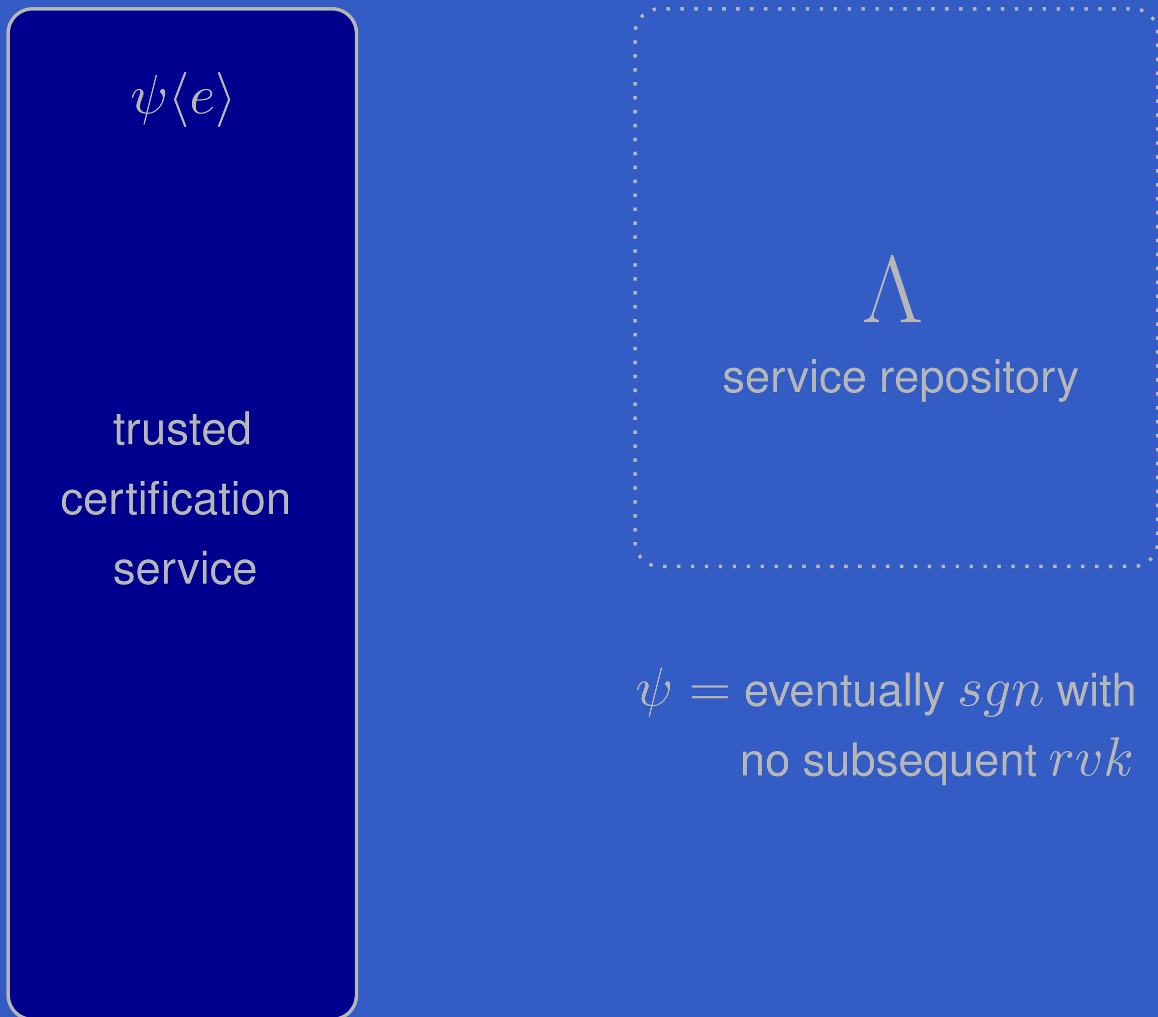
Enforcing Secure Service Composition

Massimo Bartoletti Pierpaolo Degano Gian Luigi Ferrari

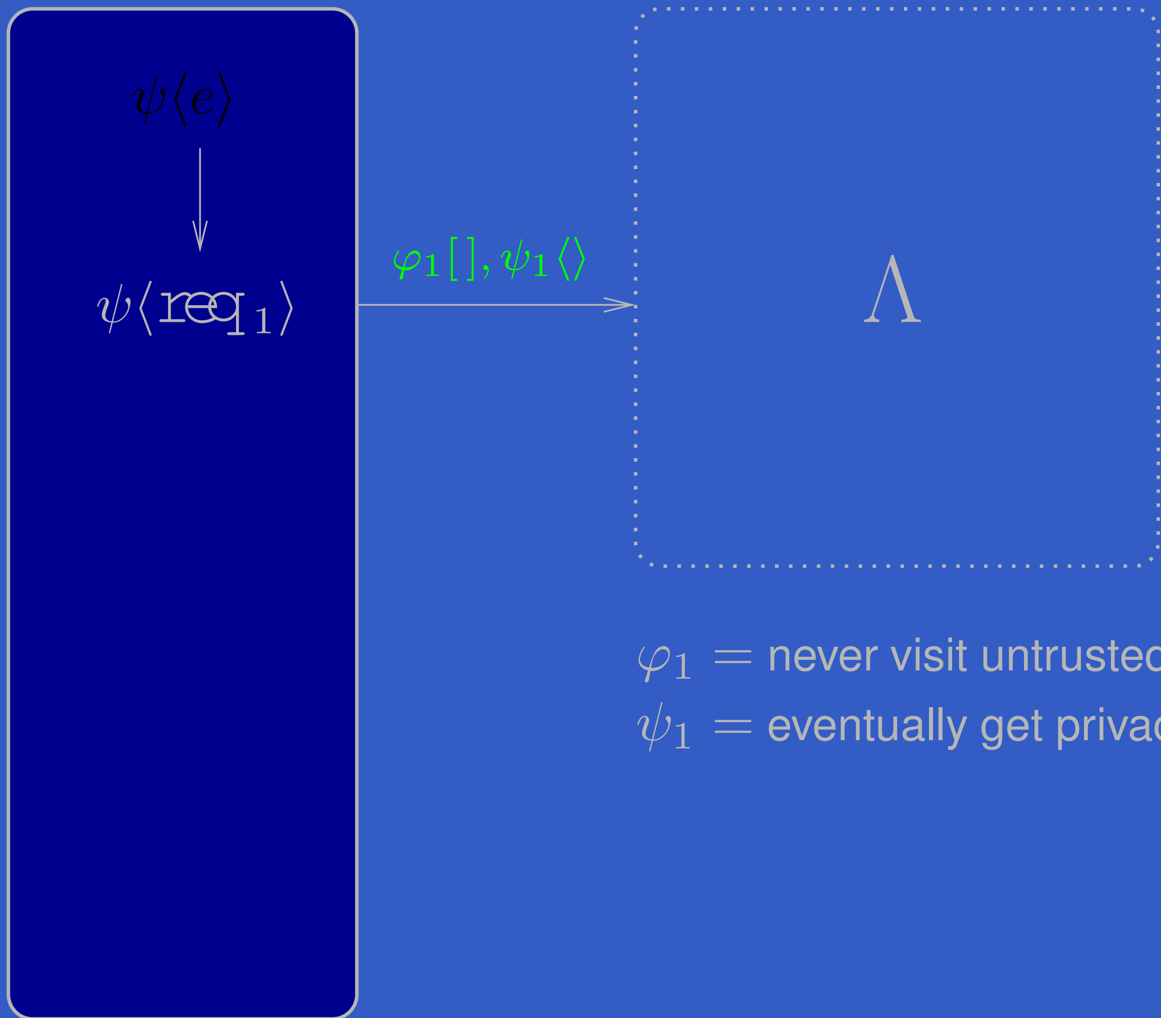
Dipartimento di Informatica, Università di Pisa

