UNIVERSITÀ DEGLI STUDI DI PADOVA
FACOLTÀ DI SS. MM. FF. NN.
CORSO DI LAUREA IN MATEMATICA

TESI DI LAUREA

Analisi statica dello Spi-calcolo

Relatore: Ch.mo Prof.re Gilberto File'

Laureanda: Alessia Ceccato

A.A. 2002-2003

Indice

Elenco delle tabelle

Capitolo 1

Introduzione

Il termine generale analisi statica viene usato per indicare ogni metodo che, partendo dalla specifica del sistema dinamico, riesce ad ottenere staticamente, cioè senza eseguire il sistema, approssimazioni corrette di proprietà del sistema [13, 27]. Praticamente sono state proposte delle analisi statiche per tutti i linguaggi di programmazione interessanti o specifiche di linguaggi e per i CCS (Calculi of Communicating Systems), assieme alle loro estensioni come il π- e lo Spi-calcolo [16, 17, 25, 24, 28, 6, 3]. Poichè sia i CCS [22, 23] che le loro estensioni vengono usate per descrivere protocolli di comunicazione, la loro analisi statica spesso ha come risultato quello di trovare proprietà di sicurezza per questi protocolli [18, 9, 1, 8, 21]. In molti casi l'analisi consiste nell'assegnare dei tipi al sistema [3] e si può dire che il protocollo è sicuro se i tipi assegnati soddisfano determinate ipotesi, in altri casi l'analisi consiste nel tradurre il protocollo in un insieme di regole da sottoporre agli "attacchi" di una spia [3], se tramite tali regole la spia non riesce a derivare dal protocollo informazioni segrete allora si può concludere che il protocollo è sicuro. In questo lavoro ci proponiamo quindi di analizzare i protocolli di comunicazione, utilizzando lo Spi-calcolo come modello per poterli rappresentare. Lo Spi-calcolo estende un particolare modello per la connessione dei sistemi interattivi (il π-calcolo) al quale vengono aggiunte le operazioni comuni nei protocolli di sicurezza. Vi sono molti dialetti dello Spi-calcolo, noi considereremo quello proposto in [3] che utilizza una semantica di tipo "late" [26].

I risultati da noi ottenuti sono i seguenti:

1. Se P è il processo da analizzare, l'analisi proposta computa l'insieme PAIR(P) delle coppie di azioni di input e di output che si possono sincronizzare durante ogni computazione del processo in analisi (si escludono così molte coppie per cui la sincronizzazione è impossibile). Per ogni coppia si controlla se tutti i test precedenti

vengono soddisfatti contemporaneamente ed in caso affermativo si memorizza la sostituzione che avviene per l'azione di sincronizzazione;

2. L'analisi, quando riesce a terminare, si dimostra essere corretta, e la dimostrazione è basata su una nuova nozione di computazione che chiamiamo *computazione con sostituzioni esplicite.*

3. Per simulare tutti i contesti in cui il processo può essere eseguito, si è introdotto un processo "cattivo". Tale processo se composto in parallelo con P funge da peggior contesto possibile;

4. Tramite l'analisi statica, tale processo e delle versioni meno "cattive" di tale processo, si sono trovate proprietà di sicurezza per i protocolli di comunicazione.

Nel capitolo 2 definiamo brevemente il π-calcolo [24]. Tale calcolo è la base per la versione del π-calcolo applicato (Spi-calcolo) che utilizzeremo per modellare i protocolli di comunicazione. Nel capitolo 3 diamo la definizione del calcolo usato nel presente lavoro. Nel capitolo 4 enunciamo la definizione di sicurezza per un protocollo e riportiamo i risultati ottenuti da Martín Abadi e Bruno Blanchet [3] nell'analisi di protocolli di comunicazione, che risultano utili per dimostrare alcune proprietà del processo che intende simulare tutti i contesti possibili. Nel capitolo 5 elenchiamo una serie di assunzioni che risultano utili per la nostra analisi statica. Nel capitolo 6 presentiamo la nostra analisi statica e le proprietà di sicurezza che grazie alla sua applicazione si riescono ad ottenere.

Capitolo 2

Il π-calcolo

Come in [24] introduciamo il π-calcolo come modello per la mutevole connessione dei sistemi interattivi. Come tale, gioca due ruoli distinti:

- Per prima cosa, modella le reti nel senso moderno di comunicazione, dove i messaggi vengono spediti da sito a sito e possono anche contenere collegamenti a processi attivi. Tali messaggi possono anche consistere di processi che possono venire attivati dal destinatario e che, inoltre, possono avere ulteriori collegamenti;

- Per seconda cosa, il π-calcolo può essere visto come un modello base di computazione. Ogni modello fondamentale si basa su di un piccolo numero di nozioni primitive; il π-calcolo si basa sulla nozione primitiva di *interazione* , come ad esempio le macchine di Turing si basano sulla nozione di lettura e scrittura su celle di memoria.

La capacità di cambiare la connessione di reti di processi è la differenza cruciale tra il π-calcolo e gli automi. Questa proprietà del π-calcolo merita una pausa di riflessione: la possibilità di cambiare le varie connessioni è veramente quello che noi indichiamo con il termine *mobilità*?

Il primo articolo sul π-calcolo scritto nel 1980 e pubblicato nel 1992, venne intitolato *A calculus of mobile processes* [25] e la parola 'mobilità' viene ora usata molto più di quanto non fu allora; questo è dovuto al fatto che negli ultimi anni si è verificata una esplosione nella tecnologia per le comunicazioni via rete e nel worldwide web. Come normalmente accade tale parola viene usata con significati differenti. I vari significati differiscono in due aspetti: *che tipo di entità si muove* e *in quale spazio si muove?*. Si presentano molte scelte naturali:

- i processi si muovono nello spazio fisico dove vengono computati;

- i processi si muovono nello spazio virtuale dei processi collegati;
- i collegamenti si muovono nello spazio virtuale dei processi collegati.

Il π-calcolo non è il risultato di una analisi completa di tutte queste possibilità perchè questo risulterebbe vago ed incocludente. Ma tra queste tre possibilità è stata scelta l'ultima per questo motivo: ad un determinato livello di astrazione, la locazione di un processo nello spazio virtuale dei processi è determinata dai collegamenti che ha con gli altri processi, in altre parole si può comunicare solo con i propri vicini. Secondo questo modo di pensare, il movimento di un processo può essere completamente rappresentato dal movimento dei suoi collegamenti; quindi la seconda scelta si può ridurre alla terza. Inoltre l'ultima scelta è più generale poiché si può muovere il collegamento di un processo P senza muovere tutti gli altri. E cosa dire della prima scelta? Si potrebbe affermare che tale scelta potrebbe essere ridotta alla seconda, in quanto lo spazio fisico dove i processi vengono computati può essere modellato come un processo (virtuale), e la sua locazione modellata come un collegamento; cioè per essere locati ad un certo sito bisogna avere il collegamento a tale sito. Formalmente parlando, i processi in computazione in un particolare sito sono quelli che possiedono il collegamento con tale sito (come un programma che può venire interpretato come dato di input da un programma interprete). Purtroppo la riduzione della prima scelta alla seconda risulta meno convincente del passaggio dalla seconda alla terza. Il π-calcolo adotta la terza scelta, non perchè risulta fondamentale per tutti e tre i significati di 'mobilità' ma perchè risulta essere economica, flessibile e abbastanza semplice. Per proseguire il nostro discorso è meglio soffermarci un attimo sulla parola 'locazione'. Ci sono stati molti studi sulla nozione di locazione (fisica), alcuni dei quali nell'ambito dei CCS (Calculi of communicating systems) e in quello del π-calcolo. Un recente studio, molto motivato dalla moderna tecnologia di computazione mobile, è il calcolo degli ambienti di Cardelli e Gordon [11]. Un 'ambiente' è una locazione per attività ed inoltre gli ambienti si possono annidare tra loro; anche se il calcolo degli ambienti usa molti dei costrutti del π-calcolo, si basa su una nozione considerevolmente diversa di azione: l'azione base non risulta essere l'interazione ma il movimento di un ambiente verso un altro. Per apprezzare la definizione formale del π-calcolo che daremo più avanti, presentiamo un esempio il cui la mobilità è di centrale importanza. Immaginiamo dei veicoli che si muovono, ognuno connesso ad una trasmittente T da un'unica lunghezza d'onda. Tutte le trasmittenti sono connesse ad un'unica unità di controllo centrale. In qualche caso (ad esempio quando sparisce il segnale) un veicolo può essere collegato ad un'altra trasmittente T. Non dobbiamo confondere i due tipi di movimento: il movimento fisico delle macchine ed il movimento virtuale dei legami di comunicazione tra veicoli e

trasmittenti. Le due cose sono ovviamente indipendenti, perciò vedremo nell'analisi del protocollo di comunicazione (Esempio 2) che il movimento fisico del veicolo dà luogo al movimento virtuale del suo collegamento alla trasmittente.

