



PLANNING SECURE SERVICE COMPOSITION



il problema che ci siamo posti era quello di fornire una orchestrazione sicura dei servizi. L'obiettivo è come possiamo progettare linguaggi di programmazione sicuri per questo tipo di attività.



Abbiamo di fronte un cliente che deve usare servizi distribuiti soggetti a politiche di sicurezza, il cliente può voler imporre delle politiche, l'obiettivo è comporre tutte queste caratteristiche insieme. L'orchestratore è una componente esterna che può usare una base linguistica per aiutare a fare in modo che il programma così sviluppato soddisfi le proprietà di sicurezza. L'orchestratore deve identificare staticamente quali sono i servizi richiesti dal cliente o dal programma del cliente in modo tale da rispettare i requisiti di sicurezza. Rispettando questa caratteristica l'orchestratore diventa parte della trusted computed base, ovvero diventa quella entità del run time support che noi definiamo trusted

PROBLEM & METHODOLOGY & TCB

- **The problem we face**
 - **The client program to behave safely (and correctly) requires the composition of services which respect certain security guarantees.**
- **The proposed methodology**
 - **Identify statically a trusted orchestrator that provides client the viable plans guaranteeing that the invoked services always respect the required properties.**
- **The orchestrator is the TCB**

una scelta locale può avere degli effetti successivi proprio per il fatto che noi abbiamo delle richieste di sicurezza che possono avere degli effetti laterali (information flow) sul comportamento complessivo. La selezione di quali servizi e vincoli usare per selezionare i servizi non dipende localmente soltanto dalla richiesta dell'orchestratore per risolvere la richiesta del cliente, ma dipende dall'orchiestrazione globale. In altri termini, la strategia eager che seleziona un servizio appena lo trova non scala a politiche di sicurezza più articolate. Questo vuol dire che dovremmo avere una strategia differente che tiene conto del fatto che siamo in un contesto distribuito con dei meccanismi distribuiti.

TRUSTED ORCHESTRATION

- Issue: **selecting a service for a request impacts not only the execution of that service but may affect the whole orchestration:**
 - **The selection of services that satisfy the constraints imposed by the requests, only does not scale up.**

La proposta che andremo a delineare è come si possono definire varie tecniche che sono state ideate durante la progettazione di diversi linguaggi di programmazione e il motivo per cui la vediamo è esattamente perchè fa vedere in un unico contesto tante tecniche differenti. Usiamo un sistema dei tipi che è leggermente diverso, che ha una proprietà di poter approssimare il possibile comportamento di tutti i servizi che sono coinvolti, quindi da una visione statica, astratta e simbolica del comportamento dell'orchestratore. Vogliamo avere una struttura statica, la quale deve avere non solo informazioni relativamente a cosa calcola l'ingresso, cosa calcola l'uscita e quale è la politica di sicurezza considerata, ma deve avere anche l'informazione di come avviene la computazione a tempo di esecuzione. E' un'informazione di natura simbolica che i sistemi di tipo più raffinati possono derivare dalla struttura del codice.

SECURE ORCHESTRATION STATICALLY

1. We extract the **abstract behaviour** of the orchestration via a suitable **type system** that **over-approximates the possible run-time behavior of all the services involved in an orchestration**
2. This abstract behaviour is **compiled** into a suitable form to be **model-checked** in order **to verify whether or not the security constraints are satisfied**
3. As a result of the model checking we build the **trusted orchestrator** that **securely coordinates the running services**.

Adesso abbiamo una rappresentazione astratta simbolica del comportamento, la compiliamo in una forma speciale che ci permette di fare un'operazione ulteriore, il MODEL CHECKING. Lo abbiamo visto all'inizio quando abbiamo parlato di protocolli di sicurezza e abbiamo fatto vedere come i model checker vanno a verificare delle proprietà. Quello che noi andremo a vedere usando il comportamento simbolico astratto, lo controlliamo rispetto alle politiche di sicurezza, questo vuol dire che va a verificare se tutte le politiche di sicurezza con i vincoli relativi sono soddisfatti. Da una risposta SI/NO sul comportamento astratto, prima di mandare in esecuzione il programma, a questo punto il risultato dell'orchestrazione nel caso in esame del model checking lo otteniamo andando a vedere come viene eseguito il model checking e quindi ci da un'idea di come derivare l'orchestratore trusted e a questo punto siamo sicuri che può orchestrare correttamente i servizi. COMPORTAMENTO ASTRATTO -> MECCANISMO DI VERIFICA -> PLACEMENT che associa alle etichette simboliche dei servizi degli endpoint. Questa strategia può essere usata quando in un contesto come quello dei SO si hanno diverse entità che devono essere sicure.

La prima parte: il sistema dei tipi si porta dietro un astrazione del comportamento, dice come un servizio esegue determinate operazioni rilevanti per la sicurezza, quindi bisogna capire come definire questa approssimazione del comportamento. Nel nostro contesto l'abbiamo chiamato history expression.

THE TYPE SYSTEM

Type & effect system

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

Le astrazioni funzionali nei linguaggi di programmazione funzionali sono delle coppie tau,tau', ho il tipo dell'argomento tau, restituisco il tipo del risultato tau'. Se voi usate Ocaml, quando nell'eval loop top level, nel caso di una funzione dice il tipo dell'argomento e del risultato. Solo che i tipi funzionali non solo dicono che il tipo tau viene trasformato nel tipo tau', allo stesso tempo da una astrazione di come fa a fare una trasformazione di questo tipo

TYPES

(basic types are pretty standard)

$\tau ::= \text{int} \mid \text{bool} \mid \text{unit} \mid \dots \mid$

$$\tau \xrightarrow{\mathsf{H}} \tau'$$

Functional types

Ci dice qual'è l'effetto, nel gergo dei sistemi di tipo ad effetto si chiama effetto latente, che è associato alla funzione, sostanzialmente è un insieme di comportamenti, ovvero quando a questa funzione fornisco in ingresso un argomento di tipo tau, allora il comportamento che mi produrrà un risultato di tipo tau' sarà selezionato tra questo insieme di comportamenti che sono descritti astrattamente da H, quindi l'history expression da una rappresentazione astratta delle possibili esecuzioni che posso avere quando alla funzione viene applicato un argomento in ingresso di tipo tau e produce come risultato un valore di tipo tau'

FUNCTIONAL TYPES

$$\tau \xrightarrow{H} \tau'$$

Functional types

The history expression H describes the latent effect associated with the functional abstraction (the set of possible execution histories)

When the abstraction is applied to a value, its execution will generate a run time behaviours belonging to the execution histories represented by H

Quando viene definito, e lo abbiamo visto nel caso dell'information flow, un giudizio di tipo quindi una regola di assegnamento del linguaggio, abbiamo un ambiente dei tipi gamma, quindi una tabella dei simboli o qualunque entità possiamo avere a tempo statico definito dal compilatore. Quando scriviamo nell'ambiente dei tipi gamma che l'espressione ha tipo tau, diciamo diverse cose, innanzitutto diciamo che e si comporta correttamente rispetto ai tipi, ovvero a tempo di esecuzione e non andrà in blocco a causa di un errore dei tipi, perchè i tipi che userà sono corretti. Questa ci dà una informazione di correttezza parziale rispetto ai tipi. Ci dice anche che produce un valore e il tipo del valore ha tipo tau.

INTERMEZZO

- Plain (traditional) type systems have judgments of the form

$$\Gamma \vdash e : \tau$$

- to mean:
 - **e won't get stuck**
 - **if e produces a value, that value has type τ**

INTERMEZZO

- Adding effects to typing rules mean to compute statically something about *“how program executes”*.
- There are many things we may want to conservatively approximate

Quando abbiamo un tipo effetto ci dice anche come produce questi valori, chiaramente vogliamo definire un sistema dei tipi che approssima in modo conservativo il comportamento a runtime, vuol dire che vogliamo avere più comportamenti astratti ma sicuramente tutti i comportamenti concreti sono contenuti nei comportamenti astratti.

Il primo passo che dobbiamo fare è capire come rappresentare simbolicamente i comportamenti del nostro linguaggio che stiamo progettando, il primo passo quindi è avere una rappresentazione simbolica dei comportamenti. La tecnica importante qui instanziata in un caso particolare è introdurre una rappresentazione simbolica dei comportamenti. I tipi a effetto erano stati introdotti per andare a descrivere nei linguaggi di programmazione funzionali che avevano anche inglobato meccanismi imperativi, e quindi operare con effetti laterali e puntatori, erano stati introdotti proprio per descrivere le proprietà dei tipi dei puntatori, quindi gli effetti delle operazioni sui puntatori. Questo è un esempio di uso di questi tipi, in altri contesti vengono chiamati tipi comportamentali.

EFFECTS (HISTORY EXPRESSIONS)

$H ::=$

ϵ

empty

h

variable

α

access event



$H \cdot H'$

sequence

$H + H'$

choice

$\mu h.H$

recursion

$\varphi[H]$

security block

$\ell : H$

localization

$\{\pi_1 \triangleright H_1 \dots \pi_k \triangleright H_k\}$

planned selection

Se non ci fosse la ricorsione ma avessimo l'iterazione, questa history expression sarebbe parenti stretti di espressioni regolari e quindi descritti da opportuni automi.

H è il la grammatica che mi descrive il linguaggio dei possibili comportamenti a tempo di esecuzione, quindi mi da tutti i possibili comportamenti che posso avere a tempo di esecuzione, in realtà me ne da di più perchè è un approssimazione in grande, ad esempio in H possono esserci dei comportamenti che sono concettualmente ammissibili ma non si verificheranno mai a tempo di esecuzione. A questo punto quello che noi dobbiamo fare è andare a vedere esattamente come determinare il significato che ora vedere intuitivo delle history expression.

WHAT IS THE MEANING OF HISTORY EXPRESSIONS?

$$\tau \xrightarrow{H} \tau'$$

**When the service is sent into execution,
it will generate one of the histories
abstract represented by H.**

History expressions are a sort of context-free grammars

**The language generated by this grammars are
abstract program behaviours**

Il primo passo è un passo sintattico dobbiamo rappresentare in un qualche modo il fatto che stiamo entrando in un blocco di sicurezza e usciamo da un blocco di sicurezza. Questo blocco di sicurezza è rappresentato da una politica PHI. Lo rappresentiamo astrattamente da quelle notazioni, quindi abbiamo delle azioni parametrizzate dalle politiche di sicurezza che stiamo considerando e la parentesi aperta Φ significa che stiamo entrando in un blocco altrimenti stiamo uscendo.

MEANIG OF HISTORY EXPRESSIONS (STEP I)

- Extend the set of actions with suitable **framing** events to denote
- The action of **entering** inside a security frame $[\varphi$
- The eaction of **exiting** from a security frame $]\varphi$

In questo caso l'history expression è fatta in questo modo: alfa concatenata all'apertura di sicurezza con politica PHI, esecuzione di alfa', uscita dal blocco di sicurezza caratterizzato dalla politica PHI. Che computazione rappresenta questa sequenza di azioni che contiene 4 azioni?

EXAMPLE

- The history

$$\alpha [\varphi \alpha']_\varphi$$

- Represents a computation that

- generates an event α , generazione di un evento di accesso alfa al di fuori di una qualunque politica di sicurezza
- enters the scope of φ operazione che entra nello scope della politica PHI
- generates α' within the scope of φ , generazione di evento di accesso all'interno dello scope di PHI
- leaves the scope of φ uscita dallo scope di sicurezza
- $\varphi[H]$ for $\{[\varphi \eta]_\varphi : \eta \in H\}$

Per semplicità quando scriverò $\text{PHI}[H]$ di fatto sto andando a prendere tutte le eta che ci sono all'interno e ci vado a mettere le azioni che rappresentano l'ingresso nel blocco e l'azione che rappresenta l'uscita dal blocco per tutti gli eta che sono comportamenti presenti nella history.

EXAMPLES

$$H = \ell_0 : \alpha_0 \cdot \{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1, r[\ell_2] \triangleright \ell_2 : \sigma \cdot \alpha_2\} \cdot \beta_0$$

$$\pi = r[\ell_1]$$

What is the set of run-time executions abstractly represented by H?

Immaginiamo di avere questa history expression rappresentata in questo modo. Alla locazione l0 localizzato eseguo una certa cosa, notate che l0 localizza tutta la H, caratterizzata da una azione alfa0, poi abbiamo un insieme che è una planned selection, quando cerco di risolvere alla locazione l1 la richiesta di servizio caratterizzata da r, a questo punto avrò un comportamento che mi dice: sono l1, faccio un azione sigma, ovvero l'attivazione del servizio ed eseguo alfa1, se invece lo risolvessi ad l2 avrei localizzato l'esecuzione ad l2, attivo il servizio e poi faccio l'azione alfa2. Dopo aver fatto la planned selection eseguo l'azione beta0 con qualunque politica. Adesso supponiamo che il nostro orchestratore ci dice: in questa esecuzione tu la richiesta del servizio etichettata con r la devi risolvere ad l1. Quindi adesso abbiamo in ingresso un astrazione di comportamento e un piano che ci dice dove andiamo a risolvere, quindi l'endpoint lo troviamo localizzato ad l1. Secondo voi qual'è l'insieme di tutte le possibili esecuzioni che sono presentate astrattamente dall'history expression H?