Example: contract signing



$\psi =$ eventually sgn with
no subsequent rvk

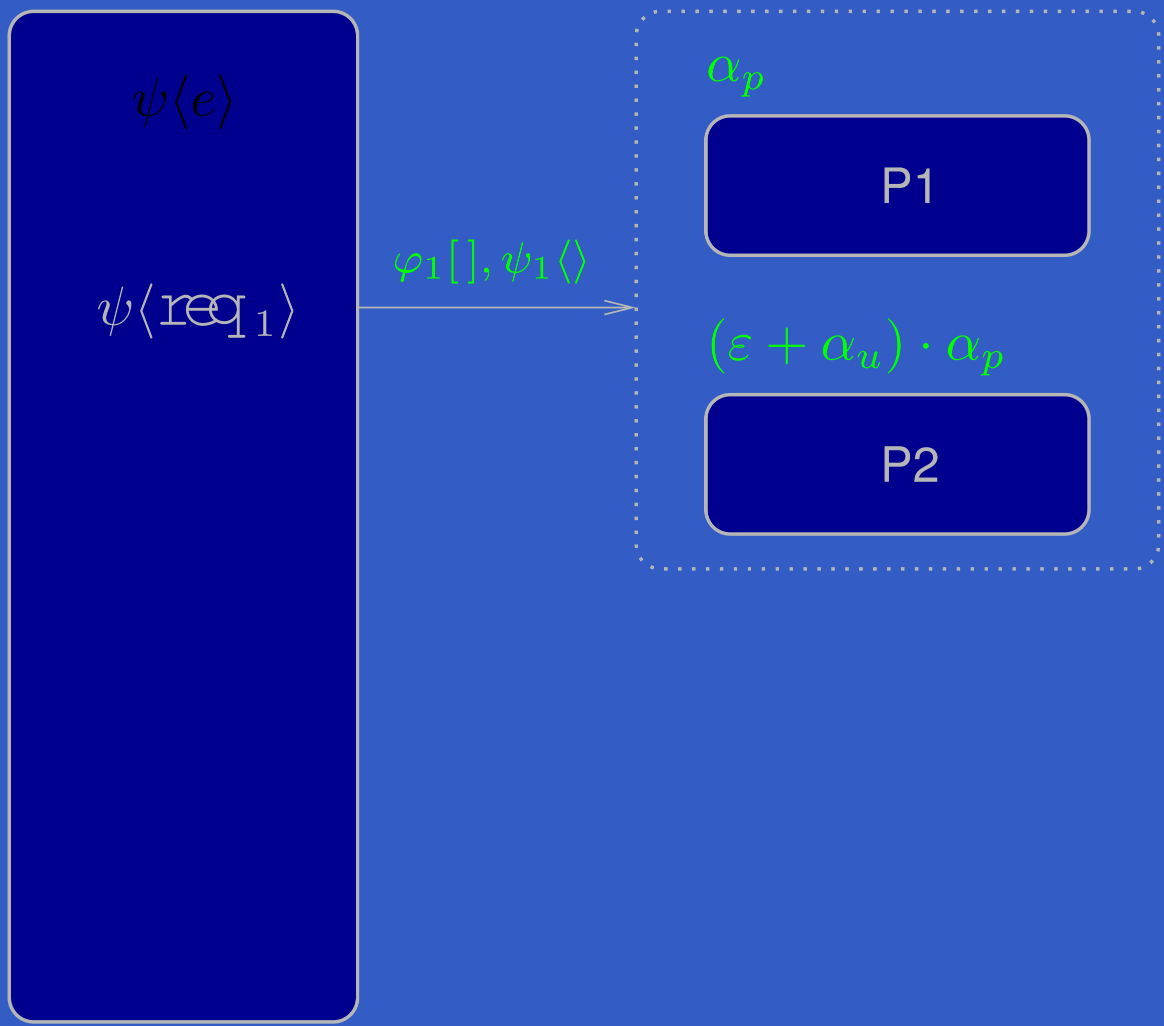
Example: contract signing



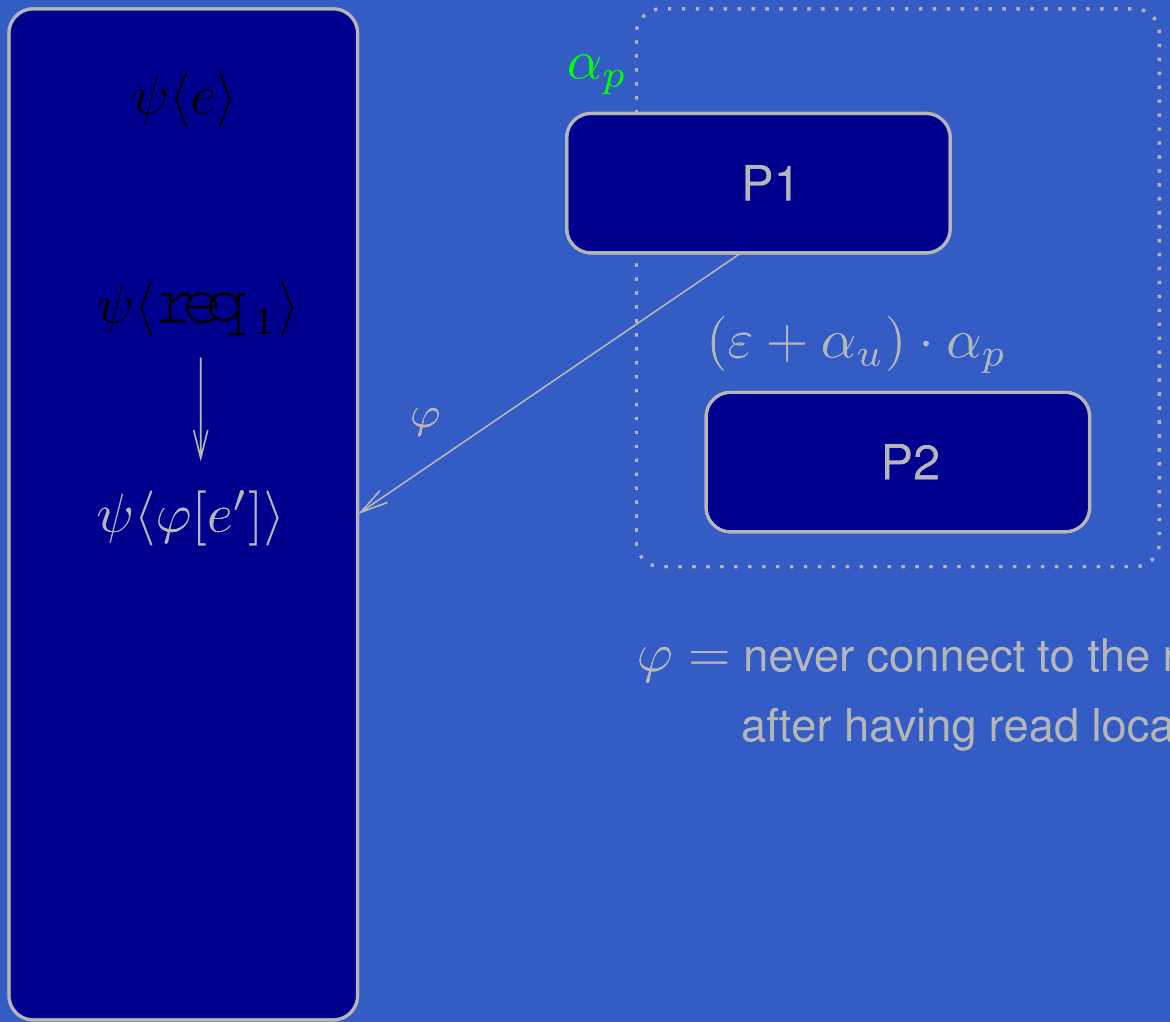
φ_1 = never visit untrusted sites (α_u)

ψ_1 = eventually get privacy policy (α_p)