Definiamo ora il π-calcolo formalmente. Iniziamo con la versione monadica, nella quale un messaggio consiste di esattamente un nome. Più avanti daremo anche la definizione per un calcolo poliadico.

2.1 Nomi, azioni e processi

Assumiamo che esista un insieme infinito $\mathcal{N}$ di nomi. Le lettere minuscole $x, y, z, \ldots$, variano in $\mathcal{N}$. L'*azione prefisso* π del π-calcolo rappresenta sia l'invio che la ricezione del messaggio (un nome) e anche transizioni non osservabili (ad esempio azioni interne). La sintassi è la seguente:

π:= $x(y)$ riceve y lungo il canale x;
$\overline{x} < y >$ manda y lungo il canale x;
τ azione non osservabile.

Vediamo ora come si può comporre un processo.

- Con $\sum_{i \in I} \pi_i.P_i$ (I insieme di indici finito) indichiamo il fatto che a scelta può venire eseguito uno tra i processi $\pi_i.P_i$ con $i \in I$.
- Con $P_1 \mid P_2$ indichiamo il fatto che i due processi vengono eseguiti in parallelo.
- Con $(\nu a)P$ si indica la creazione del nuovo nome a e la limitazione del suo scope al processo P.
- Con !P si indica la replica del processo P infinite volte.

Definizione 1 ***Il π-calcolo*** *L'insieme $\mathcal{P}^\pi$ di espressioni di processi in π-calcolo è definito dalla seguente sintassi:*

$$P := \sum_{i \in I} \pi_i.P_i \mid P_1 \mid P_2 \mid (\nu a)P \mid !P$$

dove I è un insieme finito di indici. Il processo $\sum_{i \in I} \pi_i.P_i$ viene chiamato sommatoria o somma.

Nella somma $\sum_{i \in I} \pi_i.P_i$ diciamo che P_i ha come 'guardia' π_i in quanto l'azione rappresentata da π_i deve essere eseguita prima che P_i risulti attivo. Con 0 indichiamo la somma vuota e la omettiamo se posta dopo un' azione, scrivendo per esempio $\overline{x} < y >$ al posto di $\overline{x} < y > .0$. Sia la restrizione (νy) che l'azione di input $x(y)$ rendono bound il nome y (limitano cioè il suo scope); d'altro canto diciamo che il nome y è libero nell'azione di output $\overline{x} < y >$. I nomi liberi di un processo P li indichiamo con $fn(P)$.

Esempio 1 *Prima di presentare le regole formali per le reazioni, le illustriamo con un esempio. Sia*

$$P = (\nu z)((\overline{x} < y > + z(w).\overline{w} < y >) \mid x(u).\overline{u} < v > \mid \overline{x} < z >)$$

(Da notare che gli unici nomi liberi in P sono x, y e v.) Diciamo che una coppia di azioni una positiva (input) e una negativa (output) che usano lo stesso canale, come ad esempio $x(u)$ e $\overline{x} < y >$, sono complementari. Se, come qui, tali azioni non hanno guardie e non sono nella stessa somma allora costituiscono un redex; lo sviluppo di questi redex costituisce la reazione $P \to P'$, che invoca la sostituzione, in questo caso, $\{y/u\}$.

In questo esempio P ha due redex, la coppia $x(u)$,$\overline{x} < y >$ e la coppia $x(u)$,$\overline{x} < z >$. Perciò sono possibili due reazioni $P \to P_1$ e $P \to P_2$ dove

$$P_1 = (\nu z)(0 \mid \overline{y} < v > \mid \overline{x} < z >)$$

$$P_2 = (\nu z)((\overline{x} < y > + z(w).\overline{w} < y >) \mid \overline{z} < v > \mid 0)$$

Non ci sono ulteriori redex in P_1, ma ce n'è uno in P_2: la coppia $z(w)\,, \overline{z} < v >$, che deriva dalla sostituzione $\{z/u\}$ innescata dalla prima reazione. Quindi abbiamo anche $P_2 \to P_3$, dove

$$P_3 = (\nu z)(\overline{v} < y > \mid 0 \mid 0).$$

2.2 Congruenza strutturale e reazioni

Definizione 2 ***Contesti di processo*** *I contesti di processo $\mathcal{C}$ sono dati dalla seguente sintassi:*

$$\mathcal{C} := [] \mid \pi.\mathcal{C} + M \mid (\nu a)\mathcal{C} \mid \mathcal{C} \mid P \mid !\mathcal{C}$$

Nel seguito $\mathcal{C}[Q]$ denota il risultato che si ottiene riempiendo [] nel contesto $\mathcal{C}$ con il processo Q. I contesti elementari sono: $\pi.[] + M$, $(\nu a)[]$, $[] \mid P$, $P \mid []$ e $![]$.

Definizione 3 ***Congruenza di processi*** *Sia $\cong$ una relazione di equivalenza su $\mathcal{P}$. Allora $\cong$ è detta una congruenza di processi se viene preservata da tutti i contesti elementari; cioè se $P \cong Q$ allora*

$$\pi.P + M \cong \pi.Q + M;$$

$$(\nu x)P \cong (\nu x)Q;$$

$P \mid R \cong Q \mid R$;

$R \mid P \cong R \mid Q$;

$!P \cong !Q$.

La seguente proposizione si ricava facilmente:

Proposizione 1 *Una arbitraria relazione di equivalenza $\cong$ viene detta una congruenza di processi se e solo se, per ogni contesto $\mathcal{C}, P \cong Q$ implica $\mathcal{C}[P] \cong \mathcal{C}[Q]$.*

Definizione 4 ***Congruenza strutturale*** *Due espressioni di processi P e Q sono strutturalmente equivalenti nel π-calcolo, $P \equiv Q$, se possiamo trasformare una nell'altra usando le seguenti equazioni (in entrambe le direzioni):*

- *Rinomina di nomi bound (α-conversion);*
- *Riordinamento dei termini in una somma;*
- $P \mid 0 \equiv P$, $P \mid Q \equiv Q \mid P$, $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$;
- $(\nu x)(P \mid Q) \equiv P \mid (\nu x)Q$ *se* $x \notin fn(P)$, $(\nu x)0 \equiv 0$, $(\nu xy)P \equiv (\nu yx)P$;
- $!P \equiv P \mid !P$.

Definizione 5 *Una espressione del tipo*

$$(\nu \vec{a})(M_1 \mid \ldots \mid M_m \mid !Q_1 \mid \ldots \mid !Q_n)$$

viene detta in forma standard se ogni M_i è una somma non vuota, ed ogni Q_j è in forma standard. (Se $m = n = 0$ allora la forma è $(\nu \vec{a})0$, se $\vec{a}$ è il vettore nullo allora non ci sono restrizioni.)

