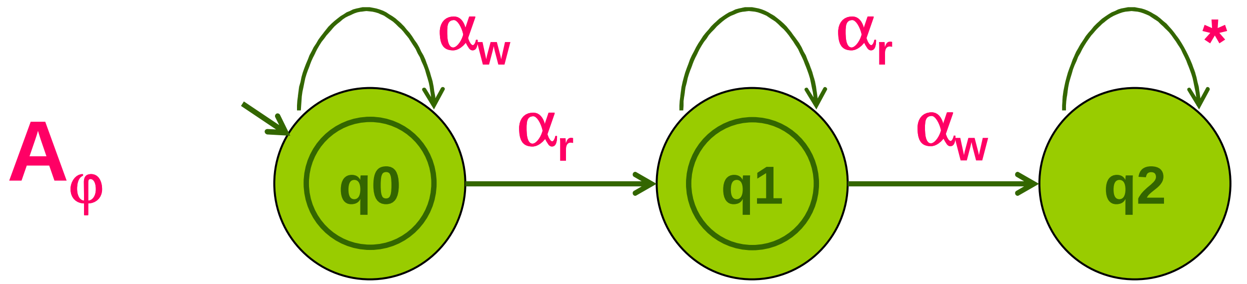
$$H = \beta \cdot (\mu h.\ \alpha + h \cdot h + \varphi[h])$$

$$BPA(H) = \beta \cdot X,$$
$$\{ X = \alpha + X \cdot X + [_\varphi \cdot X \cdot]_\varphi \}$$

Theorem: [[H]] = [[BPA(H)]]