Example: contract signing

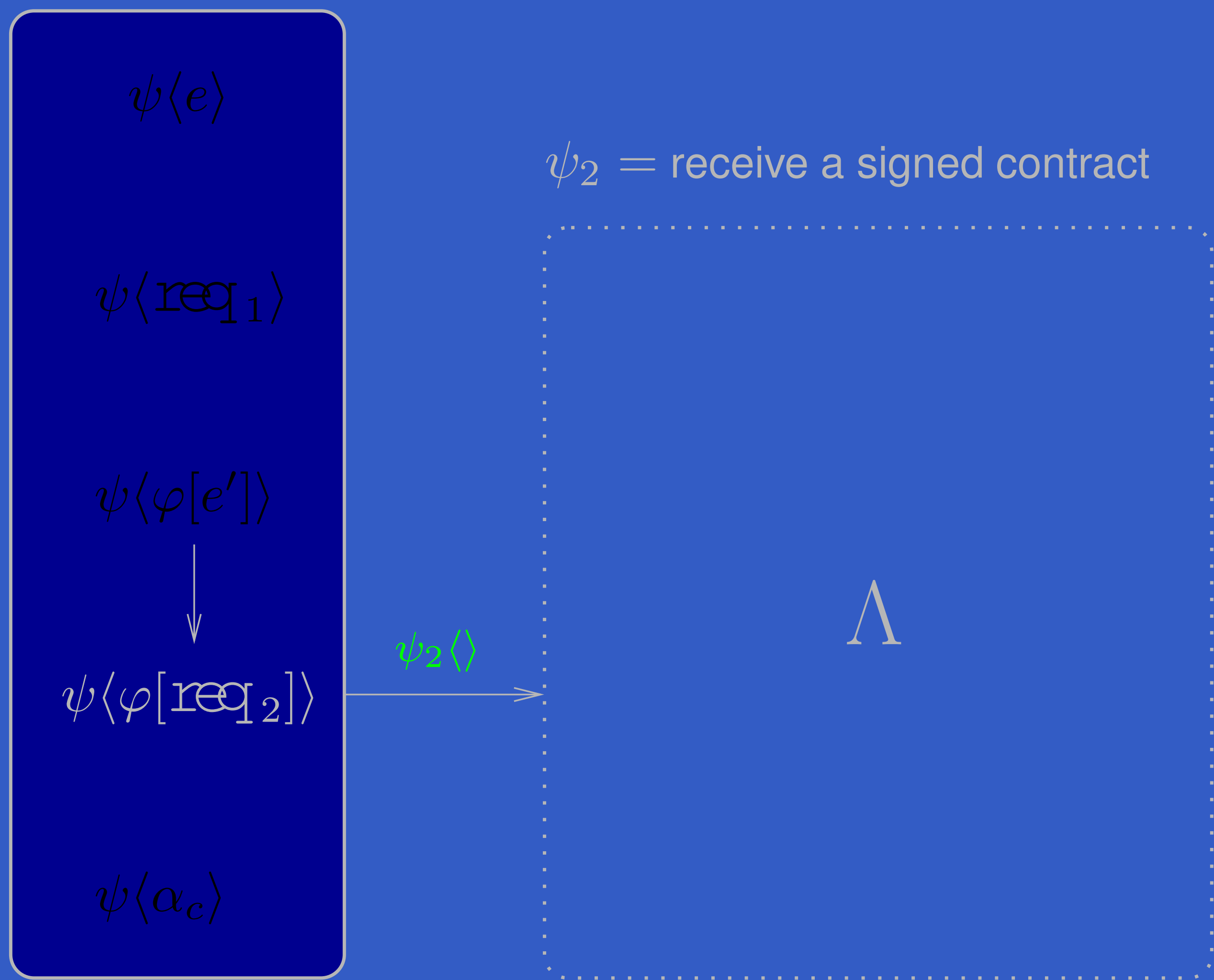


Example: contract signing

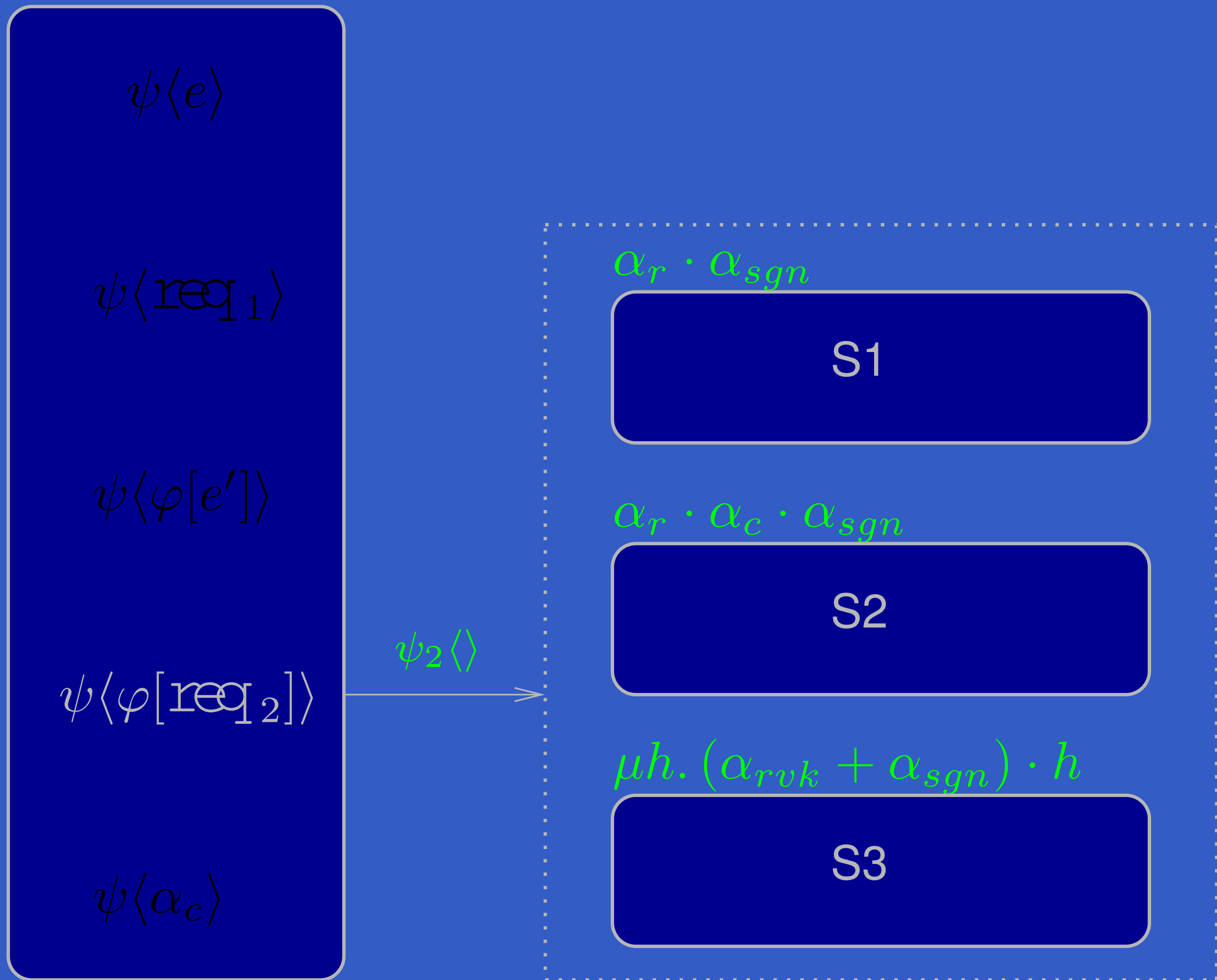


$\varphi =$ never connect to the network (α_c)
after having read local files (α_r)