Proposizione 2 *Ogni processo è strutturalmente congruente ad una forma standard.*

Siamo ora pronti per definire come differenti componenti concorrenti di un processo possono reagire assieme. Con l'aiuto della congruenza strutturale, determiniamo le possibili reazioni in un processo con due semplici regole: le prime due della definizione 6. La regola REACT permette la sincronizzazione tra azioni di input e di output, la regola TAU considera le reazioni interne. Le regole PAR e RES ci dicono che le reazioni possono avvenire all'interno di una composizione o di una restrizione. L'ultima regola, la STRUCT, permette di usare la congruenza strutturale ogni volta che una reazione viene inferta.

Definizione 6 ***Reazione*** *La relazione di reazione* $\rightarrow$ *su* $\mathcal{P}^{\pi}$ *contiene esattamente quelle transizioni che possono venire inferte dalle seguenti regole:*

- *TAU:* $\tau.P + M \rightarrow P$;
- *REACT:* $(x(y).P + M) \mid (\overline{x} < z > .Q + N) \rightarrow \{z/y\}P \mid Q$;
- *PAR:* $\frac{P \rightarrow P'}{P|Q \rightarrow P'|Q}$;
- *RES:* $\frac{P \rightarrow P'}{(\nu x)P \rightarrow (\nu x)P'}$;
- *STRUCT:* $\frac{P \rightarrow P'}{Q \rightarrow Q'}$ *se* $P \equiv Q$ *e* $P' \equiv Q'$.

Da notare il fatto che non c'è nessuna regola di reazione per l'operatore di replica. Quello di cui abbiamo bisogno per analizzare il comportamento di tale operatore è la regola STRUCT. Inoltre con $\rightarrow *$ indichiamo la chiusura riflessiva e transitiva di $\rightarrow$. Ecco allora che l'esempio delle trasmittenti si traduce così:

Esempio 2 *(Per semplicità consideriamo solo il caso di un veicolo e di due trasmittenti) Consideriamo per prima la trasmittente* $Trans$*, che risulta essere in contatto con la macchina (Car) ed il controllo centrale (Control). Consideriamo i suoi collegamenti correnti come parametri, e la rappresentiamo in questo modo:*

$$Trans < talk_1, switch_1, gain_1, lose_1 >.$$

La macchina può comunicare ($talk_1$*) tramite la trasmittente, ma in ogni momento la centrale di controllo può ordinare alla trasmittente di abbandonare il collegamento con la macchina (*$lose_1$*). Usiamo una equazione ricorsiva per definire il comportamento della trasmittente come segue:*

$$Trans(talk, switch, gain, lose) :=$$

$$talk.Trans < talk, switch, gain, lose >$$

$$+lose(t, s).\overline{switch} < t, s > .Idtrans < gain, lose >$$

$$Idtrans(gain, lose) := gain(t, s).Trans < t, s, gain, lose >.$$

L'azione $lose(t, s)$ *riceve nomi, mentre* $\overline{switch} < t, s >$ *invia nomi. Da notare che* $Control$*, dicendo a* $Trans$ *di abbandonare la macchina, emette una nuova coppia di canali da trasmette alla macchina per sostituire la sua vecchia coppia. Emette anche la stessa coppia all'altra trasmittente, che inizia a interagire con la macchina. Nella nostra semplice formulazione,* $Control$ *emette sempre una delle due possibili coppie di canali:*

$$Control_1 := \overline{lose_1} < talk_2, switch_2 > .\overline{gain_2} < talk_2, switch_2 > .Control_2;$$

$$Control_2 := \overline{lose_2} < talk_1, switch_1 > .\overline{gain_1} < talk_1, switch_1 > .Control_1.$$

In conclusione la macchina può o parlare o, se richiesto, collegarsi alla nuova coppia di canali:

$$Car(talk, switch) := \overline{talk}.Car < talk, switch > + switch(t, s).\, Car < t, s >.$$

Vediamo ora di mettere assieme tutto il sistema in modo da ottenere un processo. Non è molto difficile, infatti basta creare una composizione dei quattro processi:

$System_1 := (\nu\, talk_1, switch_1, gain_1, lose_1, talk_2, switch_2, gain_2, lose_2)$
$(Car < talk_1, switch_1 > | \; Trans_1 \mid Idtrans_2 \mid Control_1)$
dove:

$$Trans_i := Trans < talk_i, switch_i, gain_i, lose_i >;$$

$$Idtrans_i := Idtrans < gain_i, lose_i > \; (i = 1, 2).$$

Si verifica facilmente che:

$System_1 \rightarrow *System_2$
ove
$System_2 := (\nu\, talk_1, switch_1, gain_1, lose_1, talk_2, switch_2, gain_2, lose_2)$
$(Car < talk_2, switch_2 > | \; Trans_2 \mid Idtrans_1 \mid Control_2)$.

2.3 Mobilità

Supponiamo di avere il seguente processo:

$$(\nu z)(P \mid R) \mid Q$$

dove $P = \overline{x} < z > .P'$ e $Q = x(y).Q'$. Abbiamo così la seguente reazione:

$$(\nu z)(P \mid R) \mid Q \rightarrow P' \mid (\nu z)(R \mid Q'')$$

dove $Q'' = \{z/y\}Q'$. Un modo per interpretare questa transizione è dire che R si è spostato da P a Q. Ma è più preciso dire che il collegamento a R si è spostato da P a Q, poichè ci potrebbero essere altri collegamenti verso R. Supponiamo per esempio che R sia una persona, z il suo numero di telefono e che gli altri collegamenti a R siano fisici (ad esempio i vicini di casa di R). Se P dice a Q il numero di telefono di R e lo dimentica, non vuol certo dire che R abbia cambiato di casa. Risulta perciò appropriato parlare di movimenti di accessi, o cambio di contiguità piuttosto che di movimenti di processi.

2.4 Il π-calcolo poliadico

Naturalmente vogliamo avere la possibilità di spedire messaggi composti da più nomi come nell'esempio 2. Perciò vogliamo permettere la forma

$$x(y_1 \dots y_n).P \text{ e } \overline{x} < z_1 \dots z_n > .Q$$

(dove tutti gli y_i sono distinti) per ogni $n \geq 0$. Questo può essere effettuato con il codice del π-calcolo monadico ma non nel modo che appare più ovvio. Il modo che appare più ovvio è quello di codificare l'espressione in questo modo:

$$x(y_1).\dots.x(y_n).P \text{ e } \overline{x} < z_1 > \dots.\overline{x} < z_n > .Q$$

Ma consideriamo il seguente esempio:

$$x(y_1y_2).P \mid \overline{x} < z_1z_2 > .Q \mid \overline{x} < z_1'z_2' > .Q'$$

ovviamente ci si aspetta che la coppia y_1y_2 sia rimpiazzata o da z_1z_2 o da $z_1'z_2'$. Ma con questo modo di procedere otteniamo:

$$x(y_1).x(y_2).P \mid \overline{x} < z_1 > .\overline{x} < z_2 > .Q \mid \overline{x} < z_1' > .\overline{x} < z_2' > .Q'$$

che permette ad esempio che y_1y_2 venga rimpiazzato con $z_1'z_1$ cosa non prevista. Per una codificazione corretta , dobbiamo assicurarci che non ci siano interferenze lungo il canale in cui il messaggio composto viene inviato. Per inviare il messaggio $< z_1, \dots, z_n >$, per prima cosa inviamo il nome $fresh\ w$ (nome aggiunto con l'operazione (νw)) su x, e poi inviamo le componenti z_i una alla volta lungo w. Traduciamo così le azioni con prefissi multipli come segue:

$$x(y_1, \dots, y_n).P \to x(w).w(y_1).\ \dots\ .w(y_n).P$$
$$\overline{x} < z_1, \dots, z_n > .Q \to (\nu w)(\overline{x} < w > .\overline{w} < y_1 > .\dots.\overline{w} < y_n > .Q$$

Questo tipo di codifica ci permette di considerare il calcolo monadico come base del $\pi-$calcolo. Inoltre risalta l'importanza dei nomi ristretti. Quindi nell'usare il π-calcolo poliadico indicheremo con $x(\overrightarrow{y})$ e $\overline{x} < \overrightarrow{z} >$ le azioni con prefissi multipli. Perciò per permettere la trasmissione di messaggi composti, estendiamo la principale regola di reazione come segue:

$$\text{REACT: } (x(\overrightarrow{y}).P + M) \mid (\overline{x} < \overrightarrow{z} > .Q + N) \to \{\overrightarrow{z}/\overrightarrow{y}\}P \mid Q$$

dove i vettori $\overrightarrow{y}$ e $\overrightarrow{z}$ devono avere la stessa lunghezza.