EXAMPLES

$$H = \ell_0 : \alpha_0 \cdot \{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1, r[\ell_2] \triangleright \ell_2 : \sigma \cdot \alpha_2\} \cdot \beta_0$$

$$\pi = r[\ell_1]$$

What is the set of run-time executions abstractly represented by H ?

Take location ℓ_0

$$\ell_0 : \alpha_0 \cdot \beta_0$$



Con l1 sicuramente il comportamento ad l1 viene caratterizzato dal fatto che prima ho eseguito l0, ma dal l1 non vedo alfa0 perchè vedo solo la storia locale di l1, quindi ad l1 vedo l'attivazione del servizio e l'azione che è fatta nel servizio quindi ne alfa0 che beta0. Essendo in un contesto distribuito dobbiamo tenere conto che le computazioni avvengono localmente e che abbiamo le storie locali, non abbiamo una nozione di storia globale di esecuzione. Adesso prendiamo la locazione l1.

EXAMPLES

$$H = \ell_0 : \alpha_0 \cdot \{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1, r[\ell_2] \triangleright \ell_2 : \sigma \cdot \alpha_2\} \cdot \beta_0$$

$$\pi = r[\ell_1]$$

What is the set of run-time executions abstractly represented by H?

Take location ℓ_1

$$\ell_1 : \sigma \cdot \alpha_1$$

Alla locazione l1, uno potrebbe dire di avere questo comportamento, eseguo l'attivazione, faccio alfa1 e poi concludo con beta0. Questa non è ammissibile per quello che dicevamo prima, perchè beta0 è localizzata in l0, quindi le storie sono locali!!

EXAMPLES

$$H = \ell_0 : \alpha_0 \cdot \{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1, r[\ell_2] \triangleright \ell_2 : \sigma \cdot \alpha_2\} \cdot \beta_0$$

$$\pi = r[\ell_1]$$

What is the set of run-time executions abstractly represented by H?

Take location ℓ_1

$$\ell_1 : \sigma \cdot \alpha_1 \cdot \beta_0$$

EXAMPLES

$$H = \ell_0 : \alpha_0 \cdot \{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1, r[\ell_2] \triangleright \ell_2 : \sigma \cdot \alpha_2\} \cdot \beta_0$$

$$\pi = r[\ell_1]$$

What is the set of run-time executions abstractly represented by H?

Take location ℓ_1

$$\ell_1 : \cancel{\alpha_0 \cdot \beta_0}$$

β_0 is located at ℓ_0

Secondo voi l2 sigma alfa2 potrebbe sembrare un comportamento legale localizzato ad l2. Nel contesto delle orchestrazioni che stiamo considerando in questo momento, ovvero che nel piano dell'orchestrazione sappiamo che la richiesta di servizio viene risolta da l1, in quel caso non si andrà mai ad l2 per cui, il piano non permette di risolvere la richiesta ad l2 quindi non è un comportamento atteso rappresentato da questa storia qui scritta.

EXAMPLES

$$H = \ell_0 : \alpha_0 \cdot \{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1, r[\ell_2] \triangleright \ell_2 : \sigma \cdot \alpha_2\} \cdot \beta_0$$

$$\pi = r[\ell_1]$$

What is the set of run-time executions abstractly represented by H?

$$\ell_2 : \sigma \cdot \alpha_2$$

EXAMPLES

$$H = \ell_0 : \alpha_0 \cdot \{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1, r[\ell_2] \triangleright \ell_2 : \sigma \cdot \alpha_2\} \cdot \beta_0$$

$$\pi = r[\ell_1]$$

What is the set of run-time executions abstractly represented by H ?



The service request cannot be solved by the current plan π

$$H = \ell_0 : (\mu h. \beta_0 + \alpha_0 \cdot \{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1\} \cdot h)$$

$$\pi = r[\ell_1]$$

The history expression represents a service (located at ℓ_0) that recursively execute α_0 and raises a request r which can be served by ℓ_1 only!!!

What is the set of run-time executions abstractly represented by H?

Facciamo un qualcosa che fa vedere come usare la ricorsione. La history expression è localizzata da ℓ_0 e poi ha un comportamento ricorsivo: ha un scelta tra e quindi vuol dire che ci sarà una guardia che guiderà l'esecuzione o nel fare una scelta che corrisponde a fare l'azione β_0 e terminare oppure l'altra possibilità è che la guardia ci permette di fare l'azione α_0 e poi fare una richiesta di servizio, in questo piano di orchestrazione verrà risolta da ℓ_1 e ad ℓ_1 verrà attivato il servizio che poi verrà eseguito α_1 . Poi si continua, perchè abbiamo la H che rappresenta la continuazione della ricorsione. Questa storia rappresenta un cliente in un qualche modo ricorsivo che ricorsivamente emette una richiesta di servizio dopo aver eseguito α_0 e a seguito di questa richiesta di servizio esegue un servizio alla locazione ℓ_1 ed è in grado solo ℓ_1 di farlo e lo fa in modo ricorsivo. Notate il motivo per cui non abbiamo usato la stella perchè un servizio come questo, il numero di attivazioni ricorsive è unbound ovvero è bound staticamente mentre con la stella di clini avrei potuto descrivere sicuramente sempre delle richieste di servizio limitate, perchè avrei avuto una iterazione limitata che corrisponde al fatto di avere una ripetizione bounded della richiesta di servizio. La domanda che ci poniamo è: quale'è l'insieme delle runtime execution che sono rappresentate in modo astratto da questa storia?

$$H = \ell_0 : (\mu h. \beta_0 + \alpha_0 \cdot \{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1\} \cdot h)$$

$$\pi = r[\ell_1]$$

The history expression represents a service (located at ℓ_0) that recursively execute α_0 and raises a request r which can be served by ℓ_1 only!!!

What is the set of run-time executions abstractly represented by H?

$$\ell_0 : \beta_0$$

$$\ell_1 : \sigma \cdot \alpha_1$$

$$\ell_0 : \alpha_0 \cdot \beta_0$$

$$\ell_0 : \alpha_0 \cdot \alpha_0 \cdot \beta_0$$

Sicuramente abbiamo la localizzazione l0 della sequenza che contiene solo beta0, abbiamo ad l0 la sequenza che fa alfa0 e termina con beta0 oppure abbiamo una sequenza che fa alfa0, alfa0 e poi termina, vi immaginate che avremo anche alfa0, alfa0, alfa0 e poi termina e così via. Ad l1 abbiamo soltanto l'attivazione con alfa1 quindi la localizzazione e il fatto di avere ancora una volta una storia locale mi permette di discriminare i comportamenti.

$$H = \ell_0 : (\mu h. \beta_0 + \alpha_0 \cdot \{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1\} \cdot h)$$

$$\pi = r[\ell_1]$$

The history expression represents a service (located at ℓ_0) that recursively execute α_0 and raises a request r which can be served by ℓ_1 only!!!

What is the set of run-time executions abstractly represented by H?

$$\ell_0 : \beta_0$$

$$\ell_1 : \sigma \cdot \alpha_1$$

$$\ell_0 : \alpha_0 \cdot \beta_0$$

$$\ell_1 : \sigma \cdot \alpha_1 \textcolor{red}{\cancel{\cdot}} \alpha_1$$

$$\ell_0 : \alpha_0 \cdot \alpha_0 \cdot \beta_0$$

Perchè l1 non ha anche la sequenza: l1 sigma alfa1 sigma alfa1? Immaginate di localizzare ad l1 la sequenza che contiene 2 attivazioni successive del servizio l1, secondo voi è un comportamento che ci si aspetta a runtime?

$$H = \ell_0 : (\mu h. \beta_0 + \alpha_0 \cdot \{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1\} \cdot h)$$

$$\pi = r[\ell_1]$$

The history expression represents a service (located at ℓ_0) that recursively execute α_0 and raises a request r which can be served by ℓ_1 only!!!

What is the set of run-time executions abstractly represented by H?

$$\ell_0 : \beta_0$$

$$\ell_0 : \alpha_0 \cdot \beta_0$$

$$\ell_0 : \alpha_0 \cdot \alpha_0 \cdot \beta_0$$

$$\begin{array}{c} \ell_1 : \sigma \cdot \alpha_1 \\ \ell_1 : \sigma \cdot \alpha_1 \end{array}$$


Essendo stateless il servizio, non vengono memorizzate insieme ma è come se fosse una nuova esecuzione. Se io avessi la sequenza che fa vedere che ho attivato 2 volte sigma alfa1 vuol dire che avrei servizi statefull ma questo non è un comportamento atteso.

SERVICES ARE STATELESS



THE FORMAL DEFINITION





Dopo aver detto che vogliamo servizi stateless adesso gioco un pochino con la formalità, conviene partire da servizi statefull ovvero avere tutte le possibili sequenze di esecuzione. Adesso andiamo a vedere cosa significa questo e andiamo a vedere come definire tutto questo meccanismo.

THE STATEFULL MEANING


$$\llbracket h \rrbracket_\theta^\pi = \theta(h)$$

$$\llbracket \varepsilon \rrbracket_\theta^\pi = (?) : \{\varepsilon\})$$

$$\llbracket \alpha \rrbracket_\theta^\pi = (?) : \{\alpha\})$$

$$\llbracket \ell : H \rrbracket_\theta^\pi = \llbracket H \rrbracket_\theta^\pi \{ \ell / ? \}$$

Il significato delle history expression sono definite in base a un piano e quello è il pi che c'è in alto e poi ha un teta che fa vedere l'associazione tra una variabile e una history expression. Questo mapping delle variabili di storie serve per la gestione della ricorsione. Qual'è il significato di una funzione? Vado a vedere nell'ambiente dei nomi quale è la history expression associata ad h. Quando trovo la history expression vuota, ho l'insieme che contiene soltanto la stringa epsilon quindi ho sempre la stringa vuota e poi ci metto una localizzazione con un punto interrogativo, nel senso che non so ancora dove verrà visualizzata dato che sto andando per composizione e soltanto quando raggiungerò il costrutto che me la localizza a quel punto interrogativo andrò a metterci l'effettivo luogo endpoint dove viene localizzata l'espressione. Quando eseguo un evento di accesso similmente ottengo l'insieme dei comportamenti che contiene soltanto l'azione alfa, anche questo non viene localizzato, quindi c'è un punto interrogativo perchè non so ancora dire quando verrà localizzato. Verrà localizzato non appena trova il costrutto di localizzazione !:h, quindi vado in modo compositivo a vedere quale è l'insieme delle storie associate a h e poi a tutti i punti interrogativi che sono presenti in h che ho generato per l'epsilon e per l'alfa, ci vado a sostituire !, quindi sto dicendo che a questo punto l'insieme dei comportamenti è localizzato ad !, che è esattamente il significato del costrutto di localizzazione.


$$[\![\varphi[H]]]_{\theta}^{\pi} = \varphi[\![H]\!]_{\theta}^{\pi}$$

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

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

Andiamo a vedere quale è il significato dell'insieme delle storie h associato alla politica PHI supponendo di avere il piano PI e teta che mi definisce il nome delle variabili. Quello che faccio è procedere in modo **composizionale**, vado a calcolare l'insieme delle storie dell'argomento h senza considerare il framing e poi ci vado a mettere a tutte queste storie il tag di ingresso e in fondo ci vado a mettere il tag di chiusura, vi ricordate che con PHI applicato a un insieme di storie, è la metanotazione che abbiamo usato e abbiamo introdotto in precedenza per dire che mettiamo l'accesso all'inizio e l'uscita alla fine. Composizione sequenziale: ho due storie H H' , introduco un punto cerchiato che mi fa la concatenazione. Possiamo immaginare che prende tutti i comportamenti del primo, del secondo e fa la composizione in base al fatto che ci devono essere delle compatibilità, ad esempio se ho un comportamento localizzato in H ad I_1 e I_2 , e in H' ho un comportamento localizzato ad I_2 e I_3 , quando faccio la composizione avrò un comportamento localizzato ad I_1, I_2, I_3 e devo fare il matching degli opportuni comportamenti dove sono localizzati, non posso mettere il comportamento localizzato ad I_1 seguito dal comportamento localizzato ad I_2 , proprio per il fatto che ho delle storie locali. Quello detto a voce è sostanzialmente la definizione un pò più precisa dell'operatore .