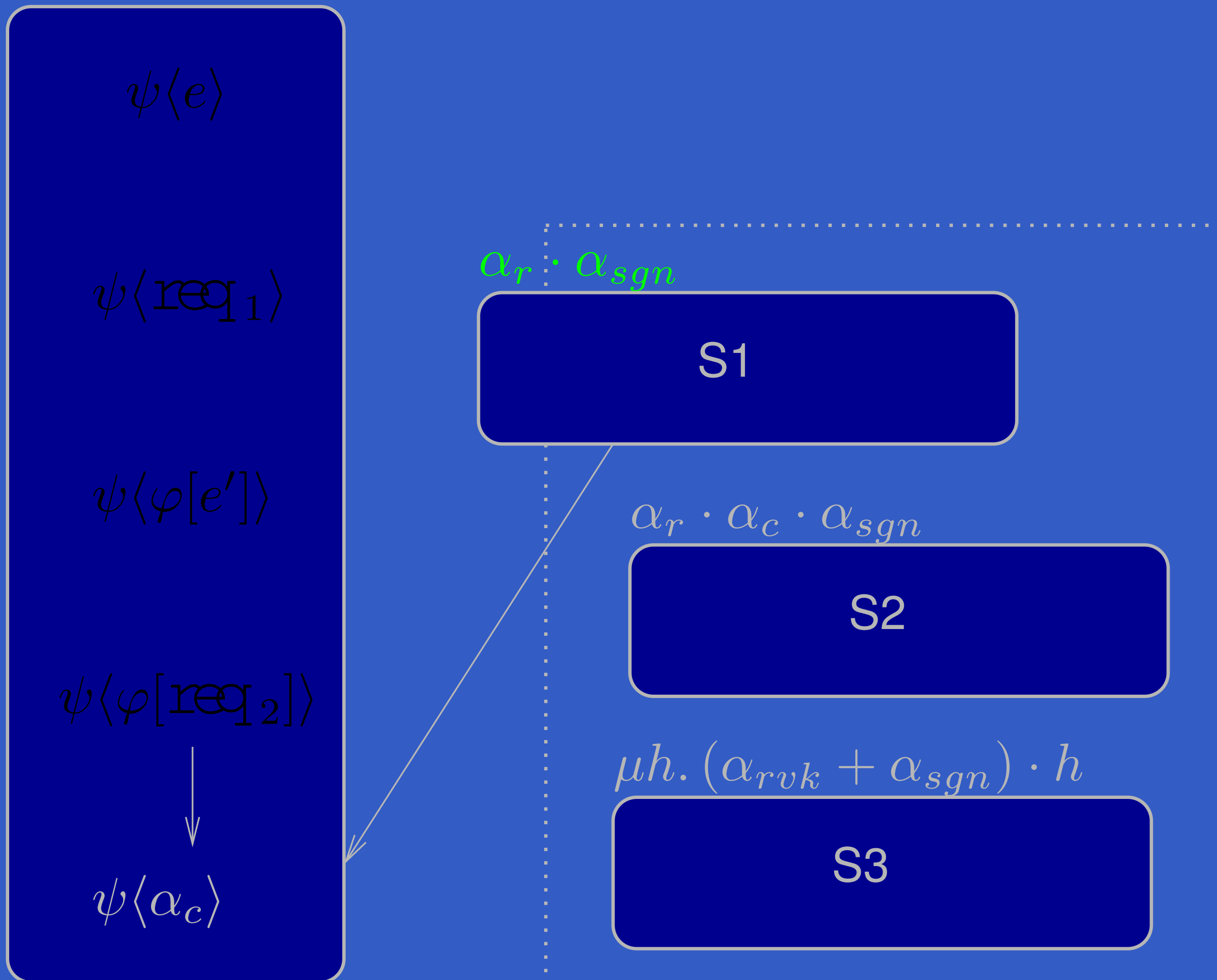
Example: contract signing



Example: contract signing



Example: contract signing



Overview

- calculus for secure service composition
  - local safety/liveness policies
  - call-by-contract service invocation
  - subsumes history-based access control
- dynamic semantics (liveness?)
- static semantics: type & effect system
  - approx. runtime behaviour (history expr.)
- static verification via model checking
  - selects services matching contracts

A calculus for service composition

- λ + histories + local policies + call-by-contract
- a history η is a sequence of events α
- policies φ, ψ are regular properties of η
- each computation step in a safety framing $\varphi[e]$ must respect φ
- some step in a liveness framing $\psi\langle e \rangle$ must eventually satisfy ψ
- a request $\mathtt{req}\ \tau \xrightarrow{\varphi[\,],\psi\langle\rangle} \tau'$ selects the services respecting (always) φ and (eventually) ψ

Why local policies ?

Problem: securization of code upon receipt

- an untrusted program e is received
- I want the execution of e to obey a policy φ
- where to insert the checks in e ?
- one issue: e could invoke unknown code
- solution: dynamic sandboxing $\varphi[e]$
- each execution step of e is subject to φ
- the scope of φ is left when e terminates

Enforcing Principle of least privilege

"Programs should be granted the minimum set of rights that are needed to accomplish their tasks."

- an expression must always obey *all* the active policies (no policy can be overridden)
- policies can always inspect the *whole* past history (no event can be hidden)
- so how to implement "privileged calls" / discard the past ?
- answer: policies must explicitly allow for it!

Extracting History Expressions

- $e_1 = \texttt{if}\ b\ \texttt{then}\ \alpha\ \texttt{else}\ \alpha'$