Capitolo 3

Un dialetto dello Spi-calcolo

3.1 Preliminari

La sintassi e la semantica che vengono usate sono quelle del π-calcolo a cui si toglie l'operatore di somma e si aggiungono delle operazioni classificate in costruttori e distruttori [3]. Queste operazioni possono essere ad esempio criptazione e decriptazione asimmetrica, criptazione e decriptazione simmetrica, creazione di n-uple e corrispondenti proiezioni. Come nel π-calcolo applicato [5], tali operazioni non sono ben definite ed inoltre il π-calcolo applicato risulta essere più generale in quanto non richiede la classificazione delle operazioni in costruttori e distruttori. La sintassi usata e' quella proposta nella tabella 3.1. In tale sintassi si distinguono quindi due categorie: quella dei termini e quella dei processi. Assumiamo un insieme infinito per i nomi ed uno infinito per le variabili (ogni nome e ogni variabile possono essere o liberi o bound, diciamo che sono bound in P quando il loro scope è limitato al processo P, altrimenti diciamo che sono liberi), assumiamo inoltre un insieme di simboli per i costruttori ed i distruttori, dove f è una variabile sull'insieme dei costruttori e g una variabile sull'insieme dei distruttori. I costruttori sono usati per costruire nuovi termini utilizzando variabili, nomi e costruttori. I distruttori invece manipolano i termini in un processo: sono funzioni parziali sui termini che i processi possono applicare. Per ogni distruttore g di arità n esiste un insieme minimo di equazioni def(g) che definisce g , tale che per ogni elemento $e \in def(g)$, $e = g(M_1, \ldots, M_n) = M$, dove M_1,...,M_n,M sono termini senza nomi liberi. Allora $g(N_1, \ldots, N_n)$ è definita se e solo se esiste una sostituzione σ (sostituzione che mappa nomi e variabili in termini) ed una equazione $g(M_1,\ldots,M_n)=M$ in def(g) tale che $N_i=\sigma M_i$ per ogni $i \in \{1, \ldots, n\}$, e si ha $g(N_1,\ldots,N_n)=\sigma M$. Questo insieme di equazioni può essere infinito, ma in esempi concreti è quasi sempre finito e di pic-

M,N :		termini
	x,y,z	variabili
	a,b,c,k,s	nomi
	$f(M_1,\ldots,M_n)$	applicazione di costruttore
P,Q :		processi
	$\overline{M}< N >$.P	output
	M(x).P	input
	0	processo nullo
	P | Q	composizione parallela
	!P	replica
	(νa)P	restrizione
	$let\, x = g(M_1, \ldots, M_n) in\, P\, else\, Q$	applicazione di distruttore
	$let\, x = M\, in\, P$	definizione locale
	$if\, M = N\, then\, P\, else\, Q$	condizionale

Tabella 3.1: Sintassi del calcolo

cole dimensioni. Quindi il processo $let\, x = g(M_1, \ldots, M_n)\, in\, P\, else\, Q$ cerca di valutare $g(M_1, \ldots, M_n)$ cioè determina se $g(M_1, \ldots, M_n)$ è definita, e se questo succede allora x è reso bound al risultato e P viene eseguito, altrimenti viene eseguito Q. Usando questi costruttori e distruttori, possiamo rappresentare strutture dati , come n-uple, ed operazioni crittografiche, ad esempio come segue:

- $ntuple(M_1, \ldots, M_n)$ è l'n-upla di termini M_1,...,M_n, dove $ntuple$ è un costruttore. Le n proiezioni sono i distruttori ith_n per $i \in \{1, \ldots, n\}$, definiti da:

  $$def(ith_n) = \{ith_n(ntuple(M_1, \ldots, M_n)) = M_i\};$$

- $sencrypt(M, N)$ è la criptazione simmetrica del messaggio M con chiave N, dove $sencrypt$ è un costruttore. Il corrispondente distruttore $sdecrypt$ è definito come segue:

  $$def(sdecrypt) = \{sdecrypt(sencrypt(M, N), N) = M\},$$

  perciò, $sdecrypt(M', N)$, ritorna la decriptazione di M' se M' è un messaggio criptato con chiave N;

- per rappresentare la criptazione asimmetrica dobbiamo usare due costruttori *pk* e *pencrypt*: $pk(M)$ crea una chiave pubblica partendo dalla chiave segreta M e $pencrypt(M, N)$ cripta M con chiave N. Il corrispondente distruttore *pdecrypt* è definito da:

$$def(pdecrypt) = \{pdecrypt(pencrypt(M, pk(N)), N) = M\}.$$

Perciò, il calcolo supporta molte delle operazioni comuni nei protocolli di sicurezza. Gli altri costrutti della sintassi sono standard, molti di essi derivano dal π-calcolo. Il processo di input $M(x)$.P riceve il messaggio lungo il canale M ed esegue P con x bound al messaggio di input. Il processo di output $\overline{M}< N >$.P manda il messaggio lungo il canale M ed esegue P. Qui si usa il termine M per rappresentare il canale: M può essere un nome, una variabile o un termine composto, in ogni caso il processo si blocca se M non si riduce ad un nome durante l'esecuzione. Il calcolo considerato è monadico, in ogni caso si può considerare anche un calcolo poliadico in quanto le n-uple possono essere viste come termini. Inoltre è sincrono nel senso che P viene eseguito dopo che il messaggio è stato inviato. Il processo 0 (nil) non fa nulla. Il processo P | Q è la composizione parallela di P e Q. La replica !P rappresenta un illimitato numero di copie di P in parallelo. La restrizione (νa)P crea un nuovo nome a, e poi esegue P. La definizione locale $let\, x = M\, in\, P$ esegue P con x bound al termine M. Il condizionale $if M = N\, then\, P\, else\, Q$ esegue P se M ed N si riducono allo stesso termine durante l'esecuzione, altrimenti esegue Q. Come al solito si omette la clausola *else* quando consiste in Q=0. Il nome a è bound nel processo (νa)P. La variabile x è bound in P nei processi $M(x).P$, $let\, x = M\, in\, P$, e $let\, x = g(M_1, \ldots, M_n)\, in\, P\, else\, Q$.

Con $fn(P)$ e $fv(P)$ si indicano gli insiemi dei nomi e delle variabili libere rispettivamente.

Diciamo che un processo è chiuso se non ha variabili libere, può però avere nomi liberi. I processi vengono inoltre identificati a meno di rinomina di nomi bound.

3.2 Semantica formale

Le regole proposte nella tabella 3.2 assiomatizzano la relazione di riduzione $\rightarrow$ per i processi, definendo cioè la semantica operazionale del calcolo. Regole ausiliarie assiomatizzano la relazione di congruenza strutturale $\equiv$; questa relazione è molto utile per trasformare i processi in modo tale da poter applicare le regole di riduzione. Sia $\equiv$ che $\rightarrow$ sono definiti solo su processi chiusi. Con $\rightarrow^*$ si denota la chiusura transitiva e riflessiva di $\rightarrow$. Si dice inoltre che P invia immediatamente M su c se e solo se $P \equiv \overline{c} < M > .Q \mid R$

per qualche processo Q e qualche processo R. Si dice che il processo P invia M su c se e solo se $P \rightarrow^* Q$ e Q invia M immediatamente su c.

Due processi sono congruenti a meno di α-conversione.