La somma fa l'unione

Andiamo a vedere la ricorsione ma prima dobbiamo dire che le H sono insiemi di storie quindi può essere ordinato in base alla relazione di inclusione tra insiemi, quindi un insieme è più piccolo di un altro insieme se tutti i comportamenti con tutte le storie che stanno sul primo insieme sono anche contenute sul secondo. Però questo operare con insiemi con questa struttura è difficile per poi dare significato ad operazioni di punto fisso, che vuol dire definire una funzione che opera su un opportuno spazio algebrico e prende un valore x e da un risultato. Un punto fisso di $f(x)$ è esattamente un valore x_0 tale che quando applico la f ad x_0 ottengo esattamente x_0 , quindi è un valore che è esattamente la soluzione di un'equazione del tipo $x = f(x)$, quando trovo x_0 che risolve questa equazione ho trovato il punto fisso di quella funzione. Ci sono varie tecniche matematiche, in modo particolare se prendo le funzioni che sono contrattive sugli spazi metrici, ho un teorema di Banach che mi dice che le funzioni contrattive su spazi metrici completi hanno un punto fisso, quindi l'idea è che ho uno spazio con una nozione di distanza, le funzioni contrattive sono funzioni che quando prendo un valore restringono l'informazione e la rendono più vicina, quindi la restringe fin tanto che riesce ad ottenere l'informazione che mi dà il risultato del punto fisso. Esistono anche un'altra tecnica introdotta su reticolati completi per cui una funzione continua su un reticolo completo ha un reticolo completo di punti fissi, un altro risultato è che se io vado a prendere un ordinamento parziale completo, l'ordinamento parziale completo non è un reticolo ma una struttura d'ordine dove ho un elemento minimo e poi ho una nozione di ordinamento e poi ho che ogni insieme diretto, quindi un insieme ordinato totalmente ammette un estremo superiore. Allora una funzione continua su ordinamento parziale completi ammette il minimo punto fisso.

$$\llbracket \mu h.H \rrbracket_{\theta}^{\pi} = \bigcup_{n>0} f^n(\{\ell_i : \{!\}\}_i) \text{ where } f(X) = \llbracket H \rrbracket_{\theta\{X/h\}}^{\pi}$$

$$\llbracket \{\pi_1 \triangleright H_1 \cdots \pi_k \triangleright H_k\} \rrbracket_{\rho}^{\pi} = \bigoplus_{i \in 1..k} \llbracket \{\pi_i \triangleright H_i\} \rrbracket_{\rho}^{\pi}$$

Le storie nel nostro caso vogliamo considerarle come dei cpo, quindi degli ordinamenti parziali completi ma facendo un'operazione leggermente diversa. L'elemento minimo lo mettiamo, si chiama bottom e lo aggiungiamo, sono strutture in più che creiamo per renderlo un cpo, quindi ci mettiamo l'evento minimo e questo è bottom e poi estendiamo la nozione di ordinamento dovuta al sottoinsieme alla presenza di questo elemento minimo. A questo punto sappiamo che questo è un cpo e quindi sappiamo che esiste un minimo punto fisso. A questo punto come definiamo il significato di una ricorsione? Il significato della ricorsione lo definiamo in questo modo: introduciamo un particolare elemento BANG che ci sta ad indicare una storia che può essere potenzialmente estesa, allora a questo punto quando vado a definire il significato della ricorsione vuol dire che lo definisco prendendo il punto fisso di quella funzione f^n che parte e applica prima f_0 poi f_0 applica f_1 al risultato di f_0 e così via, quindi prendo l'operazione di limite di questa funzione e questa f è definita come l'approssimazione al passo precedente di H . Quindi inizialmente parto con il punto esclamativo e poi prendo l'insieme di tutte le storie che ci aggiungono una azione e poi terminano con bottom. A quel punto vado a prendere il punto fisso di questa definizione di funzione e questa dato che l'insieme delle storie è un CPO ho un risultato e abbiamo ottenuto il punto fisso.

A questo punto andiamo a vedere la planned selection con cui faccio l'unione per tutte le possibili scelte che sono compatibili.



$$\llbracket \{0 \triangleright H\} \rrbracket_{\rho}^{\pi} = \llbracket H \rrbracket_{\rho}^{\pi} \quad \llbracket \{\pi_0 \mid \pi_1 \triangleright H\} \rrbracket_{\rho}^{\pi} = \llbracket \{\pi_0 \triangleright H\} \rrbracket_{\rho}^{\pi} \oplus \llbracket \{\pi_1 \triangleright H\} \rrbracket_{\rho}^{\pi}$$

$$\llbracket \{r[\ell] \triangleright H\} \rrbracket_{\rho}^{\pi} = \begin{cases} \llbracket H \rrbracket_{\rho}^{\pi} & \text{if } \pi = r[\ell] \mid \pi' \\ (? : \perp) & \text{otherwise} \end{cases}$$

La compatibilità è definita da questa parte: se sono in un piano vuoto e voglio vedere cosa è compatibile con H , nel piano vuoto non faccio scelte e quindi è sicuramente un H . Se sono in un piano che ha una definizione π_0 e una definizione π_1 , a questo punto vado a decomporre, prendo le storie secondo il piano π_0 e le unisco alle storie secondo il piano π_1 , quindi sto operando per composizione. Finalmente arrivo al caso base, ovvero un piano che dice di risolvere in I il servizio di richiesta e poi avere H come storia, a questo punto devo andare a vedere se questo piano $r[I]$ è compatibile con il piano che in questo momento sto considerando. Se è compatibile, vuol dire che è esattamente una scelta specifica del piano, a questo punto vado a scegliere il relativo H e vado a valutarlo, se non è così non sono localizzato e magicamente introduco il bottom quindi sono in una situazione in cui non produrrò risultati perché rimango nel minimo dell'ordinamento e nel minimo dell'ordinamento non ho informazione significativa.

Andiamo ad eseguire l'operazione e la prima cosa che andiamo a fare è di localizzarla ad ℓ_0 e poi vado ricorsivamente con la storia localizzata. A questo punto eseguo alfa0 ma alfa0 è in questo momento dato che non ho ancora applicato la sostituzione, è localizzata tramite punto interrogativo, il risultato è alfa0 concatenato con il significato della plan selection e concateno con l'esecuzione del beta0. A questo punto vado a valutare la planned selection e visto che il piano è ℓ_1 devo fare la somma con qualcosa che è bottom quindi vuol dire che quello che io vado ad ottenere è soltanto la parte che è localizzata da ℓ_1 che è compatibile con il piano che ho fatto allora a questo punto vado a selezionare soltanto sigma alfa1 mi da esattamente il comportamento ad ℓ_1 a questo punto trasporto dentro la localizzazione e notate che faccio l'operazione di composizione, che deve preservare il fatto che i comportamenti sono localizzati, per cui ad ℓ_0 avrò alfa0 beta0 che non lo posso concatenare con quello che aveva ℓ_1 perchè sono 2 locazioni diverse, ad ℓ_1 ho sigma alfa1, che è esattamente quello che ci aspettavamo prima

$$H = \ell_0 : \alpha_0 \cdot \{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1, r[\ell_2] \triangleright \ell_2 : \sigma \cdot \alpha_2\} \cdot \beta_0$$

$$\pi = r[\ell_1]$$

$$\begin{aligned} & [\![\alpha_0 \cdot \{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1, r[\ell_2] \triangleright \ell_2 : \sigma \cdot \alpha_2\} \cdot \beta_0]\!]^\pi \{\ell_0/?\} \\ &= ((?:\{\alpha_0\}) \odot [\![\{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1, r[\ell_2] \triangleright \ell_2 : \sigma \cdot \alpha_2\}]\!]^\pi \\ &\quad \odot(?:\{\beta_0\})) \{\ell_0/?\} \\ &= ((?:\{\alpha_0\}) \odot [\![\ell_1 : \sigma \cdot \alpha_1]\!]^\pi \odot(?:\{\beta_0\})) \{\ell_0/?\} \\ &= ((?:\{\alpha_0\}) \odot (\ell_1 : \{\sigma \alpha_1\}) \odot(?:\{\beta_0\})) \{\ell_0/?\} \\ &= (?:\{\alpha_0 \beta_0\}, \ell_1 : \{\sigma \alpha_1\}) \{\ell_0/?\} \\ &= (\ell_0 : \{\alpha_0 \beta_0\}, \ell_1 : \{\sigma \alpha_1\}) \end{aligned}$$



Questa era la stateful perchè se io avessi una definizione ricorsiva del servizio da I1 avrei concatenato tutte le possibili attivazioni del servizio perchè avrei fatto la composizione. Quello che noi abbiamo fatto è usare la comodità della definizione per avere un modo semplice per descrivere i comportamenti e poi renderli stateless .

THE STATELESS SEMANTICS

Invece di avere le parentesi quadrate abbiamo le parentesi angolate doppie, consideriamo un piano ed essendo stateless sono tutte localizzate ad I, quindi abbiamo localizzato le storie eta, che appartiene alla semantica del comportamento che ottengo con H con quella statefull, poi quello che succede è che sigma non appartiene ad eta allora il risultato è eta, quindi vuol dire che sigma non contiene attivazione su eta, questo è il motivo per cui abbiamo l'azione sigma diversa da tutte le altre azioni proprio perchè vogliamo marcare il territorio nelle situazioni in cui vogliamo considerare il fatto che stiamo attivando il servizio. Se invece eta è fatto da un eta0, poi un attivazione ed eta1 allora quello corrisponde a 2 attivazioni consecutive del servizio ma noi siamo in un meccanismo stateless vuol dire che dobbiamo prendere eta0 e renderlo stateless, dobbiamo unirlo ai comportamenti di eta1 stateless, ovviamente se lo stesso comportamento viene ripetuto più volte nell'unione la ripetizione viene persa.

$$\langle\langle H \rangle\rangle^\pi = \{ \ell : \{ \langle\langle \eta \rangle\rangle \mid \eta \in \mathcal{H} \} \mid \ell : \mathcal{H} \in \llbracket H \rrbracket_\emptyset^\pi \}$$

$$\text{where } \langle\langle \eta \rangle\rangle = \begin{cases} \eta & \text{if } \sigma \notin \eta \\ \langle\langle \eta_0 \rangle\rangle \cup \langle\langle \eta_1 \rangle\rangle & \text{if } \eta = \eta_0 \sigma \eta_1 \end{cases}$$

prendiamo come esempio quello di prima e nella semantica stateless togliamo l'attivazione dei servizi quindi uno non vede come sono attivati i servizi e quindi nel nostro H che aveva $\sigma \cdot \alpha_1$ quello che noi vediamo è che abbiamo eseguito alfa, abbiamo eseguito beta0 e poi su ℓ_1 vediamo l'unica operazione osservabile che è alfa1.

$$H = \ell_0 : \alpha_0 \cdot \{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1, r[\ell_2] \triangleright \ell_2 : \sigma \cdot \alpha_2\} \cdot \beta_0$$

$$\pi = r[\ell_1]$$

$$\begin{aligned} & [\![\alpha_0 \cdot \{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1, r[\ell_2] \triangleright \ell_2 : \sigma \cdot \alpha_2\} \cdot \beta_0]\!]^\pi \{\ell_0/?\} \\ &= ((? : \{\alpha_0\}) \odot [\![\{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1, r[\ell_2] \triangleright \ell_2 : \sigma \cdot \alpha_2\}]\!]^\pi \\ &\quad \odot (? : \{\beta_0\})) \{\ell_0/?\} \\ &= ((? : \{\alpha_0\}) \odot [\![\ell_1 : \sigma \cdot \alpha_1]\!]^\pi \odot (? : \{\beta_0\})) \{\ell_0/?\} \\ &= ((? : \{\alpha_0\}) \odot (\ell_1 : \{\sigma \alpha_1\}) \odot (? : \{\beta_0\})) \{\ell_0/?\} \\ &= (? : \{\alpha_0 \beta_0\}, \ell_1 : \{\sigma \alpha_1\}) \{\ell_0/?\} \\ &= (\ell_0 : \{\alpha_0 \beta_0\}, \ell_1 : \{\sigma \alpha_1\}) \end{aligned}$$

$$\langle\langle H \rangle\rangle^\pi = (\ell_0 : \{\alpha_0 \beta_0\}, \ell_1 : \{\alpha_1\})$$

l'esempio che facevamo all'inizio con il servizio che faceva beta0 e poi ricorsivamente attiva il servizio ad l1 dopo aver eseguito alfa0. Se a questo punto vado nel piano pigreco che viene risolto da l1 che è l'unico piano localizzato dalla planned selection, quello che vado a fare è andare a risolvere l'operazione di punto fisso, quindi faccio alfa0 + beta0 dove risolvo questa operazione di punto fisso tramite questo costrutto visto prima che sostanzialmente quello che fa in termini intuitivi parte da bottom, poi mette beta0, poi ci mette beta0 bottom più alfa0 con bottom, poi ci va a mettere beta0 più alfa0 bottom e così via, e infatti ottengo beta0, poi alfa0, poi alfa0 beta0, e così via....

$$\begin{aligned}
 \llbracket H \rrbracket^\pi &= \llbracket \ell_0 : \mu h. \beta_0 + \alpha_0 \cdot \{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1\} \cdot h \rrbracket^\pi \\
 &= \llbracket \mu h. \beta_0 + \alpha_0 \cdot \{r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1\} \cdot h \rrbracket^\pi \{\ell_0/?\} \\
 &= (\bigcup_{n>0} \llbracket f^n(\ell_0 : \{!\}, \ell_1 : \{!\}) \rrbracket^\pi) \{\ell_0/?\}
 \end{aligned}$$

by the fixpoint construction

$$\begin{aligned}
 &(\ell_0 : \{\beta_0, \alpha_0!, \alpha_0\beta_0, \alpha_0\alpha_0!, \alpha_0\alpha_0\beta_0, \alpha_0\alpha_0\alpha_0! \dots\}, \\
 &\quad \ell_1 : \{\varepsilon, \sigma\alpha_1, \sigma\alpha_1!, \sigma\alpha_1\sigma\alpha_1, \sigma\alpha_1\sigma\alpha_1!, \dots\})
 \end{aligned}$$

Se vado a prendere la semantica stateless vedete che quello che contiene è che faccio il comportamento beta0, poi posso fare alfa0 con ! vuol dire che ho un comportamento che può essere esteso, il punto esclamativo mi sta ad indicare esattamente che può essere esteso. Può essere esteso concludendo oppure con un altro comportamento non ancora terminato ovvero estendendo alfa0 mettendoci il !.

$$(\ell_0 : \{\beta_0, \alpha_0!, \alpha_0\beta_0, \alpha_0\alpha_0!, \alpha_0\alpha_0\beta_0, \alpha_0\alpha_0\alpha_0! \dots\}, \\ \ell_1 : \{\varepsilon, \sigma\alpha_1, \sigma\alpha_1!, \sigma\alpha_1\sigma\alpha_1, \sigma\alpha_1\sigma\alpha_1!, \dots\})$$

The stateless semantics $\langle\langle H \rangle\rangle^\pi$ is the set:

$$(\ell_0 : \{\beta_0, \alpha_0!, \alpha_0\beta_0, \alpha_0\alpha_0!, \dots\}, \ell_1 : \{\varepsilon, \alpha_1, \alpha_1!\})$$

Abbiamo definito un modo per rappresentare i comportamenti, adesso dobbiamo dire: abbiamo un modo per rappresentare solo storie corrette? Immaginate di avere la politica no write after read...