  $H_1 = \alpha + \alpha' \quad \alpha, \alpha'$

Extracting History Expressions

- $e_1 = \texttt{if}\ b\ \texttt{then}\ \alpha\ \texttt{else}\ \alpha'$
  $H_1 = \alpha + \alpha' \quad \alpha, \alpha'$
- $e_2 = (\lambda x.\, \alpha') : unit \xrightarrow{\alpha'} unit$
  $H_2 = \varepsilon \quad \emptyset$

Extracting History Expressions

- $e_1 = \texttt{if}\ b\ \texttt{then}\ \alpha\ \texttt{else}\ \alpha'$
  $H_1 = \alpha + \alpha' \quad \alpha, \alpha'$
- $e_2 = (\lambda x.\, \alpha') : unit \xrightarrow{\alpha'} unit$
  $H_2 = \varepsilon \quad \emptyset$
- $e_3 = (\lambda x.\, \alpha')\alpha$
  $H_3 = \alpha \cdot \alpha' \quad \alpha\alpha'$

Extracting History Expressions

- $e_1 = \texttt{if}\ b\ \texttt{then}\ \alpha\ \texttt{else}\ \alpha'$
  $H_1 = \alpha + \alpha' \quad \alpha, \alpha'$
- $e_2 = (\lambda x.\, \alpha') : unit \xrightarrow{\alpha'} unit$
  $H_2 = \varepsilon \quad \emptyset$
- $e_3 = (\lambda x.\, \alpha')\alpha$
  $H_3 = \alpha \cdot \alpha' \quad \alpha\alpha'$
- $e_4 = (\lambda y.\, \varphi[\alpha; y*])(\lambda x.\, \varphi'[\alpha'])$
  $H_4 = \varphi[\alpha \cdot \varphi'[\alpha']] \quad [_\varphi \alpha [_{\varphi'} \alpha']_{\varphi'}]_\varphi$

Extracting History Expressions

- $e_1 = \mathsf{if}\ b\ \mathsf{then}\ \alpha\ \mathsf{else}\ \alpha'$
  $H_1 = \alpha + \alpha' \quad \alpha, \alpha'$
- $e_2 = (\lambda x.\, \alpha') : unit \xrightarrow{\alpha'} unit$
  $H_2 = \varepsilon \quad \emptyset$
- $e_3 = (\lambda x.\, \alpha')\alpha$
  $H_3 = \alpha \cdot \alpha' \quad \alpha\alpha'$
- $e_4 = (\lambda y.\, \varphi[\alpha; y*])(\lambda x.\, \varphi'[\alpha'])$
  $H_4 = \varphi[\alpha \cdot \varphi'[\alpha']] \quad [_\varphi \alpha [_{\varphi'} \alpha']_{\varphi'}]_\varphi$
- $e_5 = (\lambda_z x.\, \alpha; z\, x)*$
  $H_5 = \mu h.\, \alpha \cdot h \quad \varepsilon, \alpha, \alpha\alpha, \ldots$