P | 0 ≡ P	P ≡ Q ⇒ P | R ≡ Q | R
P | Q ≡ Q | P	P ≡ Q ⇒ !P ≡! Q
(P | Q) | R ≡ P | (Q | R)	P ≡ Q ⇒ (νa)P ≡ (νa)Q
!P ≡ P | !P	P ≡ P
$(\nu a_1)(\nu a_2)P \equiv (\nu a_2)(\nu a_2)P \, se \, a_1 \neq a_2$	Q ≡ P ⇒ P ≡ Q
$(\nu a)(P \mid Q) \equiv P \mid (\nu a)Q \, se \, a \notin fn(P)$	P ≡ Q, Q≡ R ⇒ P ≡ R

$\overline{a} < M >$.Q | $a(x)$.P → Q | P $\{M/x\}$	(Red I/O)
$let \, x = g(M_1, \ldots, M_n) \, in \, P \, else \, Q \rightarrow P\{M'/x\}$ se $g(M_1, \ldots, M_n) = M'$	(Red Destr 1)
$let \, x = g(M_1, \ldots, M_n) \, in \, P \, else \, Q \rightarrow Q$ se $g(M_1, \ldots, M_n)$ risulta non definita	(Red Destr 2)
$let \, x = M \, in \, P \rightarrow P\{M/x\}$	(Red Let)
$if \, M = M \, then \, P \, else \, Q \rightarrow P$	(Red Cond1)
$if \, M = N \, then \, P \, else \, Q \rightarrow Q$ se $M \neq N$	(Red Cond 2)
$P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R$	(Red Par)
$P \rightarrow Q \Rightarrow (\nu a)P \rightarrow (\nu a)Q$	(Red Res)
$P' \equiv P, \, P \rightarrow Q, \, Q \equiv Q' \Rightarrow P' \rightarrow Q'$	(Red ≡)

Tabella 3.2: Congruenza strutturale e riduzione

Capitolo 4

Segretezza e risultati precedenti

Questo capitolo ci elenca i risultati ottenuti nell'approccio all'analisi di protocolli [3], tra loro molto diverse (si basano però sulle nostre stesse nozioni di sintassi -Tabella 3.1- e semantica -Tabella 3.2-) ma che si dimostrano essere equivalenti. Il primo approccio è quello del calcolo di processi tipati. Ci sono molti sistemi a tipi che danno informazioni per quanto riguarda la sicurezza [1, 4, 10, 19, 20, 21]. Il secondo approccio è quello in cui i protocolli di sicurezza sono rappresentati come programmi logici che verranno successivamente analizzati [15, 29, 7, 12, 14]. Prima di passare a tale elenco diamo la definizione di segretezza per un protocollo P.

4.1 Definizione di segretezza

Un protocollo P preserva la segretezza del dato M se P non rende mai noto M o qualunque altro dato che permetta la computazione di M, anche se messo in parallelo con un avversario Q. Definizioni analoghe sono comuni nell'analisi di protocolli. (Ci sono in ogni caso delle alternative, in particolare alcune basate sul concetto di non-interferenza [2]). L'avversario Q viene rappresentato come un processo del calcolo, con delle ipotesi che caratterizzano le capacità iniziali di Q. Queste ipotesi vengono formulate semplicemente usando un insieme di nomi S. Intuitivamente, all'inizio Q sarà a conoscenza dei nomi in S, e potrà acquisire ulteriori capacità non presenti in S durante la computazione, creando nuovi nomi e ricevendo messaggi da P. Per rappresentare il fatto che Q inizialmente può conoscere non solo nomi ma anche termini più complessi, P invierà inizialmente questi termini lungo il canale pubblico $c \in S$, così il fatto che S sia un insieme di nomi non causa perdita di generalità.

Definizione 7 *Sia S un insieme finito di nomi. Il processo chiuso Q è un S-avversario*

se e solo se $fn(Q) \subseteq S$. *Il processo chiuso* P *preserva la segretezza di* M *da* S *se e solo se* $P \mid Q$ *non invia* M *su* c *per un qualsiasi* S*-avversario* Q *ed un qualsiasi* $c \in S$.

Se P preserva la segretezza di M da S , chiaramente non può inviare M su qualche $c \in S$, cioè su qualche canale conosciuto dall'avversario. Questa sicurezza corrisponde alla richiesta formale che P non invii mai M, in più, P non può mandare dati che possano permettere all'avversario di calcolarsi M, in quanto l'avversario potrebbe inviare M in qualche $c \in S$.

Perciò possiamo dire che un protocollo è sicuro se preserva la segretezza di tutti i suoi dati privati.

Passiamo ora ai risultati precedenti.

4.2 Risultati ottenuti con un sistema a tipi

Questa sezione presenta un sistema a tipi per i processi in considerazione ideato da Martín Abadi [3]. La tabella 4.1 ci fornisce le regole per tale sistema.

Nelle varie regole il termine u varia tra i nomi e le variabili e T varia nell'insieme dei tipi. Le regole riguardano tre situazioni:

- $E \vdash \diamond$ significa che E è un ambiente ben formato;
- $E \vdash M : T$ significa che il termine M è di tipo T nell'ambiente E;
- $E \vdash P$ ci dice che il processo P è ben tipato nell'ambiente E.

Il sistema di tipi è parametrizzato da un insieme di tipi $Types$ e da un sottoinsieme non vuoto $T_{Pubblic} \subseteq Types$. Questi parametri verranno fissati in ogni istanza del sistema a tipi. Con $T_{Pubblic}$ si indicherà sempre l'insieme dei tipi a cui appartengono i dati che l'avversario può conoscere. Il sistema a tipi inoltre si basa su di una funzione $conveys : Types \rightarrow \mathcal{P}(Types)$ tale che:

- (P0) Se $T \in T_{Pubblic}$, allora $conveys(T) = T_{Pubblic}$.

Intuitivamente $conveys(T)$ è l'insieme dei tipi di dati che vengono trasmessi (o ricevuti) da canali di tipo T. (E' l'insieme vuoto quando gli elementi di tipo T non possono essere usati come canali). I dati trasmessi (o ricevuti) da un canale pubblico devono essere pubblici, poiché l'avversario può ottenerli leggendoli da quel canale. D'altro canto i dati pubblici possono apparire su canali pubblici in quanto possono essere stati spediti da un avversario. Il sistema a tipi si basa inoltre su di una funzione parziale da tipi a tipi

$O_f : Types^n \to Types$ per ogni costruttore f di arità n e su di un'altra funzione da tipi ad insiemi di tipi $O_g : Types^n \to \mathcal{P}(Types)$ per ogni distruttore g di arità n (i costruttori ed i distruttori possono accettare argomenti di tipi differenti, e ritornano dei risultati che dipendono dai tipi degli argomenti). Questi operatori O_f e O_g danno i tipi delle applicazioni di costruttori e distruttori. Queste applicazioni non hanno necessariamente un unico tipo o più tipi (anche se i termini hanno un unico tipo in un dato ambiente).

Esempio 3 *Sia g un distruttore tale che, data una terna di nomi come input, ritorna il nome minore in ordine lessicografico, cioè* $g(a,b,c) = a$, $g(d,b,c) = b$ *ecc. Siano inoltre* $a, d : T_1$, $b : T_2$ *e* $c : T_3$*; quindi* $g(a,b,c) : T_1$, $g(d,b,c) : T_2$ *anche se le due terne hanno gli stessi tipi. Perciò* $O_g(T_1, T_2, T_3) \supseteq \{T_1, T_2\}$.

Si richiedono le seguenti proprietà:

- (P1) Se per ogni $i \in \{1, \ldots, n\}$, $T_i \in T_{Pubblic}$, allora $O_f(T_1, \ldots, T_n)$ è definito ed inoltre $O_f(T_1, \ldots, T_n) \in T_{Pubblic}$.
- (P2) Se per ogni $i \in \{1, \ldots, n\}$, $T_i \in T_{Pubblic}$ e $T \in O_g(T_1, \ldots, T_n)$, allora $T \in T_{Pubblic}$.
- (P3) Per ogni equazione $g(M_1, \ldots, M_n) = M$ in def(g), se per ogni $i \in \{1, \ldots, n\}$, $E \vdash M_i : T_i$, allora esiste $T \in O_g(T_1, \ldots, T_n)$ tale che $E \vdash M : T$.