VALIDITY

Histories are valid when they arise from computations that do not violate any security constraint

$$\eta = \alpha_w \cdot \alpha_r \cdot \varphi[\alpha_w]$$

φ = no write after read

VALIDITY

Histories are valid when they arise from computations that do not violate any security constraint

$$\eta = \alpha_w \cdot \alpha_r \cdot \varphi[\alpha_w]$$



NOT VALID

φ = no write after read

Questa storia qui sappiamo non essere valida.

Riusciamo a fare in modo di rappresentare questa nozione di validità e avere un modo per rappresentare sui comportamenti astratti esattamente quali sono le storie valide? Perchè se avessimo già questo avremmo un mucchio di informazione che non è soltanto sintattica ma cominceremmo a mettere dell'astrazione del comportamento che abbiamo visto un po di informazione relativa alle policy di sicurezza. Se riuscissimo a fare questo potremmo saltare al punto 2 di questa metodologia definendo quando questi comportamenti astratti soddisfano delle proprietà in modo particolare delle proprietà di sicurezza. Definiamo con eta pallino 0 mi dice che prendiamo una storia e togliamo tutti i framing sia di apertura che chiusura. Con eta^delta voglio prendere tutti gli insiemi dei prefissi della storia eta, epsilon, immaginiamo fi avere la storia alfa beta e voglio prendere i prefissi: quindi epsilon, alfa, alfa beta.

η° is the history obtained by erasing all framing events

η^δ is the set of all prefixes of the history η

$$\eta = \alpha_w \cdot \alpha_r \cdot \varphi[\alpha_w]$$

$$\eta' = \eta^\circ = \alpha_w \cdot \alpha_r \cdot \alpha_w$$

$$\eta'^\delta = \{\epsilon, \alpha_w, \alpha_w \cdot \alpha_r, \alpha_w \cdot \alpha_r \cdot \alpha_w\}$$

Il safe set di una storia η lo definiamo intuitivamente: mi dice se la storia è epsilon bene il safe set è vuoto, se la storia consiste di un eta seguito da un alfa, e notate che il safe sets devono sostanzialmente essere usati per rappresentare i comportamenti che verificano le proprietà di sicurezza, non ho framing allora mi mangio l'alfa e dato che alfa non è soggetta ad alcuna politica questa storia è corretta se quello che mi è stato fatto arrivare ad alfa soddisfa la politica. Adesso per definire il safe set mi rimane esattamente la gestione del framing.

SAFE SETS & VALIDITY

The *safe sets* $S(\eta)$ of a history η are defined as:

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

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:

$$\langle\langle H \rangle\rangle^\pi \neq (?) : \perp \text{ and } \forall \ell : \forall \eta \in \langle\langle H \rangle\rangle^\pi @ \ell : \models \eta$$

where $(\ell_i : \mathcal{H}_i)_{i \in I} @ \ell_j = \mathcal{H}_j$.

Vado a caratterizzare il safeset di eta0 seguito da eta1, a cui però ci vado a mettere l'insieme che ottengo mettendo l'accesso e l'uscita degli eventi a cosa? A eta0 in cui gli ho tolto tutti gli altri framing event, quindi vado a verificare se la politica PHI è verificata, in eta0 e vado a prendere tutti i possibili prefissi di eta1 perchè poi eta1 è soggetto alla politica, l'idea di questa operazione è che mi devo ricordare del passato che mi ha portato ad eta1, il passato può avere al suo interno dei frame, ma devo averli già risolti quando sono arrivato lì, quindi depuro dai framing, mi tengo soltanto gli eventi e poi vado a vedere se tutti i prefissi di eta1 soddisfano la politica, perchè la politica deve essere soddisfatta ad ogni step dell'esecuzione. Una volta definito un safe set dico che una storia è valida e come notazione metto models eta, se quando vado a vedere un insieme di storie che sono soggette alla politica, allora per ogni storia H deve andare a verificare ogni politica PHI, allora a questo punto ho una nozione di history expression che è valida quando vado a localizzare la politica rispetto al path, lo localizzo rispetto al path e alla locazione e a questo punto la storia è valida rispetto a quella particolare locazione.

Supponiamo di avere questa storia caratterizzata da una politica esterna che dice che non devo mai eseguire alfa3. Quando a vedere il safe set associato alla semantica stateless della storia H localizzata ad l1 ottengo il PHI applicato ad epsilon, alfa0 e alfa1, sto semplicemente applicando le operazioni viste prima. Se invece vado a fare l'operazione del safe set alla storia H con il plan che invece mi va a risolvere la richiesta del servizio ad l2 ottengo prima la politica che è uguale a prima e poi la politica PHI' che ha alfa0 e alfa2.

EXAMPLE

$$H = \varphi[\alpha_0 \cdot \{r[\ell_1] \triangleright \alpha_1, r[\ell_2] \triangleright \varphi'[\alpha_2]\}] \cdot \alpha_3$$

$\varphi = \text{never } \alpha_3$

$\varphi' = \text{never } \alpha_2$

$$S(\langle\langle H \rangle\rangle^{r[\ell_1]}) = S([\varphi \alpha_0 \alpha_1]_\varphi \alpha_3) = \{ \varphi[\{\varepsilon, \alpha_0, \alpha_0 \alpha_1\}] \}$$

$$\begin{aligned} S(\langle\langle H \rangle\rangle^{r[\ell_2]}) &= S([\varphi \alpha_0 [\varphi' \alpha_2]_{\varphi'}]_\varphi \alpha_3) \\ &= \{ \varphi[\{\varepsilon, \alpha_0, \alpha_0 \alpha_2\}], \varphi'[\{\alpha_0, \alpha_0 \alpha_2\}] \} \end{aligned}$$

Quello che succede applicando la notazione è che la prima è valida perchè PHI mi dice "mai alfa3" e dei comportamenti che sono derivati dalla semantica statefull con l'insieme dei safe sets non ho sotto PHI alfa0, mentre la seconda ho un comportamento dove compare alfa2 e alfa2 è sotto la politica PHI' che dice che non deve mai essere eseguito alfa2 e quindi questa nozione permette di rappresentare comportamenti validi rispetto alle politiche.

EXAMPLE

$$H = \varphi[\alpha_0 \cdot \{r[\ell_1] \triangleright \alpha_1, r[\ell_2] \triangleright \varphi'[\alpha_2]\}] \cdot \alpha_3$$

$\varphi = \text{never } \alpha_3$

$\varphi' = \text{never } \alpha_2$

$r[\ell_1]$ -valid

$$S(\langle\langle H \rangle\rangle^{r[\ell_1]}) = S([\varphi \alpha_0 \alpha_1]_\varphi \alpha_3) = \{ \varphi[\{\varepsilon, \alpha_0, \alpha_0 \alpha_1\}] \}$$

$$S(\langle\langle H \rangle\rangle^{r[\ell_2]}) = S([\varphi \alpha_0 [\varphi' \alpha_2]_{\varphi'}]_\varphi \alpha_3)$$

not $r[\ell_2]$ -valid

$$= \{ \varphi[\{\varepsilon, \alpha_0, \alpha_0 \alpha_2\}], \varphi'[\{\alpha_0, \alpha_0 \alpha_2\}] \}$$



THE TYPE SYSTEM





TYPING JUDGEMENTS

$$\Gamma, H \vdash e : \tau$$

**the service e evaluates to a
value of type τ , and produces
a history denoted by the
effect H**

Il sistema è un sistema dei tipi ad effetto dove si vuol fornire non solo le informazioni relative al fatto che un programma, in questo caso siamo in un linguaggio funzionale e l'espressione ha tipo tau, ma vuole anche introdurre la possibilità di dare un astrazione simbolica del comportamento dell'espressione. Una struttura linguistica che ci dice come si comporta l'esecuzione del comando dell'espressione che stiamo considerando, per cui i giudizi di tipo ovvero le regole che staticamente ci permettono di affermare che un espressione ha un tipo tau, sono in realtà diverse e della forma gamma, dove gamma è la tabella dei simboli che associa ad ogni nome presente nel programma il relativo tipo, e permette di dedurre una storia in questo H che è una history expression che rappresenta un astrazione simbolica del comportamento dell'espressione e che quando viene mandato in esecuzione restituisce un valore di tipo tau, H è l'effetto latente dell'esecuzione dell'espressione.

supponiamo di avere un servizio il cui codice è e, supponiamo che questo servizio è pubblicato su l. Dobbiamo prima localizzare il sistema dei tipi, come il discorso in generale che le storie sono locali, deve tenere conto della localizzazione del codice, se riesce a tipare alla locazione l che il servizio e ha tipo tau producendo un effetto H , allora possiamo derivare che nell'ambiente dei tipi gamma, localizzando l'effetto H ad l, che è esattamente l'endpoint dove il servizio è stato pubblicato, riusciamo a tipare e con tipo tau, questa è una regola che ci mette in relazione la pubblicazione del servizio con la localizzazione del servizio stesso. A questo punto esaminiamo le regole locali supponendo di essere in un endpoint l e vogliamo assegnare un tipo effetto tenendo conto della location dove stiamo effettuando il controllo dei tipi. Vediamo le costanti, la costante simbolica che abbiamo usato per indicare unit, ovvero un valore distinto che serve per diversi scopi, qualunque sia la locazione, non produce alcun effetto latente, è epsilon. Il tipo associato a un azione alfa che ha a che vedere con una risorsa di sicurezza ha anche essa tipo unit qualunque sia la locazione dove viene eseguito e produce l'effetto latente alfa.

Per andare a vedere qual'è il tipo associato ad una variabile dobbiamo fare un operazione di lookup nell'ambiente vedere quale è il tipo associato nell'ambiente dei tipi gamma alla x, e ovviamente non abbiamo alcun effetto latente.

$$\frac{\Gamma, H \vdash_{\ell} e : \tau}{\Gamma, \ell : H \vdash e : \tau} \quad \text{if } e \text{ is published at } \ell$$

$$\Gamma, \varepsilon \vdash_{\ell} * : 1 \quad \Gamma, \alpha \vdash_{\ell} \alpha : 1 \quad \Gamma, \varepsilon \vdash_{\ell} x : \Gamma(x)$$

LE REGOLE DI PUBBLICAZIONE E UNIT SONO IMMEDIATE

Andiamo a vedere la regola di definizione di una lambda, quindi di un astrazione funzionale. Quelle per le funzioni ricorsive sono identiche ma non mostrate. Il nostro obiettivo è quand'è che riusciamo a tipare ad λ un'astrazione lambda di una funzione con nome z , parametro formale x e corpo e vogliamo comprendere quand'è che possiamo dare a questa astrazione funzionale il tipo $\tau \rightarrow \tau'$, per indicare che prende un argomento di tipo τ , produce un argomento di tipo τ' , con l'effetto della sua esecuzione. Mettiamo sopra la freccia che definisce l'astrazione funzionale l'effetto latente dell'esecuzione delle funzioni. Ovviamente la dichiarazione di un astrazione funzionale non ha alcun effetto latente è la sua esecuzione che la ha, questo è il motivo per cui l'effetto latente dell'assegnamento di tipo è epsilon a livello basso. Per poter fare questa operazione di assegnamento di questo tipo a questa astrazione funzionale, dobbiamo innanzitutto estendere l'ambiente del tipo con il legame tra il nome del parametro formale x e il tipo del parametro formale che è τ , estendiamo gamma con l'associazione $x : \tau$. Dobbiamo estendere anche l'ambiente del tipo con il legame tra il nome della funzione z e il suo tipo. Il suo tipo è quello che stiamo cercando di derivare, quindi è una funzione che prende un argomento di tipo τ , produce un risultato di tipo τ' con effetto latente H e mettiamo questa associazione con Z nell'ambiente. Una volta fatta questa associazione dobbiamo andare a vedere se nell'ambiente esteso, nell'ambiente abbiamo gamma, il legame nome parametro formale tipo del parametro formale, nome della funzione ricorsiva tipo della funzione ricorsiva, dobbiamo andare a vedere se in quell'ambiente dei tipi andando a tipare il corpo otteniamo un tipo τ' che è esattamente il tipo del risultato producendo come effetto latente H cioè l'astrazione dell'esecuzione ovvero H ci definisce esattamente l'astrazione simbolica dell'esecuzione della funzione che quando riceve un argomento attuale di tipo τ produce come risultato un valore di tipo τ' .