Validity of Histories

- obeying all the policies, within their scopes
- ex: $\varphi[\alpha_r]\alpha_c$ valid, $\varphi[\alpha_r\varphi'[\alpha_c]\alpha_w]$ not valid
- safe-sets of $\varphi[\alpha_r\varphi'[\alpha_c]\alpha_w]$:

$$\varphi[\{\varepsilon, \alpha_r, \alpha_r\alpha_c, \alpha_r\alpha_c\alpha_w\}] \qquad \varphi'[\{\alpha_r, \alpha_r\alpha_c\}]$$

- ex: $\psi\langle\alpha_{rvk}\alpha_{sgn}\rangle$ valid, $\psi\langle\alpha_{sgn}\alpha_{rvk}\psi\langle\rangle\alpha_{sgn}\rangle$ not
- live-sets of $\psi\langle\alpha_{sgn}\alpha_{rvk}\psi\langle\rangle\alpha_{sgn}\rangle$:

$$\psi\langle\{\varepsilon, \alpha_{sgn}, \ldots\}\rangle \qquad \psi\langle\{\alpha_{sgn}\alpha_{rvk}\}\rangle$$

Validity of Histories

- η valid iff, for each safe-set $\varphi[\{\eta_1, \ldots, \eta_k\}]$ of η:

$$\forall i \in 1..k.\ \eta_i \models \varphi$$

  and, for each live-set $\psi\langle\{\eta_1, \ldots, \eta_h\}\rangle$ of η:

$$\exists i \in 1..h.\ \eta_i \models \psi$$

- H valid iff each $\eta \in [\![H]\!]$ is valid
- how to verify validity of history expressions ?

Selecting Services

- service request: $\mathtt{req}_{\ell}\,\tau \xrightarrow{\varphi[\,],\psi\langle\,\rangle} \tau'$
- lookup Λ for services with signature:

$$e_i : \tau \xrightarrow{H_i} \tau'$$

- $I(\ell)$ selects those services such that:

$$\varphi[\psi\langle H_i\rangle] \text{ valid}$$

- the latent effect will be: $\Sigma_{i\in I(\ell)} H_i$

Type & Effect System

- typing judgements $\Gamma \vdash e : \tau \;\triangleright H, I$
- types: $\tau ::= unit \mid \tau \xrightarrow{H} \tau'$
- effect: history expr. H + service selection I
- correctness of history expressions:

$$\varepsilon, e \rightarrow^* \eta, e' \implies \eta \in [\![H]\!]$$

- type safety: if H is valid and $I(\ell) \neq \emptyset$ for each req_ℓ, then e will not go wrong

Verifying History Expressions

If validity were a regular property . . .

- $H \longrightarrow BPA(H)$ Pushdown automaton
- validity $\Omega(H) \longrightarrow A_{\Psi(H)}$ Büchi automaton
- H valid if $\mathcal{L}(BPA(H)) \cap \mathcal{L}(A_{\neg\Omega(H)}) = \emptyset$

But validity *of histories* is non-regular !

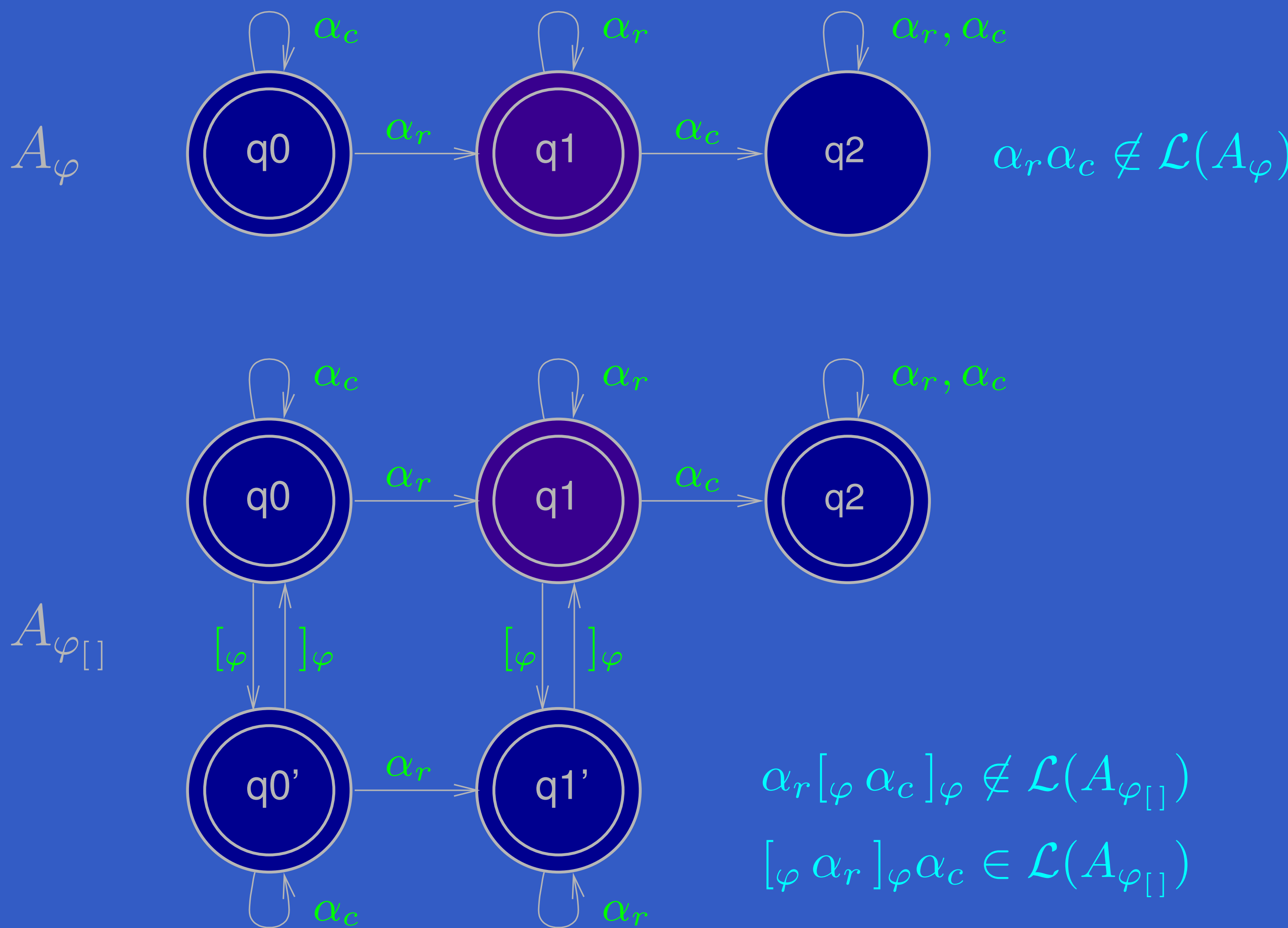
- ex: $\mu h.\, \alpha + h \cdot h + \varphi[h]$
- $[_\varphi\ [_\varphi\ \alpha\ ]_\varphi\ [_\varphi\ \alpha$ – in or out ?