Le prime due proprietà riflettono il fatto che una funzione applicata a termini pubblici deve per forza essere pubblica, poiché l'avversario può calcolarla. La terza proprietà dice essenzialmente che la definizione di O_g sui tipi è compatibile con la definizione di g sui termini.

Vediamo ora le regole che caratterizzano un tale sistema (Tabella 4.1).

Le regole per i tipi per il processo nullo (regola 7), per la composizione parallela (regola 8), per la replica (regola 9), per la restrizione (regola 10) e per la definizione locale (regola 12) sono standard. Usiamo una scrittura corsiva per la restrizione, così non esplicitiamo il tipo di a nel costrutto (νa). Questo modo di procedere fa si che il tipo di a cambi in base all'ambiente. Per la regola 5 (regola di output), il processo $\overline{M} < N > .P$ è ben tipato solo se dati di tipo T' di N possono essere ricevuti o inviati su di un canale di tipo T di M, cioè se $T' \in conveys(T)$. Invece, per effettuare il controllo di tipi per il processo $M(x).P$ con la regola 6 (regola di input) la variabile x è considerata con tutti i tipi $T' \in conveys(T)$ dove T è il tipo di M. Questa scelta risulta essere inusuale, ma è basata sul fatto che può gestire dati di diverso tipo. Nei protocolli di sicurezza

- Ambienti ben formati:
  - Regola 1: $\frac{}{0 \vdash \diamond}$
  - Regola 2: $\frac{E \vdash \diamond \, u \notin dom(E)}{E, u : T \vdash \diamond}$
- Termini:
  - Regola 3: $\frac{E \vdash \diamond \, (u : T) \in E}{E \vdash u : T}$
  - Regola 4: $\frac{E \vdash \diamond \forall i \in \{1, \ldots, n\}, E \vdash M_i : T_i \, O_f(T_1, \ldots, T_n) \, risulta \, definito}{E \vdash f(M_1, \ldots, M_n) : O_f(T_1, \ldots, T_n)}$
- Processi:
  - Regola 5: $\frac{E \vdash M : T \, E \vdash N : T' \, T' \in conveys(T) \, E \vdash P}{E \vdash \overline{M} < N > .P}$
  - Regola 6: $\frac{E \vdash M : T \, \forall T' \in conveys(T) \, E, x : T' \vdash P}{E \vdash M(x).P}$
  - Regola 7: $\frac{E \vdash \diamond}{E \vdash 0}$
  - Regola 8: $\frac{E \vdash P \, E \vdash Q}{E \vdash P \mid Q}$
  - Regola 9: $\frac{E \vdash P}{E \vdash !P}$
  - Regola 10: $\frac{E, a : T \vdash P}{E \vdash (\nu a)P}$
  - Regola 11: $\frac{\forall i \in \{1, \ldots, n\} \, E \vdash M_i : T_i \, \forall T \in O_g(T_1, \ldots, T_n), \, E, x : T \vdash P \, E \vdash Q}{E \vdash let \, x = g(M_1, \ldots, M_n) \, in \, P \, else \, Q}$
  - Regola 12: $\frac{E \vdash M : T \, E, x : T \vdash P}{E \vdash let \, x = M \, in \, P}$
  - Regola 13: $\frac{E \vdash M : T \, E \vdash N : T' \, se \, T = T' \, allora \, E \vdash P \, altrimenti \, E \vdash Q}{E \vdash if \, M = N \, then \, P \, else \, Q}$

Tabella 4.1: Regole per i tipi

questa flessibilità è molto importante in quanto un canale può gestire sia dati di un avversario che di un partecipante onesto, così i tipi aiutano a distinguere i due casi. La regola 4 (applicazione di costruttore) tipa $f(M_1, \ldots, M_n)$ in relazione al corrispondente operatore O_f. La regola 11 (applicazione di distruttore) è simile alla regola per l'input; in $let\, x = g(M_1, \ldots, M_n)\, P\, else\, Q$, la variabile x viene considerata con tutti i possibili valori di $g(M_1, \ldots, M_n)$, cioè tutti gli elementi di $O_g(T_1, \ldots, T_n)$. La regola 13 (Condizionale) considera il fatto che se due termini hanno tipi diversi certamente sono differenti. In questo caso $if\, M = N\, then\, P\, else\, Q$ può essere ben tipata anche se P non risulta essere ben tipato.

4.2.1 Proprietà dei sistemi a tipi

Il lemma di riduzione ci dice che la tipabilità è preservata nelle computazioni.

Lemma 1 ***(Riduzione)*** *Se $E \vdash P$ e $P \to Q$ allora $E \vdash Q$.*

Il teorema di segretezza ci dice che se un processo chiuso è ben tipato in un ambiente E, ed il nome s non è di tipo $T_{Pubblic}$ in E, allora P preserva la segretezza di s da S, dove S è l'insieme dei nomi che sono di tipo $T_{Pubblic}$ in E. In altre parole, P preserva la sicurezza dei nomi il cui tipo non è in $T_{Pubblic}$ contro un avversario che può inviare, ricevere ed eseguire computazioni con i nomi in $T_{Pubblic}$.

Teorema 1 ***(Segretezza)*** *Sia P un processo chiuso. Supponiamo che $E \vdash P$, $E \vdash s : T'$, e $T' \notin T_{Pubblic}$. Sia $S = \{a \mid E \vdash a : T \, e \, T \in T_{Pubblic}\}$. Allora P preserva la segretezza di s da S.*

Questo teorema di segretezza è una conseguenza del lemma di riduzione e della tipabilità dell'avversario (cioè che ogni S-avversario Q può essere tipato con tutti i nomi e le variabili di tipo in $T_{Pubblic}$).

4.3 Risultati ottenuti con un sistema non a tipi

In questa sezione diamo una definizione precisa del protocol checker ideato da Bruno Blanchet [3], basato su di un sistema non a tipi.

4.3.1 Il Protocol Checker

Dato un processo chiuso P_0 ed un insieme di nomi S, il protocol checker costruisce un insieme di regole sotto forma di clausole di Horn. Da queste regole e tramite delle implicazioni, il PC (Protocol Checker) computa l'insieme di termini che un generale avversario può conoscere interagendo con il processo in questione, e quindi decide che un protocollo è sicuro se nessuno dei suoi nomi segreti appartiene a tale insieme, cioè se nessuno dei suoi nomi segreti può venire derivato tramite delle implicazioni dall'insieme di regole costruito.

Assumiamo che ogni restrizione $(\nu a)P$ in P_0 abbia un differente nome a, e che tale nome sia differente da ogni nome libero in P_0. Le regole usano due predicati: *attacker* e *message*. Il fatto $attacker(p)$ significa che l'avversario può conoscere p, mentre il fatto $message(p, p')$ significa che il messaggio p' può apparire sul canale p.

F:	fatti
$attacker(p)$	l'avversario conosce p;
$message(p, p')$	il messaggio p puo' apparire su p'.

Ora p e p' variano nell'insieme dei termini che vengono generati dalla seguente grammatica:

p:	termini
x, y, z	variabili;
$a[p_1, \ldots, p_n]$	nomi;
$f(p_1, \ldots, p_n)$	applicazione di costruttore.