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

Se noi avessimo funzioni non ricorsive anonime, quindi senza associare alla funzione alcun nome ovviamente ci bastava soltanto estendere l'ambiente dei tipi con il nome del parametro formale e il suo valore τ e poi andare a tipare il corpo della funzione. Questa è la regola di tipo per l'astrazione.

Adesso andiamo a vedere l'applicazione funzionale e dobbiamo andare a vedere come è fatto il tipo dell'applicazione di e ad e' e dobbiamo anche vedere come derivare l'effetto latente dell'applicazione. Andiamo a leggere le premesse. La premessa è che abbiamo l'espressione e che corrisponde alla funzione che deve essere applicata, quindi siamo in un contesto di programmazione funzionale, le funzioni possono essere ottenute anche come risultato della valutazione di un'espressione quindi dobbiamo vedere il tipo di e e per essere corretto deve essere una funzione. Le premesse della regola mi dicono: supponiamo di essere in un ambiente dei tipi, e dover valutare l'espressione e che è esattamente l'espressione che è la parte sinistra dell'applicazione, quindi l'argomento che dovrà essere il tipo funzionale, quindi vuol dire che nell'ambiente dei tipi localizzata ad H , dato che stiamo definendo la localizzazione della funzione, dobbiamo assegnare il tipo ad e , deve essere una funzione che producendo un risultato τ' avendo come parametro di ingresso un valore di tipo τ e producendo un effetto latente H'' . Questo vuol dire che nell'ambiente dei tipi gamma, sapendo qual'è l'effetto H che corrisponde alla valutazione dell'espressione e non riusciamo a tipare l'espressione e con un tipo funzionale con un opportuno effetto latente. Poi andiamo a verificare se l'espressione e' che corrisponderà all'espressione parametrazione attuale con tipo τ' che è il tipo richiesto dall'astrazione funzionale, esattamente il tipo sulla parte sinistra delle frecce, e questo produce un effetto latente H' .

$$\frac{\Gamma, H \vdash_{\ell} e : \tau \xrightarrow{H''} \tau' \quad \Gamma, H' \vdash_{\ell} e' : \tau}{\Gamma, H \cdot H' \cdot H'' \vdash_{\ell} e e' : \tau'}$$

La valutazione dell'espressione e produce un effetto latente H , che ha un tipo funzionale dove il suo effetto latente è H'' , quindi a questo punto qual'è l'effetto latente che riusciamo a derivare dall'applicazione $e e'$. L'effetto latente che riusciamo a derivare è dato da: prima valutiamo l'espressione e e otteniamo l'effetto latente dell'espressione e , e la parte della premessa a sinistra ci dice questo. Poi andiamo a valutare il parametro attuale che è l'espressione e' il suo effetto latente è H' . Poi l'effetto H'' corrisponde all'effetto latente quando la funzione denotata dall'espressione e opera su un valore di tipo τ , H'' è esattamente l'effetto latente che abbiamo eseguendo l'astrazione funzionale con argomento di tipo τ , quindi l'effetto latente dell'applicazione è dato dalla composizione di H con H' con H'' . L'effetto latente della valutazione dell'espressione che poi produrrà una funzione, se è esattamente la funzione sarà epsilon, se invece è un'espressione più complessa in cui dobbiamo fare dei calcoli, sarà l'effetto ottenuto. H' è l'effetto di e e H'' è l'astrazione dell'esecuzione del comportamento della funzione su un valore di tipo τ che produce un valore di tipo τ' .

Supponiamo di volere affrontare il problema di voler associare un tipo a un blocco di sicurezza, dove e è all'interno di un blocco che deve rispettare la politica PHI. H è l'effetto latente della valutazione dell'espressione e, a cui abbiamo tolto il frame di sicurezza e riusciamo a dire che questo ha tipo tau producendo quindi H come effetto latente, allora l'effetto latente della valutazione del tipo di PHI e è esattamente PHI di H.

$$\Gamma, H \vdash_{\ell} e : \tau$$

$$\Gamma, \varphi[H] \vdash_{\ell} \varphi[e] : \tau$$

L'ifthenelse adotta la tipica strategia dei linguaggi di programmazione funzionale. La scelta che viene fatta è in analogia con la regola dei tipi di ML è che sia il ramo then che il ramo else devono avere lo stesso tipo. Il ramo then ha tipo tau ma produce un effetto latente H, il ramo else ha tipo tau e produce identico effetto latente H.

$$\frac{\Gamma, H \vdash_{\ell} e : \tau \quad \Gamma, H \vdash_{\ell} e' : \tau}{\Gamma, H \vdash_{\ell} \text{if } b \text{ then } e \text{ else } e' : \tau}$$

Questa regola ci dice che supponendo di essere riusciti a tipare ad Γ quell'espressione e con effetto latente H e risultato τ , riusciamo anche a dedurre che se aggiungiamo anche altri effetti latenti, la somma dell'effetto latente corrisponde a unire dei comportamenti, riusciamo a tipare la stessa cosa, ovvero è un astrazione e facciamo esattamente l'indebolimento, aggiungiamo comportamenti in più di quelli aspettati.

$$\Gamma, H \vdash_{\ell} e : \tau$$

$$\Gamma, H + H' \vdash_{\ell} e : \tau$$

Del blocco del planning ce ne scordiamo perchè se dobbiamo andare a vedere cosa fa in un blocco di pianificazione nuova, semplicemente andiamo a vedere il codice del blocco.

$$\Gamma, H \vdash_{\ell} e : \tau$$

$$\Gamma, H \vdash_{\ell} \{e\} : \tau$$

La regola dice che noi vogliamo capire quale è il tipo che dobbiamo associare ad una richiesta rhò, noi facevamo le richieste definendo le proprietà che vuole avere il servizio che andiamo a individuare e andiamo a dare come risultato tau. Come effetto latente della chiamata abbiamo epsilon, ovvero non facciamo alcuna azione osservabile sulla sicurezza. Andiamo ad esaminare passo per passo quali sono le condizioni della premessa.

$$\frac{\tau = \mathbb{U}\{\rho \boxplus_{r[\ell']} \tau' \mid \emptyset, \varepsilon \vdash_{\ell'} e : \tau' \quad \ell \prec \ell' \langle e : \tau' \rangle \quad \rho \approx \tau'\}}{\Gamma, \varepsilon \vdash_{\ell} \mathbf{req}_r \rho : \tau}$$

Noi andiamo a tipare l'ambiente vuoto, quindi vuol dire che non facciamo assunzioni sull'informazione di tipo, quindi vuol dire che l'espressione e è un servizio e tutti i binding dei nomi sono caratterizzati proprio dalle dichiarazioni locali di e, non dobbiamo avere altre informazioni. Andiamo a tipare il corpo del servizio, dato che stiamo facendo una richiesta del servizio e siamo nell'ipotesi che questo servizio abbia locazione l', andiamo a tipare ad l' l'espressione e che ci darà un tipo tau'. Siamo in l, quali sono gli endpoint da prendere, dobbiamo prendere tutti gli endpoint l' che hanno servizi pubblicati e visibili da l.

$$\begin{array}{c}
 \text{Type of the service} \\
 \text{certification} \\
 \downarrow \\
 \tau = \mathbb{U}\{\rho \boxplus_{r[\ell']} \tau' \mid \emptyset, \varepsilon \vdash_{\ell'} e : \tau' \quad \ell \prec \ell' \langle e : \tau' \rangle \quad \rho \approx \tau'\} \\
 \hline
 \Gamma, \varepsilon \vdash_{\ell} \mathbf{req}_r \rho : \tau
 \end{array}$$

$$\begin{array}{c}
 \text{Type of the service} \\
 \text{certification} \\
 \downarrow \\
 \tau = \mathbb{U}\{\rho \boxplus_{r[\ell']} \tau' \mid \emptyset, \varepsilon \vdash_{\ell'} e : \tau' \quad \ell \prec \ell' \langle e : \tau' \rangle \quad \rho \approx \tau'\} \\
 \hline
 \Gamma, \varepsilon \vdash_{\ell} \mathbf{req}_r \rho : \tau
 \end{array}$$

Compatibility
Type of request
Type of the service

Andiamo a vedere quali servizi sono pubblicati e raggiungibili da dove siamo come cliente e quali sono i servizi pubblicati. Questo vuol dire che andiamo a tipare un certo numero di servizi pubblicati, perchè l'approccio di una invocazione di proprietà deve esaminare tutti i servizi che sono pubblicati e deve andare a esaminare le proprietà di tutti i servizi che sono pubblicati. Infatti tra i servizi pubblicati con tipo tau' andiamo a prendere quelli compatibili con la richiesta del servizio. Vuol dire che la richiesta del servizio rho la potete immaginare come un tipo con delle variabili, diamo una nozione di compatibilità, con la compatibilità andiamo a vedere il tipo degli argomenti e la struttura del tipo. In questo modo abbiamo preso un certo insieme di tipi.

**Combine request
with the type of
the service**

$$\tau = \bigcup \{ \rho \boxplus_{r[\ell']} \tau' \mid \emptyset, \varepsilon \vdash_{\ell'} e : \tau' \quad \ell \prec \ell' \langle e : \tau' \rangle \quad \rho \approx \tau' \}$$

**Type of the service
certification**

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

Compatibility
Type of request
Type of the service

Dobbiamo combinare per ogni possibile endpoint l' dobbiamo combinare il tipo della richiesta con il tipo del servizio, cioè l'operatore quadrato caratterizzato dal fatto che abbiamo risolto la richiesta di servizio ad l' andando a certificare il tipo del servizio pubblicato ad l' con tau', la cosa che fa è fare il matching tra il tipo delle richieste caratterizzato da rhò e il tipo effettivo del servizio pubblicato e questo lo fa per tutti i tipi pubblicati. Qui come si vede abbiamo un insieme di tipi, quindi andiamo a prendere tutti i tipi pubblicati visibili da noi, tutte le locazioni degli endpoint dove sono, andiamo a fare il matching tra il tipo della richiesta e il tipo effettivo del servizio pubblicato, a questo punto abbiamo un insieme di tipi che possono contenere delle variabili al loro interno. Il tipo tau è quello che unifica ed è quello più generale che derivo tenendo conto tutti quelli possibili che mi esaudiscono la richiesta del servizio.