From policies to scoped policies

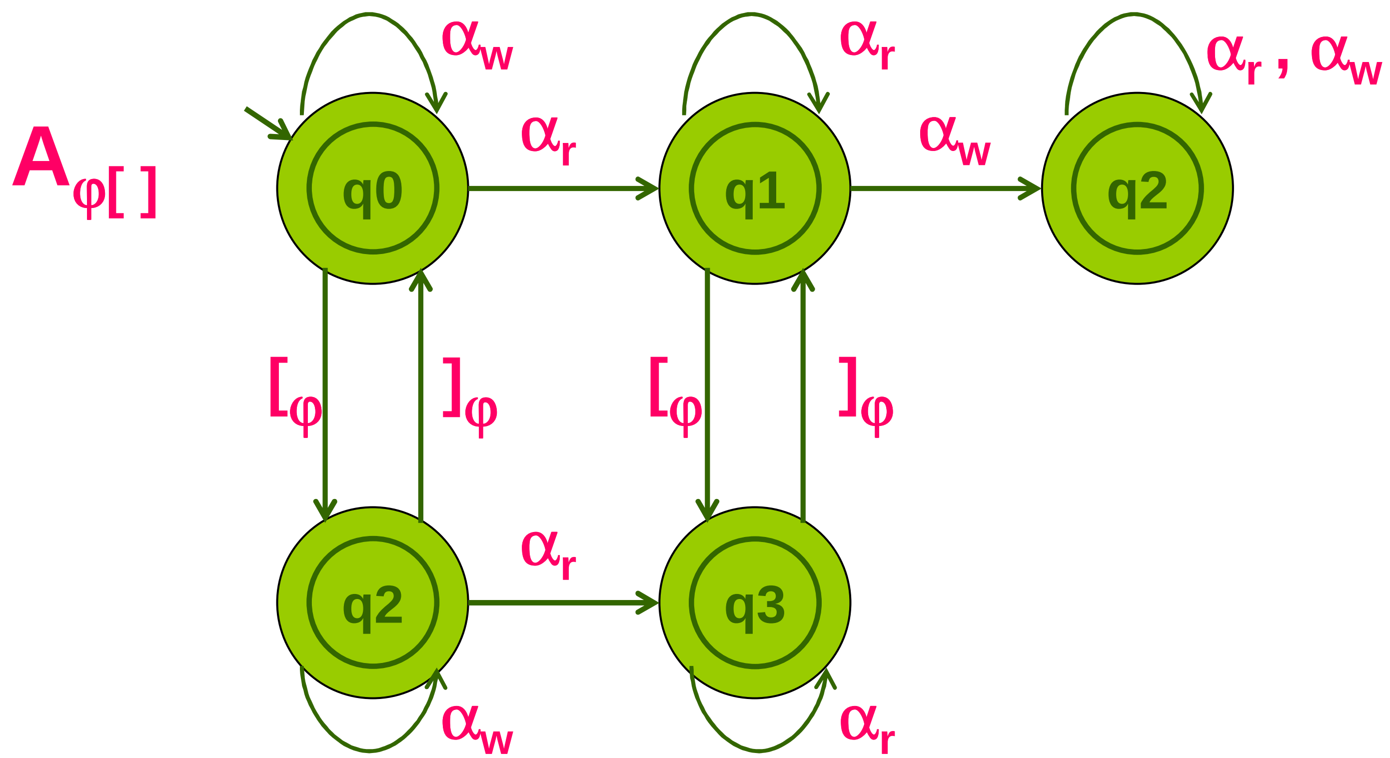
Example



Chinese Wall policy: no write after read

From policies to scoped policies

Example



From policies to scoped policies

Theorem:

η valid	**iff**
	η accepted by $A_{\varphi[\ ]}$ for all φ occurring in η

η w/o "redundant" framings $\varphi[\ldots\varphi[\ldots]\ldots] = \varphi[\ldots\ \ldots\ \ldots]$

Model- checking BPAs with FSAs

Theorem:

H valid iff

$$[[\ \mathrm{BPA(H)}\ ]] \models \bigwedge_{\varphi \text{ in } H} A_{\varphi[\,]}$$

Main result

Network N = ℓ_1\{ e_1: τ_1\} || … || ℓ_k\{ e_k: τ_k\}

$$\emptyset, H_i \vdash e_i : \tau_i \qquad \text{for } 1 \leq i \leq k$$

If H_i is π-valid then π is viable for e_i

Summing up …

- **hypothesis:** client with history expression **H**
- **linearization:** transform **H** into a semantically equivalent **H′** in linear form, i.e.:

$$\mathbf{H}' = \{\pi_1 \triangleright \mathbf{H}_1 \cdots \pi_k \triangleright \mathbf{H}_k\}$$

  and the $\mathbf{H}_i$ have no planned selections.
- **verification:** model-check the $\mathbf{H}_i$ for validity
- **theorem:** if $\mathbf{H}_i$ is valid, then π_i is viable

Conclusions

A linguistic framework for

secure service composition

- safety framings, policies, req-by-contract
- type & effect system
- verification of effects, to extract viable plans

The orchestrator securely composes and runs service-based applications

Other issues considered

- **instrumentation:** how to compile local policies into local checks, in case that some policy may fail
- **resource creation:** how to create fresh resources
- **liveness:** how to deal with properties of the form “something good will eventually happen”
- **multi-choice and dependent plans**

Future work

- other kinds of plans (e.g. dynamic)
- other kinds of effects (e.g. sessions)
- safety framings and security protocols
- safety framings for information flow
- incremental analysis, when new services can be discovered at run-time
- trust relations between services
- spatial types and logics

References

- M. Bartoletti, P. Degano, G.L. Ferrari. Types and Effects for Secure Service Orchestration. *CSFW'06.*
- M. Bartoletti, P. Degano, G.L. Ferrari. Plans for service composition. *WITS'06.*
- M. Bartoletti, P. Degano, G.L. Ferrari. Enforcing Secure Service Composition. *CSFW'05.*
- M. Bartoletti, P. Degano, G.L. Ferrari. Checking risky events is enough for local policies. *ICTCS'05.*

www.di.unipi.it/~bartolet/pubs