Per ogni nome a in P_0 abbiamo il corrispondente costrutto di termini $a[p_1, \ldots, p_n]$. Trattiamo a come simbolo di funzione, e scriviamo $a[p_1, \ldots, p_n]$ piuttosto che $a(p_1, \ldots, p_n)$ solo per chiarezza. Se a è un nome reso bound dalla restrizione $(\nu a)P$ in P_0 allora l'arità della sua funzione è il numero delle azioni di input che precedono tale restrizione nell'albero sintattico di P_0. Perciò, nel PC, un nuovo nome si comporta come una funzione sugli input che precedono (lessicalmente) la sua creazione. Perciò distinguiamo i nomi solo quando vengono creati dopo aver ricevuto differenti input. Al contrario una restrizione in un processo crea sempre nuovi nomi; così facendo le regole non riflettono esattamente la semantica delle operazioni del processo, ma questa approssimazione è utile per l'automazione e innocua in molti esempi. Le regole comprendono regole per l'avversario e regole per il prtocollo che ora definiamo:

- **(Regole per l'avversario)** Inizialmente l'avversario conosce tutti i nomi nell'insieme S, da qui le regole $attacker(a[])$ per ogni $a \in S$. Proseguendo l'abilità dell'avversario è rappresentata dalle seguenti regole:

  - (Rf) Per ogni costruttore f di arità n,
    $attacker(x_1) \wedge \ldots \wedge attacker(x_n) \Rightarrow attacker(f(x_1, \ldots x_n))$;
  - (Rg) Per ogni distruttore g, per ogni equazione $g(M_1, \ldots, M_n) = M$ in def(g),
    $attacker(M_1) \wedge \ldots \wedge attacker(M_n) \Rightarrow attacker(M)$;
  - (Rl) $message(x, y) \wedge attacker(x) \Rightarrow attacker(y)$;
  - (Rs) $attacker(x) \wedge attacker(y) \Rightarrow message(x, y)$.

  Le regole (Rf) e (Rg) significano che l'avversario può eseguire tutte le operazioni su tutti i termini che conosce, (Rf) per i costruttori ed (Rg) per i distruttori. L'insieme di queste regole è finito se l'insieme dei costruttori ed ogni insieme def(g) sono finiti. La regola (Rl) significa che l'avversario può ricevere da tutti i canali che possiede, e la regola (Rs) che può spedire tutti i messaggi che ha su tutti i canali che possiede.

- **(Regole per il protocollo)** Sia ρ una funzione che associa ogni termine a nomi e variabili. Estendiamo ρ ai costruttori in questo modo (come una sostituzione): $\rho(f(M_1, \ldots, M_n)) = f(\rho(M_1), \ldots, \rho(M_n))$. La traduzione $[|\ P\ |]\rho h$ di un processo P è un insieme di regole dove ρ è una funzione che associa ogni termine a nomi e variabili, e h è una sequenza di fatti della forma $message(p, p')$. La sequenza vuota viene denotata con 0; la concatenazione di un fatto F alla sequenza h è denotata con $h \wedge F$.

  - $[|\ 0\ |]\rho h = 0$,
  - $[|\ P \mid Q\ |]\rho h = [|\ P\ |]\rho h \cup [|\ Q\ |]\rho h$,
  - $[|!P\ |]\rho h = [|\ P\ |]\rho h$,
  - $[|\ (\nu a)P\ |]\rho h = [|\ P\ |](\rho[a \rightarrow a[p'_1, \ldots, p'_n]])h\ se$
    $h = message(p_1, p'_1) \wedge \ldots \wedge message(p_n, p'_n)$,
  - $[|\ M(x).P\ |]\rho h = [|\ P\ |](\rho[x \rightarrow x])(h \wedge message(\rho(M), x))$,
  - $[|\ \overline{M} < N > .P\ |]\rho h = [|\ P\ |]\rho h \cup \{h \Rightarrow message(\rho(M), \rho(N))\}$,

– $[|\ let\, x = g(M_1, \ldots, M_n)\, in\, P\, else\, Q\ |]\rho h =$
$\cup\{[|\ P\ |]((\sigma\rho)[x \to \sigma' p'])(\rho h) \mid g(p'_1, \ldots, p'_n) = p'$ è in def(g) e (σ, σ') è la coppia più generale di sostituzioni tali che $\sigma\rho(M_1) = \sigma' p'_1, \ldots, \sigma\rho(M_n) = \sigma' p'_n\} \cup [|$ $Q\ |]\rho h$.

Perciò la traduzione di un processo risulta essere un insieme di regole che permettono di provare che il protocollo manda certi messaggi. La sequenza h tiene conto dei messaggi ricevuti dal processo poiché tali messaggi possono innescarne degli altri. La traduzione di 0 è l'insieme vuoto, dato che questo processo non fa nulla. La traduzione della composizione parallela $P \mid Q$ è l'unione della traduzione di P e Q. L'operatore di replica viene ignorato, poiché tutte le regole logiche sono applicabili arbitrariamente più volte. Per la restrizione, sostituiamo il nome ristretto a in questione con il termine $a[\ldots]$ che dipende dai messaggi ricevuti, come viene memorizzato in h. Questa sequenza viene estesa nella traduzione di un input, con l'input in questione. D'altro canto, la traduzione di un output aggiunge una clausola, questa clausola rappresenta la ricezione del messaggio in h che può innescare l'output in questione. In ultima, la traduzione dell'applicazione di un distruttore considera l'unione di tutte le clausole dove il distruttore risulta definito (con una sostituzione appropriata) e quelle per cui il distruttore non risulta definito; perciò tralasciamo la determinazione di quando un distruttore risulta definito o meno.

4.3.2 Proprietà del Protocol Checker

Sia $\rho = \{a \to a[] \mid a \in fn(P_0)\}$. Definiamo la regola di base corrispondente al processo P_0 in questo modo:

$$B_{P_0,S} = [|\ P_0\ |]\rho\emptyset \cup \{attacker(a[]) \mid a \in S\} \cup \{(Rf), (Rg), (Rl), (Rs)\}.$$

Si dimostra il seguente teorema:

Teorema 2 ***(Segretezza)*** *Sia P_0 un processo chiuso e $s \in fn(P_0)$. Se $attacker(s[])$ non può essere derivato da $B_{P_0,S}$, allora P_0 preserva la segretezza di s da S.*

Questo risultato è la base per un metodo per provare proprietà di segretezza. Certamente, che un fatto sia derivabile o meno da $B_{P_0,S}$ può risultare un problema indecidibile, ma in pratica esistono algoritmi che terminano in numerosi esempi di protocolli.

Capitolo 5

Assunzioni

Questo capitolo mostra che senza perdità di generalità possiamo considerare particolari computazioni del nostro calcolo che risultano essere convenienti da analizzare (come in [16, 3]).

Osserviamo per prima cosa che le costruzioni del tipo $if\ M = N\ then\ P\ else\ Q$ e $let\ x = M\ in\ P$ possono essere considerate come casi speciali di $let\ x = g(M_1, \ldots, M_n)\ in\ P\ else\ Q$ in questo modo:

- Sia *equals* il distruttore binario definito come segue:

  $equals(M, M) = M$ e con $equals(M, N)$ indefinito se $M \neq N$. Quindi

  $if\ M = N\ then\ P\ else\ Q$ si può scrivere come

  $let\ x = equals(M, N)\ in\ P\ else\ Q$ dove $x \notin fv(P)$.

- Sia *id* il distruttore unario definito come $id(M) = M$. Allora

  $let\ x = M\ in\ P$ si può scrivere come

  $let\ x = id(M)\ in\ P\ else\ 0$.

Facciamo ora la seguente assunzioni:

1. Per quanto appena visto consideriamo solo le costruzioni del tipo

   $let\ x = g(M_1, \ldots, M_n)\ in\ P\ else\ Q$.