Combine request with the type of the service \downarrow
 $\tau = \mathbb{U}\{\rho \boxplus_{r[\ell']} \tau' \mid \emptyset, \varepsilon \vdash_{\ell'} e : \tau' \quad \ell \prec \ell' \langle e : \tau' \rangle \quad \rho \approx \tau'\}$

Type of the service certification \downarrow
 $\Gamma, \varepsilon \vdash_{\ell} \text{req}_r \rho : \tau$

Compatibility
Type of request
Type of the service \downarrow

$\tau = \text{unit} (1)$

Andiamo a vedere un esempio e partiamo dall'if. Vogliamo vedere quale è il tipo associato a un'espressione booleana supponendo di essere nell'ambiente dei tipi vuoto, effetto epsilon e questa espressione ifthenelse è caratterizzata dal fatto che il ramo else prende una funzione z e prende un argomento x e fa α , e il ramo else fa la stessa cosa strutturalmente solo che l'effetto è caratterizzato dall'operazione α' . Noi vogliamo andare a vedere se questa funzione è tipata $\tau \rightarrow \tau$ con $\alpha + \alpha'$. Quello che fa questo if è produrre in ogni caso una funzione da tipo $\text{unit} \rightarrow \text{unit}$, perché il corpo delle 2 funzioni non ha bisogno di un valore ed esegue un'azione di sicurezza. Quindi una funzione da unit a unit , adesso in un caso è α nell'altro α' . Per potergli dare un tipo dobbiamo fare in modo che il ramo then e il ramo else abbiano lo stesso tipo.

$$\emptyset, \varepsilon \vdash \text{if } b \text{ then } \lambda_z x. \alpha \text{ else } \lambda_z x. \alpha' : \tau \xrightarrow{\alpha + \alpha'} \tau$$

$\tau = \text{unit} (1)$

$$\frac{\Gamma, H \vdash_{\ell} e : \tau \quad \Gamma, H \vdash_{\ell} e' : \tau}{\Gamma, H \vdash_{\ell} \text{if } b \text{ then } e \text{ else } e' : \tau}$$

La regola del tipo ci dice che per tipare correttamente con alfa + alfa questa espressione sia il ramo then che else devono avere lo stesso tipo, tau -> tau, con effetto latente alfa + alfa'.

$$\frac{\emptyset, \varepsilon \vdash \lambda_z x. \alpha : \tau \xrightarrow{\alpha + \alpha'} \tau \quad \emptyset, \varepsilon \vdash \lambda_z x. \alpha' : \tau \xrightarrow{\alpha' + \alpha} \tau}{\emptyset, \varepsilon \vdash \text{if } b \text{ then } \lambda_z x. \alpha \text{ else } \lambda_z x. \alpha' : \tau \xrightarrow{\alpha + \alpha'} \tau}$$

Per fare questa operazione di tipo assumendo che in gamma andiamo a mettere il legame tra il nome della variabile della funzione e il suo tipo ovvero la funzione che va da tau a tau e con effetto latente alfa + alfa' e nell'ambiente gamma andiamo a mettere il legame del tipo tra la variabile che corrisponde al parametro formale e il tipo tau, quello che dobbiamo fare applicando la regola dell'astrazione funzionale dice che se non riusciamo a tipare il corpo della funzione assumendo questo legame per il parametro e questo legame per l'astrazione funzionale e riusciamo a derivare che in questo contesto H è esattamente l'effetto latente che ci permette di tipare con tau' il corpo della funzione, allora abbiamo tipato correttamente la funzione

$$\tau = \text{unit} (1)$$

$$\Gamma = \{ z : \tau \xrightarrow{\alpha + \alpha'} \tau; x : \tau \}$$

$$\frac{\Gamma, H \vdash_{\ell} e : \tau}{\Gamma, H + H' \vdash_{\ell} e : \tau}$$

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

$$\frac{\Gamma, \alpha \vdash \alpha : \tau}{\Gamma, \alpha + \alpha' \vdash \alpha : \tau}$$

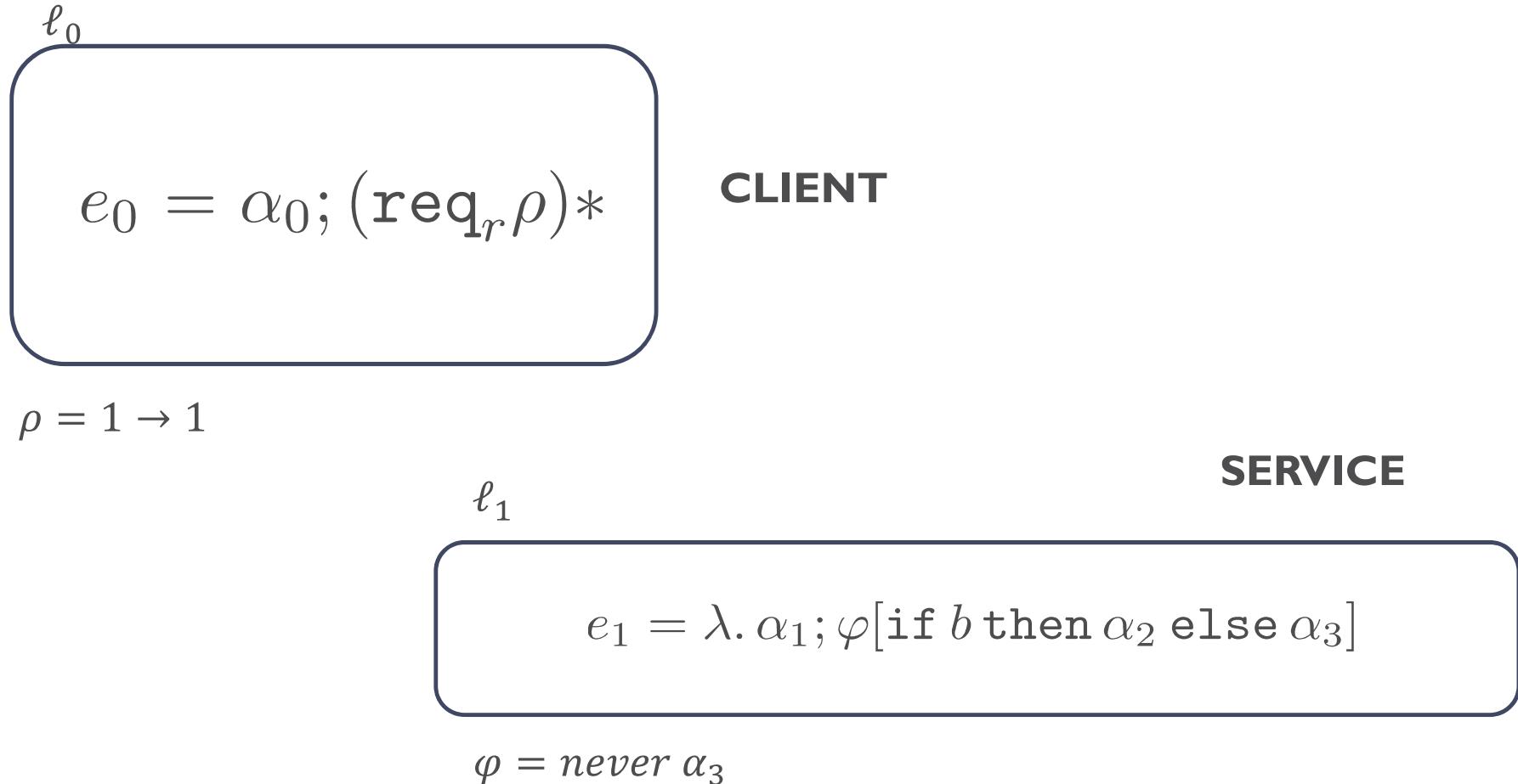
$$\frac{\emptyset, \varepsilon \vdash \lambda_z x. \alpha : \tau \xrightarrow{\alpha + \alpha'} \tau}{\emptyset, \varepsilon \vdash \text{if } b \text{ then } \lambda_z x. \alpha \text{ else } \lambda_z x. \alpha' : \tau \xrightarrow{\alpha + \alpha'} \tau}$$

$$\frac{\Gamma, \alpha' \vdash \alpha' : \tau}{\Gamma, \alpha' + \alpha \vdash \alpha' : \tau}$$

$$\frac{\emptyset, \varepsilon \vdash \lambda_z x. \alpha' : \tau \xrightarrow{\alpha' + \alpha} \tau}{\emptyset, \varepsilon \vdash \text{if } b \text{ then } \lambda_z x. \alpha \text{ else } \lambda_z x. \alpha' : \tau \xrightarrow{\alpha + \alpha'} \tau}$$

Se facciamo questa operazione sul ramo then, per il ramo else è identico. Andiamo ad estendere l'ambiente del tipo con il legame alfa, alfa', dobbiamo dire che alfa ha tipo tau, ma questo lo possiamo ottenere semplicemente andando a tipare alfa di tipo tau che è immediato perché è un assioma applicando il weaklein con le somme. Questo qui che ora stiamo vedendo è l'albero di prova che dice che quello è il tipo più generale da associare a questa espressione

Andiamo a vedere un esempio nel contesto distribuito e supponiamo di avere questo cliente, che fa una richiesta di avere un servizio che fa unit, da unit a unit e gli da come valore una funzione unit e poi esegue l'azione alfa0. Il servizio è semplicemente una funzione senza parametro che esegue alfa1 e poi esegue un ifthenelse, assumendo il valore della condizione booleana che sia vero o falso, eseguo alfa2 o alfa3 a seconda del valore della condizione booleana. Il servizio è localizzato ad l1 ed è sottoposto dato che abbiamo un blocco di sicurezza interno, alla politica di sicurezza che dice che l'azione alfa3 non deve essere mai eseguita, quindi questa è la struttura.



Questo è un comportamento applicando le regole della semantica operazionale che abbiamo visto, è un comportamento ammissibile, in realtà è il comportamento che corrisponde al fatto che la guardia dell'ifthenelse è true. A questo punto mettiamo in esecuzione in parallelo il cliente con il suo servizio, supponiamo nel plan pigreco0 la richiesta l sia risolta ad l1. La richiesta viene esaudita ad l1, eseguiamo le operazioni relative

$$\begin{aligned}
 & \ell_0 : \pi_0 \triangleright \varepsilon, e_0 \parallel \ell_1 : 0 \triangleright \varepsilon, * \\
 \rightarrow & \ell_0 : \pi_0 \triangleright \alpha_0, \text{req}_r \rho * \parallel \ell_1 : 0 \triangleright \varepsilon, * \\
 \rightarrow & \ell_0 : \pi_0 \triangleright \alpha_0, \text{wait } \ell_1 \parallel \ell_1 : 0 \triangleright \sigma, e_1 * \\
 \rightarrow & \ell_0 : \pi_0 \triangleright \alpha_0, \text{wait } \ell_1 \parallel \ell_1 : 0 \triangleright \sigma \alpha_1, \varphi[\text{if } \dots] \\
 \rightarrow & \ell_0 : \pi_0 \triangleright \alpha_0, \text{wait } \ell_1 \parallel \ell_1 : 0 \triangleright \sigma \alpha_1, \varphi[\alpha_2] \\
 \rightarrow & \ell_0 : \pi_0 \triangleright \alpha_0, \text{wait } \ell_1 \parallel \ell_1 : 0 \triangleright \sigma \alpha_1 \alpha_2, \varphi[*] \\
 \rightarrow & \ell_0 : \pi_0 \triangleright \alpha_0, \text{wait } \ell_1 \parallel \ell_1 : 0 \triangleright \sigma \alpha_1 \alpha_2, * \\
 \rightarrow & \ell_0 : \pi_0 \triangleright \alpha_0, * \parallel \ell_1 : 0 \triangleright \varepsilon, *
 \end{aligned}$$

Facciamo un sign che poi dovrà fare alfa1, va in fondo, do il risultato e se il risultato che abbiamo assunto che il valore della guardia è true l'unica cosa che verrà eseguita è alfa1 e quindi poi alla fine ritorniamo il risultato al cliente nel modo che uno si aspetta.

ℓ_0

$$e_0 = \alpha_0; (\text{req}_r \rho)^*$$

$$\rho = 1 \rightarrow 1$$

 ℓ_1

$$e_1 = \lambda. \alpha_1; \varphi[\text{if } b \text{ then } \alpha_2 \text{ else } \alpha_3]$$

$$\varphi = \text{never } \alpha_3$$

**The statefull history
extracted from e_0**

$$\ell_0 : \alpha_0 \cdot \{ r[\ell_1] \triangleright \ell_1 : \sigma \cdot \alpha_1 \cdot \varphi[\alpha_2 + \alpha_3] \}$$

**The stateless
history**

$$(\ell_0 : \{\alpha_0\}, \ell_1 : \{\alpha_1[\varphi\alpha_2]_\varphi, \alpha_1[\varphi\alpha_3]_\varphi\})$$

Se noi andiamo ad applicare l'astrazione del comportamento dunque quello che abbiamo chiamato la storia stateful e la storia stateless, quello che noi andiamo a vedere andando a vedere il sistema dei tipi associato è che quella statefull vede il fatto che abbiamo relativamente alla struttura di ℓ_0 abbiamo una storia che mi dice prima di eseguire l'operazione alfa0 ad ℓ_0 poi ho un operazione di plan selection che mi viene data dall'orchestratore che mi dice che la richiesta di servizio I è eseguita ad ℓ_1 , ad ℓ_1 avrò l'attivazione sigma, l'attivazione di alfa1 e poi alfa2 + alfa3 sottoposta alla politica PHI. Ovviamente questa è quello che abbiamo chiamato stateful perchè si vede l'attivazione. Se poi facciamo quella stateless ci dice che noi ad ℓ_0 eseguiamo alfa0, ad ℓ_1 andiamo a veder alfa1, l'apertura della politica alfa2 PHI. Se noi andiamo a vedere astrattamente la storia stateless ha 2 comportamenti ad ℓ_1 , perchè tiene conto di tutti i possibili comportamenti senza tener conto della validità, ma in realtà l'unico comportamento ammissibile è quello precedente. Il ramo else non verrà mai eseguito perchè viola la politica.

Se abbiamo una rete di servizi, se riusciamo a tipare correttamente ad ogni endpoint i servizi pubblicati localmente a quell'endpoint, se abbiamo che la storia Hi che è associata al servizio alla locazione i è valida rispetto a una selezione pigreco_i allora a quel punto quella è una esecuzione ammissibile. Nel nostro caso l'unica storia viable è quella che ha come plan selection quella di $r[1]$ perchè l'altra storia che corrisponderebbe alla plan selection di I_1 che mi manda quello e contiene soltanto alfa1 alfa2 perchè l'altra non è viable perchè non viene verificata la politica di sicurezza, quindi vuol dire che abbiamo un modo per discriminare esattamente le storie che corrispondono poi a comportamenti a tempo di esecuzione.