Regularizing History Expressions

Solution: transform H to make $\Omega(H)$ regular

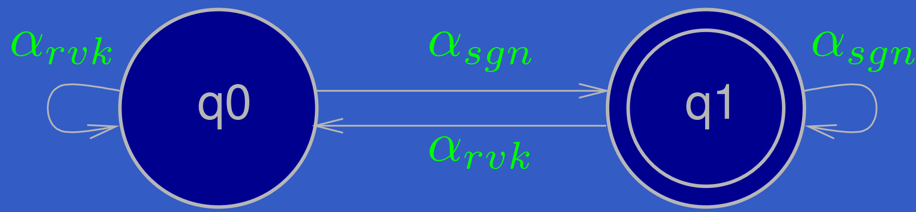
- idea: eliminating the *redundant* framings preserves the validity of history expressions
- $\varphi[\alpha\,\varphi'[\alpha'\,\varphi[\alpha'']]]$ valid iff $\varphi[\alpha\,\varphi'[\alpha'\alpha'']]$ valid
- $\psi\langle\alpha\,\psi'\langle\alpha'\,\psi\langle\alpha''\rangle\rangle\rangle$ valid iff $\alpha\,\psi'\langle\alpha'\,\psi\langle\alpha''\rangle\rangle$ valid
- safety framings can be regularized [Fossacs]
- liveness framings probably not, but not really needed (quite surprisingly!)

Verifying safety framings

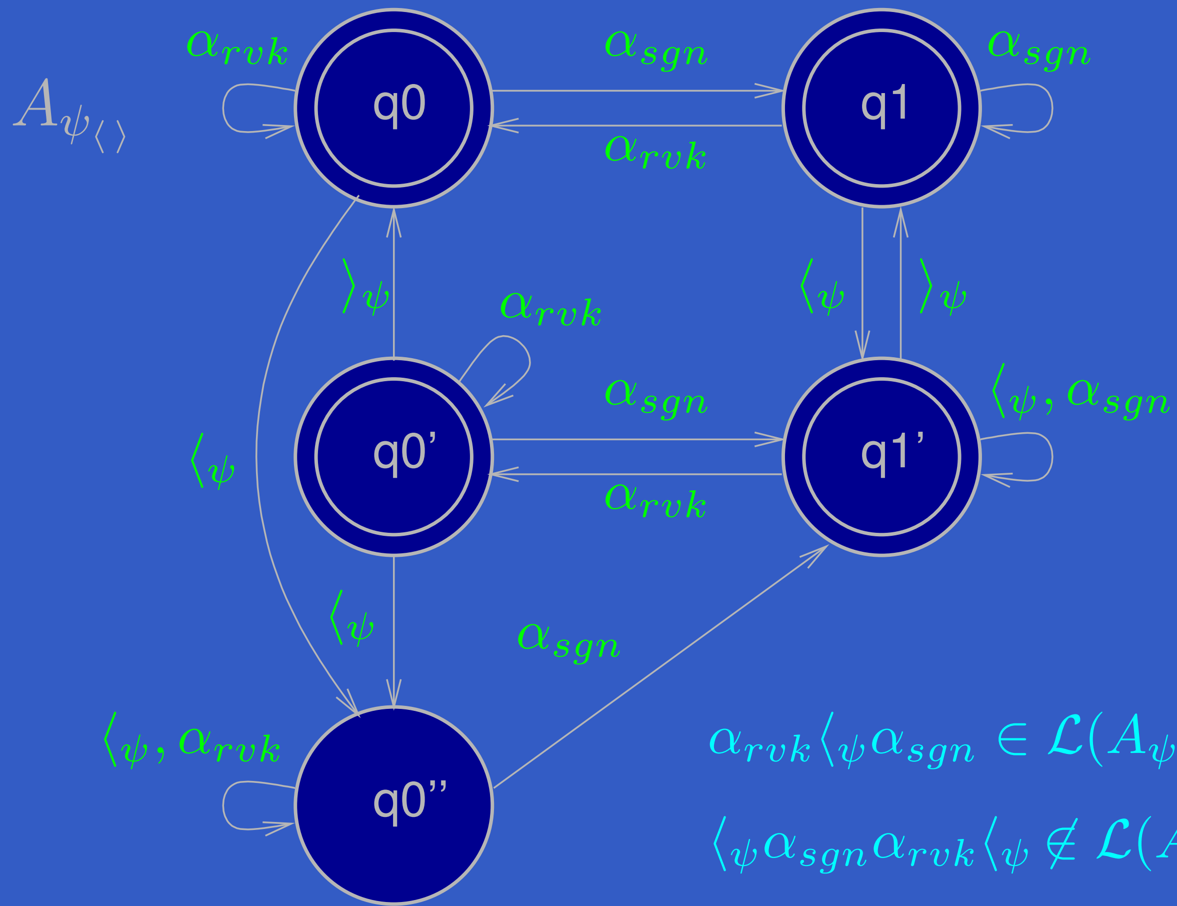


Verifying liveness framings

A_ψ



Verifying liveness framings



$\alpha_{rvk} \langle_\psi \alpha_{sgn} \in \mathcal{L}(A_{\psi_{\langle\rangle}})$

$\langle_\psi \alpha_{sgn} \alpha_{rvk} \langle_\psi \notin \mathcal{L}(A_{\psi_{\langle\rangle}})$

Conclusions

- λ + histories + local policies + call-by-contract
- type & effect system: $\Gamma \vdash e : \tau \rhd H, I$
- model-checking validity:

$$H \text{ valid} \iff [\![BPA(H\downarrow)]\!] \models \bigwedge_{[_\varphi, \langle_\psi \in H} \varphi_{[\,]} \wedge \psi_{\langle\,\rangle}$$

- type safety + verification:

$$H \text{ valid}, \forall \mathsf{req}_\ell.\, I(\ell) \neq \emptyset \implies e \text{ will not go wrong}$$