SAFETY

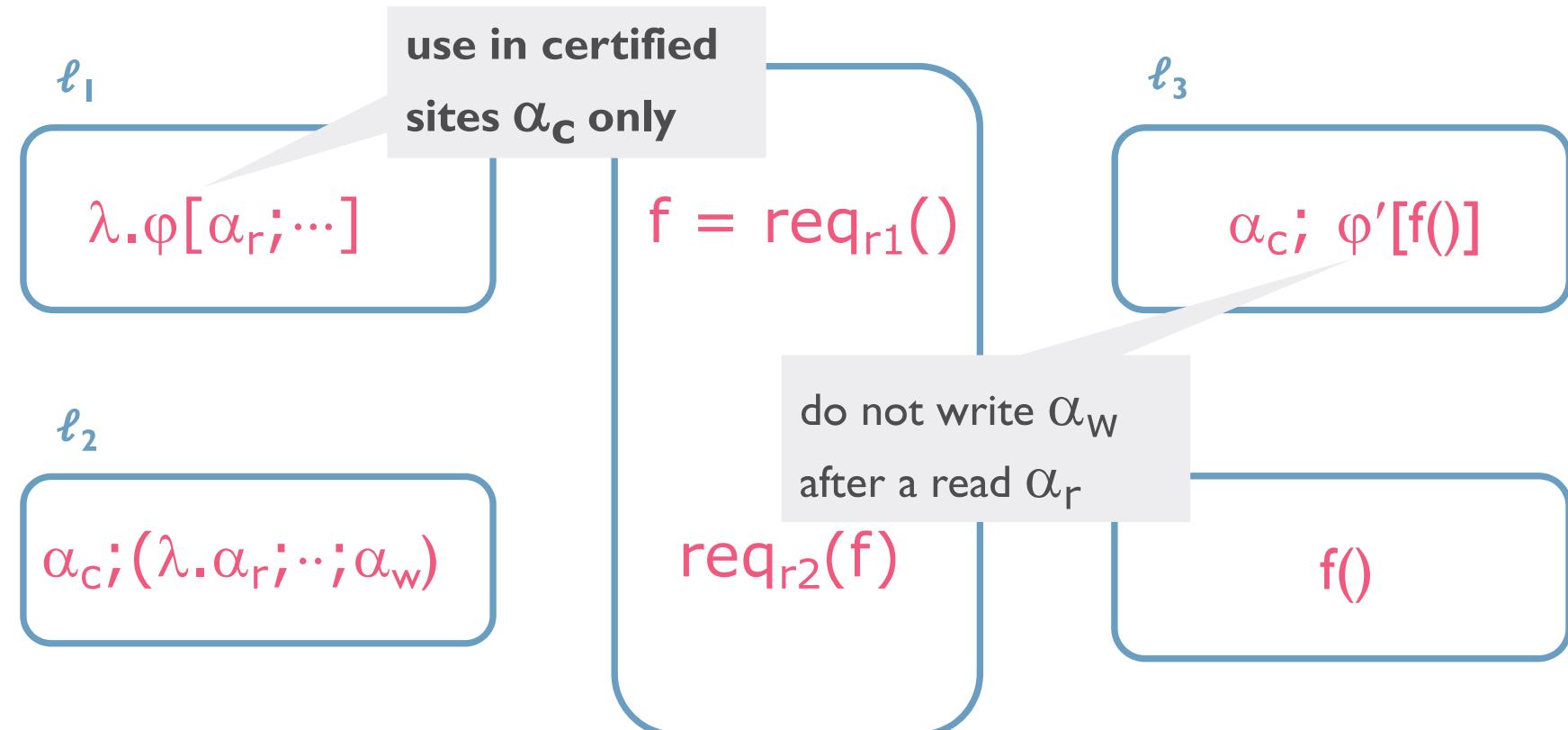
$\{\ell_i \langle e_i : \tau_i \rangle\}_{i \in I}$ **network of services**

$\emptyset, H_i \vdash e_i : \tau_i$ for all $i \in I$

if H_i is π_i valid then π_i is viable for e_i at ℓ_i

Avevamo 2 fornitori di servizio che davano le funzioni e sulla destra 2 fornitori di servizio che davano la possibilità di eseguire le funzioni. Avevamo un cliente che faceva 2 richieste, r1 che chiedeva una funzione e una r2 che voleva mandare in esecuzione una funzione. Il primo fornitore di servizi dava una funzione che era sottoposta a un framing di sicurezza che diceva che questa funzione doveva essere eseguita soltanto in siti certificati, mentre l'altro lato di fornitore di servizio che accettava in esecuzione delle funzioni, sottoponeva delle funzioni che venivano inviate dal cliente che diceva noWaR.

BACK TO OUR FIRST EXAMPLE



Se andiamo ad applicare il sistema dei tipi e andiamo a vedere quali sono astrattamente le storie associate a quell'orchestrazione dei servizi viene fuori questa espressione: se scelgo r1 la posso mandare ad l1 e la certifico, etc.. Dato che le storie sono innestate perchè ad esempio quando vado a prendere la funzione da l1 e poi la eseguo da l3, nella storia del cliente ho che prima faccio una richiesta esaudita da l1 poi la faccio ad l3, questo vuol dire che la planned selection da l1 poi al suo interno ha la planned selection di l3 perchè ad l3 arrivo soltanto dopo aver fatto la richiesta del servizio, quindi vuol dire che noi abbiamo delle storie che deriviamo dal sistema dei tipi, inferiamo gli effetti che possono avere delle planned selection innestate. Chiaramente in questa forma è difficile comprendere quali tra tutte le storie possibili sono ammissibili, perchè abbiamo detto pocanzi che il sistema è safety, quindi le storie che sono ammissibili sono esattamente catturate dal sistema dei tipi poi però vogliamo vedere tra quelle ammissibili quelle che corrispondono all'orchestrazione quelle che abbiamo chiamato viable ed è difficile dirlo in questa forma quali sono le viable, in realtà se noi facessimo una trasformazione e questa è una trasformazione guidata dall'idea intuitiva, ovvero se vado a scegliere prima la funzione dal servizio l1 e poi la mettiamo in esecuzione al servizio l3, se vado a prendere come piano di esecuzione quello che prima mi dice di scegliere la richiesta di servizio r1 e la risolvi ad l1 e poi risolvi la richiesta r2 ad l2, abbiamo esattamente un comportamento ammissibile perchè certifichiamo e poi andiamo a vedere che la politica PHI di certificazione vale con l'esecuzione e che non viola la politica PHI'

WHICH ARE THE VIABLE PLANS ?

$$\begin{aligned} & \{ r_1[\ell_1] \triangleright \ell_1: \varepsilon, r_1[\ell_2] \triangleright \ell_2: \alpha_c \} \cdot \\ & \{ r_2[\ell_3] \triangleright \ell_3: \alpha_c \cdot \varphi'[\{ r_1[\ell_1] \triangleright \varphi[\alpha_r], r_1[\ell_2] \triangleright \alpha_r \cdot \alpha_w \}], \\ & \quad r_2[\ell_4] \triangleright \ell_4: \{ r_1[\ell_1] \triangleright \varphi[\alpha_r], r_1[\ell_2] \triangleright \alpha_r \cdot \alpha_w \} \} \end{aligned}$$

Difficult to tell: the planned selections are nested!

$\{ r_1[\ell_1] \mid r_2[\ell_3] \triangleright \ell_3: \alpha_c \cdot \varphi'[\varphi[\alpha_r]] \},$	viable
$\{ r_1[\ell_2] \mid r_2[\ell_4] \triangleright \ell_2: \alpha_c, \ell_4: \alpha_r \cdot \alpha_w, \}$	viable
$\{ r_1[\ell_1] \mid r_2[\ell_4] \triangleright \ell_4: \varphi[\alpha_r], \}$	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

Riusciamo a trasformare la storia derivata dal sistema dei tipi in una storia equivalente che semanticamente rappresenta la stessa forma al punto che H' ha quella struttura che dicevamo ovvero tutte le scelte di selezione sono fatte in testa quindi non sono innestate una dentro l'altra, questo concettualmente riuscire a trasformare l'astrazione di un comportamento in una forma dove l'orchestratore fa le scelte all'inizio dei tempi e poi eseguiamo? Potremmo così ottenere una forma più utile da essere manipolata

LINEARIZATION

- transform H into an 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

E' possibile introdurre una nozione di equivalenza abbastanza semplice sulle storie. Questo insieme di equazioni ci dice che una storia H posso vederla come la planned selection vuota, Se io la sequenzializzazione di una planned selection per un certo insieme di indici e poi con un $\pi_{i,j}$ con $H_{i,j}$ per un altro insieme di indici, quello da fare è mettere a fattor comune tutte le scelte e poi sotto sequenzializzo H_i con H'_j quindi butto sopra tutte le scelte e poi eseguo i comportamenti. Le somme sono gestite allo stesso modo , la politica PHI controlla le azioni non le scelte dell'endpoint quindi posso portar fuori la scelta dell'endpoint. Orientando da sinistra verso destra queste equazioni notate che tutte le componenti della parte destra hanno esattamente la forma che ci aspettiamo, e questo è un modo di fare handwaving ma per dire che l'operazione che volevamo ovvero di trasformare le storie in una funzione in cui l'orchestrazione ha messo a top si può fare

LINEARIZATION

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

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

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

$$\phi[\{\pi_i \triangleright H_i\}_i] \equiv \{\pi_i \triangleright \phi[H_i]\}_i$$

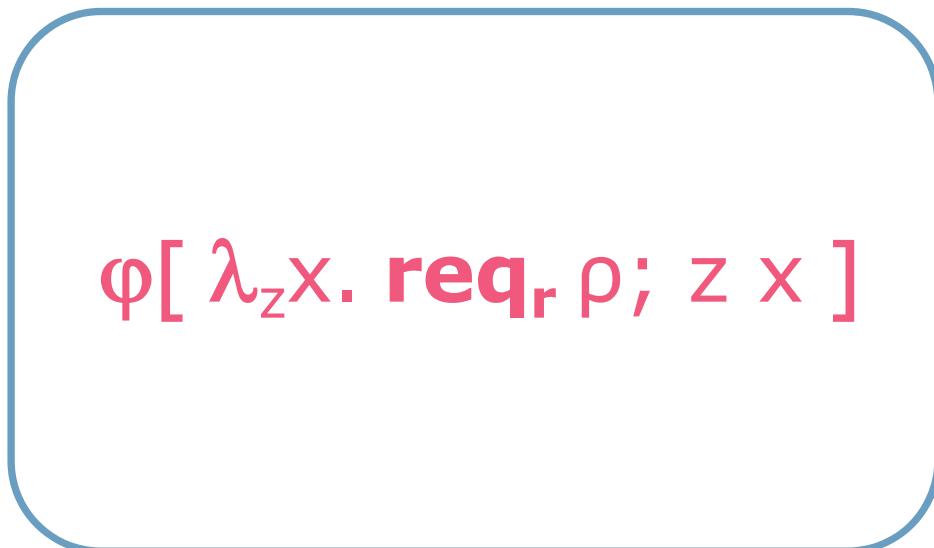
$$\mu h. \{\pi_i \triangleright H_i\}_i \equiv \{\pi_i \triangleright \mu h. H_i\}_i$$

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

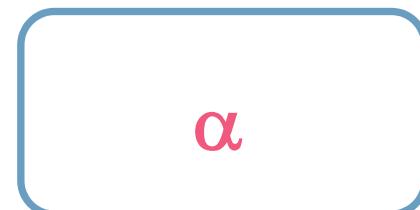
EXAMPLE

Questa è la storia che riesco a derivare di effetto latente relativo a questo codice, ha una politica PHI qualunque essa sia, al suo interno abbiamo una funzione ricorsiva che fa una richiesta rho e poi esegue la funzione zeta relativamente al parametro x. Se risolvo ad l1 alfa, se risolvo ad l2 beta poi torno con H. Quello che noi facciamo è fare una trasformazione perchè...

H



ℓ₁



ℓ₂



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

EXAMPLE

ho il phi e muh fatto in questo modo, il muh lo metto nella forma con la selezione davanti, a questo punto internamente mi faccio tutte le planned selection, le metto tutte assieme e poi le porto fuori e ho esattamente tutte le varie scelte possibili di planned selection dove poi al suo interno o faccio il comportamento alfa ricorsivamente o il comportamento beta che corrisponde a scegliere tra alfa e beta.

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

Abbiamo preso dal sistema dei tipi le storie derivate, le avevamo innestate e abbiam detto: dobbiamo linearizzarle. Poi in realtà uno scopre un'altra cosa, che si potrebbe avere sintatticamente un blocco di sicurezza, ad esempio entro in un blocco di sicurezza che ha la politica phi e al suo interno ho un blocco di sicurezza della stessa politica phi, ovvero potremmo scrivere male dei blocchi di sicurezza . Normalmente i blocchi di sicurezza fatti in questi modo normalmente comanda quello più esterno, quello più interno è ridondante, ribadisco blocchi di sicurezza della stessa politica. A questo punto quello che si riesce a fare è una trasformazione che elimina, una tipica trasformazione che viene fatta nei compilatori dove gli assegnamenti alle costanti all'interno di un ciclo vengono spostate a void, all'interno di un blocco se mi accorgo che ho la stessa struttura di blocco relativa alla funzione di sicurezza che deve essere spostata la sposto fuori al livello più alto e continua ad essere rispettata la politica.

VERIFYING VALIDITY

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

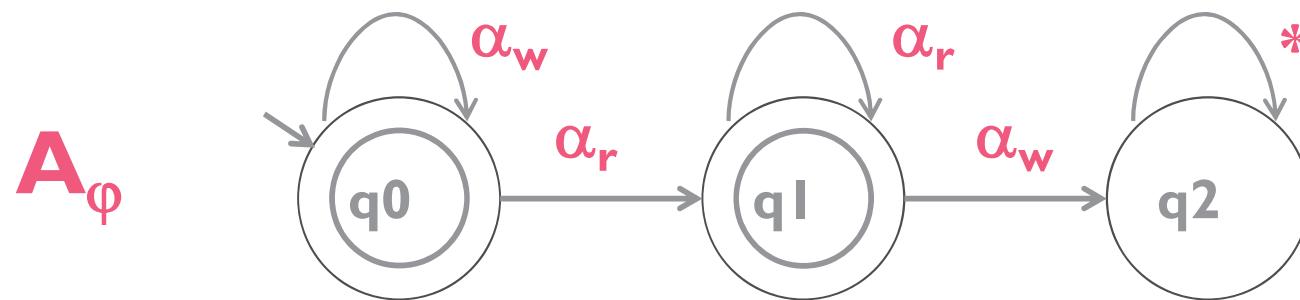
- transform linearized **history expression** into **BPA**s (Basic Process Algebras)
- transform **policies** into **scoped policies**
(in the form of Finite State Automata)

A questo punto facendo anche questa trasformazione ho una forma di storie che non sono lineare non sono ridondanti quindi hanno un'altra proprietà banale che vi risparmio, allora ho una struttura molto vicina a un formalismo già studiato, questo perchè chi fa ricerca alle volte è prigo, nel senso che riusa risultati fatti da altri, quando abbiamo affrontato questo problema ci siamo chiesti come avremmo potuto ottenere questa struttura di forme lineari non ridondanti andare a vedere le proprietà di quelle che sono le orchestrazioni ammissibili che rispettano le politiche di sicurezza? Sono parenti ai BPA che sono dei grafi regolari dove il problema di andare a verificare delle proprietà logiche su questi grafi regolari è un problema noto e decidibile. Anzichè riscoprire la ruota abbiamo fatto vedere come si riusciva a mappare il nostro formalismo con quello derivato dopo tutte queste trasformazioni in un BPA.

Adesso dobbiamo fare in modo di rappresentare le nostre politiche come delle formule logiche. Se noi riuscissimo a rappresentare le nostre politiche come forme logiche avremmo chiuso il cerchio. Riusciamo a trasformare questo automa in un meccanismo dove si fa vedere espressamente che stiamo all'interno della politica di sicurezza? Si lo riusciamo a fare in questo modo:

FROM POLICIES TO SCOPED POLICIES

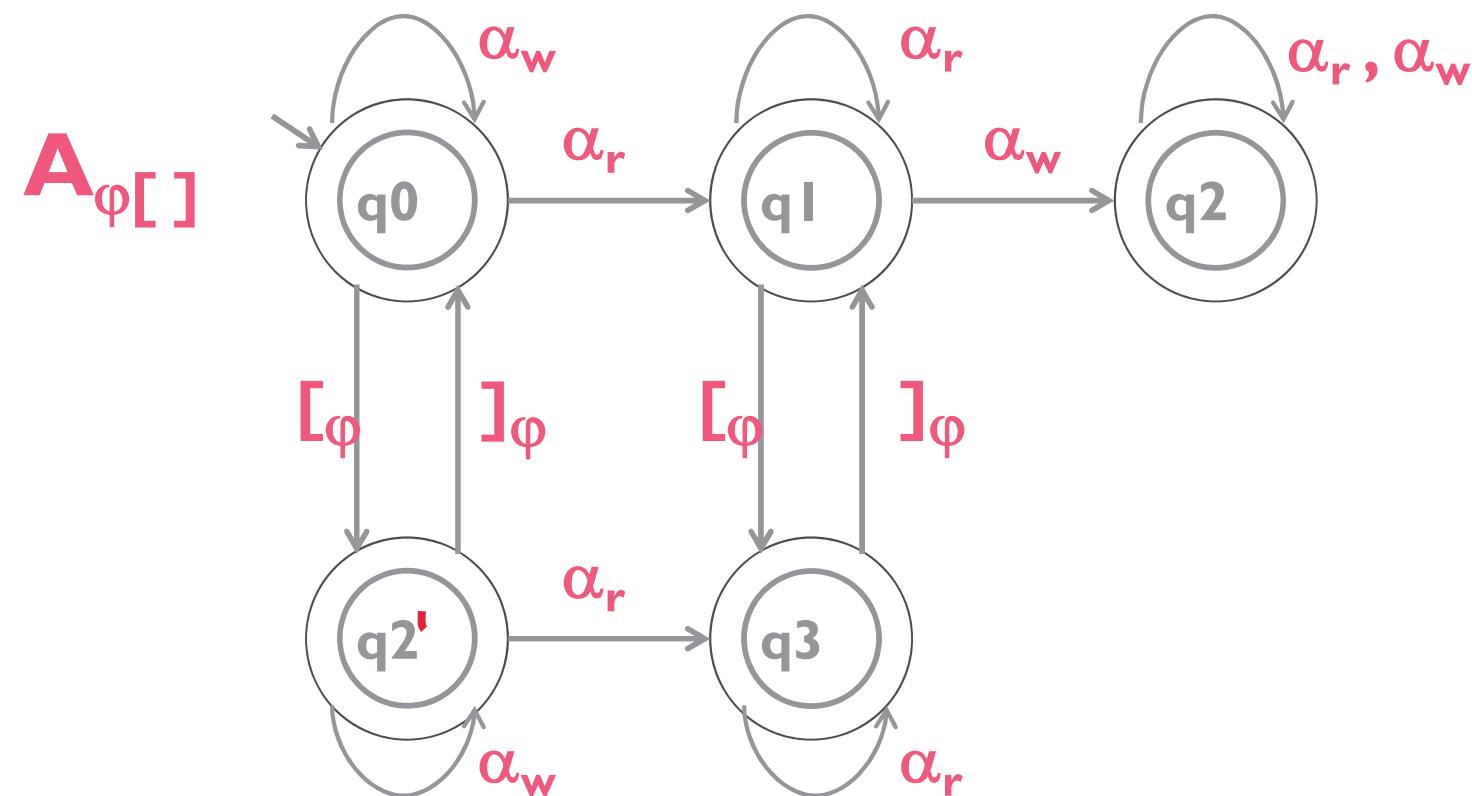
Example



Chinese Wall policy: no write after read

Riusciamo a trasformarlo in un meccanismo dove si fa vedere espressamente che stiamo all'interno della politica di sicurezza? Vi ricordate che avevamo messo negli eventi, degli eventi particolari che corrispondevano alla apertura e chiusura della politica? Bene facciamo questa operazione, vediamo gli eventi come eventi dell'alfabeto degli automi. Se vediamo questo come eventi dell'alfabeto degli automi siamo nello stato q_0 , la parte di sopra è esattamente quello che conosciamo, la parte di sotto è quello che si chiama parte relativa alle politiche con loro scoping. Se introduciamo la politica entriamo in uno stato q_2' che è ancora finale, e qui rimaniamo all'interno della politica e quando usciamo dalla politica e andiamo allo stato sopra vedete a questo punto ritorniamo esattamente come prima. Quindi abbiamo una trasformazione dell'automa che ci permette di dedurre anche quando entriamo e usciamo dalla politica.

FROM POLICIES TO SCOPED POLICIES



Dato che gli automi hanno una rappresentazione come formule logiche che riescono a rappresentare politche di safety, usiamo questa proprietà quindi quello che abbiamo fatto è stato ritrasformare la nostra politica in una forma nota e questa forma nota riusciamo a rappresentare il problema del model checking.

MODEL- CHECKING

- Step 1: Linearization of Histories**
- Step 2: From Linear Histories to Basic Process Algebra**
- Step 3: Automata for Step Policies**
- Step 4: Model checking Basic Process Algebras (BPAs) with Finite State Automata (FSA).**

Il risultato standard dice di prendere la formula che rappresenta la proprietà lineare che vogliamo verificare sui comportamenti, la BPA rappresenta i comportamenti e la neghiamo. Il linguaggio dei comportamenti, l'automa che rappresenta il linguaggio dei comportamenti corretti lo neghiamo e andiamo a vedere l'intersezione. Se l'intersezione è vuota, vuol dire che i comportamenti rispettano la politica, se così non fosse vuol dire che tutti gli elementi che stanno nell'intersezione tra il linguaggio e la negazione della politica, ovvero le proprietà che non sono soddisfatte, c'è un intersezione non vuota quindi vuol dire che esiste almeno un comportamento che non soddisfa la politica. Se noi riusciamo a fare questa operazione in modo decidibile, abbiamo esattamente risolto il problema del model checking e questo è esattamente ciò che succede.

MODEL CHECKING

The standard decision procedure for verifying that a BPA process p satisfies a regular property φ amounts to constructing the pushdown automaton for p and the Buchi automaton for the negation of φ .

The property holds if it is empty the (context-free) language accepted by the intersection of p with $! \varphi$ which is still a pushdown automaton.

This problem is known to be decidable, and several algorithms and tools show this approach feasible

Il teorema che riesce a dimostrare è di riuscire a estrarre tramite queste cose i piani viable e i piani viable sono esattamente gli orchestratori perchè sono esattamente quelli che ho al top delle storie e corrispondono esattamente ad avere un orchestrazione.

MAIN RESULT

Network $N = \ell_1\{e_1:\tau_1\} \parallel \dots \parallel \ell_k\{e_k:\tau_k\}$

$0, H_i \vdash e_i : \tau_i$ 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 \mathbf{H}
- **linearization:** transform \mathbf{H} into an equivalent \mathbf{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

Abbiamo individuato un contesto linguistico per affrontare il problema della composizione di servizio, il contesto linguistico è caratterizzato da avere politiche locali di sicurezza quelle che sono chiamate i security framing, le politiche che sono proprietà regolari del comportamento e poi vogliamo passare da una richiesta per nome con un unico endpoint ad una richiesta di proprietà con multipli endpoint. Possiamo usare le tecnologie tipiche della progettazione dei linguaggi di programmazione all'intera toolchain del problema che abbiamo affrontato? La risposta è che riusciamo a fare un sistema di tipi effetto che ci estrae le storie. A questo punto dalle storie facciamo un po di manipolazioni e le portiamo in una forma che è adatta a fare la verifica in modo particolare via model checking. A questo punto le storie che sono viable ovvero che rispettano la proprietà che il model checker le verifica mi danno il blueprint dell'orchestratore a questo punto possiamo usarlo per guidare l'esecuzione sicura di servizi.

CONCLUSIONS

A linguistic framework for secure service composition

- **Goal: apply Programming Language techniques to formally specify the language tool chain**
- **safety framings, policies, req-by-contract**
- **type & effect system**
- **verification of effects, to extract viable plans to construct the trusted orchestrator**
- **The orchestrator securely composes and runs stateless services**



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 happen”
- **multi-choice and dependent plans**

- Bodei, Dinh, Ferrari: Checking global usage of resources handled with local policies. *Sci. Comput. Program.* 133: 20-50 (2017)
- Pierpaolo Degano, Gian Luigi Ferrari, Gianluca Mezzetti: Regular and context-free nominal traces. *Acta Informatica* 54(4): 399-433 (2017)
- Massimo Bartoletti , Pierpaolo Degano, Gian Luigi Ferrari, Roberto Zunino: Model checking usage policies. *Math. Struct. Comput. Sci.* 25(3): 7
- Bartoletti, Costa, Degano, Martinelli, Zunino, Securing Java with Local Policies, Jornal of Object Technology, 2009
- Bartoletti, Degano, Ferrari: Planning and verifying service composition. *Journal of Computer Security* 17(5): 799-837 (2009)
- Bartoletti, Degano, Ferrari, Zunino: Local policies for resource usage analysis. *ACM Trans. Program. Lang. Syst.* 31(6): 23:1-23:43 (2009)
- Bartoletti, Degano, Ferrari, Zunino: Semantics-Based Design for Secure Web Services. *IEEE Trans. Software Eng.* 34(1): 33-49 (2008)
- Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari, Roberto Zunino: Secure Service Orchestration. *FOSAD* 2007: 24-74