Programming model (syntax)

e, e'	$::=$	
	x	variable
	$\lambda_z x.\, e$	abstraction
	$e\, e'$	application
	$\texttt{if}\ b\ \texttt{then}\ e\ \texttt{else}\ e'$	conditional
	α	access event
	$\varphi[e]$	safety framing
	$\varphi\langle e\rangle$	liveness framing
	$\texttt{req}_{\ell}\, \tau \xrightarrow{\varphi[\,],\psi\langle\rangle} \tau'$	service request

Programming model (semantics)

$$\frac{}{\eta, \alpha \to \eta\alpha, *} \qquad \frac{e : \tau \xrightarrow{H} \tau' \in \Lambda \quad H \models \Box\varphi \wedge \Diamond\psi}{\eta, \mathtt{req}\ \tau \xrightarrow{\varphi[\,],\psi\langle\rangle} \tau' \to \eta, e}$$

$$\frac{\eta, e \to \eta', e' \quad \eta \models \varphi \quad \eta' \models \varphi}{\eta, \varphi[e] \to \eta', \varphi[e']} \qquad \frac{\eta \models \varphi}{\eta, \varphi[v] \to \eta, v}$$

$$\frac{\eta, e \to \eta', e' \quad \eta \not\models \psi}{\eta, \psi\langle e\rangle \to \eta', \psi\langle e'\rangle} \qquad \frac{\eta \models \psi}{\eta, \psi\langle e\rangle \to \eta, e}$$

Type & Effect System

$$\frac{}{\Gamma, \alpha \vdash \alpha : unit} \qquad \frac{\Gamma, H \vdash e : \tau}{\Gamma, \varphi[H] \vdash \varphi[e] : \tau} \qquad \frac{\Gamma, H \vdash e : \tau}{\Gamma, \psi\langle H\rangle \vdash \psi\langle e\rangle : \tau}$$

$$\frac{\Gamma; x : \tau; z : \tau \xrightarrow{H} \tau', H \vdash e : \tau'}{\Gamma, \varepsilon \vdash \lambda_z x.e : \tau \xrightarrow{H} \tau'} \qquad \frac{\Gamma, H \vdash e : \tau \xrightarrow{H''} \tau' \quad \Gamma, H' \vdash e' : \tau}{\Gamma, H \cdot H' \cdot H'' \vdash ee' : \tau'}$$

$$\frac{I_\ell = \{ i \mid e_i : \tau \xrightarrow{H_i} \tau' \in \Lambda \ \wedge \ \varphi[\psi\langle H_i\rangle] \text{ valid} \}}{\Gamma, \varepsilon, I_\ell \vdash \texttt{req}_\ell \, \tau \xrightarrow{\varphi[\,],\psi\langle\rangle} \tau' : \tau \xrightarrow{\Sigma_{i \in I_\ell} H_i} \tau'}$$

A typing example

- $e \;=\; \lambda_z x.\, b\,?\,\alpha \;+\; (b'\,?\,zzx \;+\; \varphi[zx])$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, \varepsilon \vdash z : \tau \xrightarrow{H} \tau \qquad \Gamma, \varepsilon \vdash x : \tau}{\Gamma, H \vdash z\,x : \tau}}{\Gamma, H \cdot H \vdash z\,z\,x : \tau}}{\Gamma, H \cdot H + \varphi[H] \vdash z\,z\,x : \tau} \qquad \dfrac{\Gamma, \varphi[H] \vdash \varphi[z\,x] : \tau}{\Gamma, H \cdot H + \varphi[H] \vdash \varphi[z\,x] : \tau}}{\Gamma, H \cdot H + \varphi[H] \vdash b'\,?\,z\,z\,x \;+\; \varphi[z\,x] : \tau}}{\Gamma, \alpha + H \cdot H + \varphi[H] \vdash b\,?\,\alpha \;+\; (b'\,?\,z\,z\,x \;+\; \varphi[z\,x]) : \tau}$$

- $H = \alpha + H \cdot H + \varphi[H] \implies H = \mu h.\, \alpha + h \cdot h + \varphi[h]$
- $\emptyset, \varepsilon \vdash e : unit \xrightarrow{\mu h.\, \alpha + h \cdot h + \varphi[h]} unit$

Semantics of History Expressions

$$[\![\varepsilon]\!]_\rho = \varepsilon \qquad [\![\alpha]\!]_\rho = \alpha \qquad [\![h]\!]_\rho = \rho(h)$$

$$[\![H \cdot H']\!]_\rho = [\![H]\!]_\rho \, [\![H']\!]_\rho$$

$$[\![H + H']\!]_\rho = [\![H]\!]_\rho \cup [\![H']\!]_\rho$$

$$[\![\varphi[H]]\!]_\rho = \varphi[[\![H]\!]_\rho]$$

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

Regularizing safety (1)

$$\varepsilon{\downarrow_{\Phi,\Gamma}} = \varepsilon \qquad h{\downarrow_{\Phi,\Gamma}} = h \qquad \alpha{\downarrow_{\Phi,\Gamma}} = \alpha$$

$$(H \cdot H'){\downarrow_{\Phi,\Gamma}} = H{\downarrow_{\Phi,\Gamma}} \cdot H'{\downarrow_{\Phi,\Gamma}}$$

$$(H + H'){\downarrow_{\Phi,\Gamma}} = H{\downarrow_{\Phi,\Gamma}} + H'{\downarrow_{\Phi,\Gamma}}$$

$$\varphi[H]{\downarrow_{\Phi,\Gamma}} = \begin{cases} H{\downarrow_{\Phi,\Gamma}} & \text{if } \varphi \in \Phi \\ \varphi[H{\downarrow_{\Phi\cup\{\varphi\},\Gamma}}] & \text{otherwise} \end{cases}$$

Regularizing safety (2)

$$(\mu h.\, H)\downarrow_{\Phi,\Gamma} = \ \mu h.\,(H'\sigma'\downarrow_{\Phi,\Gamma\{(\mu h.H)\Gamma/h\}}\sigma)$$

where $H = H'\{h/h_i\}_i$, h_i fresh, $h \notin fv(H')$

$$\sigma(h_i) \ = \ (\mu h.H)\Gamma\downarrow_{\Phi\cup guard(h_i,H'),\Gamma}$$

$$\sigma'(h_i) \ = \ \begin{cases} h & \text{if } guard(h_i, H') \subseteq \Phi \\ h_i & \text{otherwise} \end{cases}$$