2. Consideriamo solamente processi chiusi.

3. Indichiamo con $\mathcal{N}(P)$ l'insieme di tutti i nomi in P, con $\mathcal{V}(\mathcal{P})$ l'insieme di tutte le variabili in P, con $\mathcal{R}(P)$ l'insieme di tutti i costruttori in P, con $\mathcal{F}(P) =$

$\{f(M_1, \ldots, M_n) \mid f \in \mathcal{R}(P)$ di arità $n\}$ l'insieme di tutti i termini in P dovuti all'applicazione di costruttori (dove $\mathcal{F}(P)$ è il minimo insieme che soddisfa tale equazione), ed infine con $\mathcal{T}(P) = \mathcal{N}(P) \cup \mathcal{V}(P) \cup \mathcal{F}(P)$ l'insieme di tutti i termini in P (dove $\mathcal{T}(P)$ è il minimo insieme che soddisfa tale equazione). Assumiamo che l'insieme dei nomi sia diviso in infinite classi di equivalenza (dove ogni classe ha infiniti nomi) e che anche l'insieme delle variabili sia diviso in infinite classi di equivalenza (dove ogni classe ha infinite variabili). I rappresentanti delle classi degli elementi a ed x li indichiamo rispettivamente con $[a]$ e $[x]$. Estendiamo inoltre questa assunzione anche ai costruttori in questo modo $[f(M_1, \ldots, M_n)] = f([M_1], \ldots, [M_n])$.

4. Assumiamo che ogni azione di input, ogni operatore di restrizione ed ogni applicazione di distruttore rendano bound nomi e variabili distinti.

5. Assumiamo che l'insieme dei nomi bound sia disgiunto dall'insieme dei nomi liberi.

Abbiamo già visto che l'assunzione 1 non crea problemi di generalità e nemmeno le assunzioni 3,4,5 (un processo che soddisfa tali ipotesi lo chiamiamo processo AoT in quanto facciamo una assunzione sui termini) infatti ogni nome-variabile bound non è altro che un "*place-holder*" e perciò l'etichetta usata è ininfluente. Non è difficile vedere che, se l'operatore di replica viene trattato con cautela, allora una computazione che inizia con un processo AoT produce solamente processi AoT.

Esempio 4 *Sia Q un processo che contenga il sottoprocesso !P ove*

$P = a(x).\overline{x} < f(a) > .let\, y = g(a, x).\overline{b} < y >$. *Consideriamo ora !P usando una delle regole di congruenza strutturale presente nella tabella 3.2* $!P \equiv P \mid !P$ *possiamo sostituire in Q l'espressione !P con*

$a(x').\overline{x}' < f(a) > .let\, y' = g(a, x').\overline{b} < y' > \mid !P$

ove [x]=[x'], [y]=[y'] e x', y' $\notin \mathcal{T}(Q)$. *Osserviamo che x' e y' possono essere già in Q ma solo come rinomina di x e y in una precedente replica di P.*

L'analisi statica che considereremo più avanti, quando deve esaminare un processo quale ad esempio !P, ignora semplicemente ! e analizza solo P. Più precisamente analizza P sostituendo ogni nome a ed ogni variabile x con [a] e [x] (anche quando sono argomenti di un costruttore o di un distruttore). Così ogni nome bound b di P (o variabile) e tutte le rinomine b',b",... che possono venire introdotte sono simulate dal rappresentante [b] che viene usato nell'analisi.

Definizione 8 *Una sostituzione su* $\mathcal{T}$, *è una funzione* $\sigma : \mathcal{T} \to \mathcal{T}$, *tale che l'insieme* $S \subset \mathcal{T}$, *contenente i termini per i quali* $(M)\sigma \neq M$, *è finito. L'insieme S è chiamato il*

dominio di σ, e viene denotato con $dom(\sigma)$. La composizione di due sostituzioni $\sigma_1 \circ \sigma_2$ è la sostituzione σ tale che per $M \in dom(\sigma_1)$, $(M)\sigma = ((M)\sigma_1)\sigma_2)$, e per $N \in dom(\sigma_2) \setminus dom(\sigma_1)$, $(N)\sigma = (N)\sigma_2$.

Im quello che segue la sostituzione Θ verrà applicata al processo P.

Definizione 9 *In quello che segue una computazione $P \rightarrow^* Q$ è AoT quando soddisfa le seguenti condizioni:*

1. *P è AoT.*
2. *Nessuna α-conversione è applicata.*
3. *L'operatore di replica è trattato come nell'esempio precedente.*

Osserviamo che la seconda condizione della Definizione 9, non causa perdita di generalità poichè in ogni computazione che soddisfi le condizioni 1 e 3 nessuna inclusione di nomi può essere creata durante la computazione. E' facile dimostrare che considerare solo computazioni AoT non causa perdita di generalità:

Teorema 3 *Esiste una computazione $C : P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow \ldots \rightarrow P_n$, se e solo se esiste una computazione C': P_0'$\rightarrow P_1$'$\rightarrow P_2$'$\rightarrow \ldots \rightarrow P_n$', dove per ogni $i \in [0, n]$ P_i' può essere ottenuto da P_i, α-convertendo qualcuno dei suoi nomi e delle sue variabili bound e gli step corrispondenti sono gli stessi modulo queste α-conversioni.*

Dimostrazione. L'implicazione $\Leftarrow$ è ovvia. Consideriamo $\Rightarrow$.

α-convertendo propriamente parte dei nomi e delle variabili bound di P_0 è facile ottenere un processo P_0' che sia AoT. Aggiungiamo una etichetta extra ad ogni occorrenza di nomi e variabili di P_0: il nome o la variabile che appare nella corrispondente posizione di P_0'. Ogni computazione C che parte da un tale P_0 indica anche una corrispondente computazione C' nel senso che sostituendo in P_1,...,P_n ogni nome e ogni variabile con la sua nuova etichetta, si ottiene il processo P_1',...,P_n' che forma C'. C.v.d.

Durante l'esecuzione di un processo ogni step che usa le regole *Red I/O* e *Red Destr* 1 computa una sostituzione.

Definizione 10 *Sia $S : R \rightarrow R'$ l'esecuzione AoT di uno step che usa o la regola Red I/O o la regola Red Destr* 1*, e le regole istanziate siano, rispettivamente, la prima o la seconda delle seguenti:*

Red I/O: $\overline{a} < M > .P \mid a(x).Q \rightarrow P \mid Q\{M/x\}$;

$\overline{N} < M > .Q * \Theta \mid N'(x).P * \Theta \rightarrow (Q \mid P) * \{M/x\} \circ \Theta$
se $(N)\Theta = (N')\Theta$ e $(N)\Theta$ e' un nome (Red I/O)

$(let\, x = g(M_1, \ldots, M_n)\, in\, P\, else\, Q) * \Theta \rightarrow P * \{M'/x\} \circ \Theta$
se $g(M_1, \ldots, M_n) * \Theta = M'$ (Red Destr 1)

$(let\, x = g(M_1, \ldots, M_n)\, in\, P\, else\, Q) * \Theta \rightarrow Q * \Theta$
se $g(M_1, \ldots, M_n) * \Theta$ risulta non definita (Red Destr 2)

$P * \Theta \rightarrow Q * \Theta' \Rightarrow (P \mid R) * \Theta \rightarrow (Q \mid R) * \Theta'$ (Red Par)

$P * \Theta \rightarrow Q * \Theta' \Rightarrow (\nu a)P * \Theta \rightarrow (\nu a)Q * \Theta'$ (Red Res)

$P * \Theta \equiv P' * \Theta$, $P * \Theta \rightarrow Q * \Theta'$, $Q * \Theta' \equiv Q' * \Theta' \Rightarrow$
$P' * \Theta \rightarrow Q' * \Theta'$ (Red $\equiv$)

Tabella 5.1: Riduzione con le sostituzioni esplicite

Red Destr 1: $let\, x = g(M_1, \ldots, M_n)\, in\, P\, else\, Q \rightarrow P\{M/x\}$
$se\, g(M_1, \ldots, M_n)\, e'\, definita\, e\, vale\, M.$
In entrambi i casi la sostituzione computata in S è $\Theta_S = \{M/x\}$. *Se lo step S non usa nessuna di queste regole, allora non esegue nessuna sostituzione.*

Le sostituzione computate dagli step di una computazione AoT, caratterizzano tutte le assegnazioni alle variabili che avvengono durante la computazione. E' perciò possibile definire una differente, ma equivalente, esecuzione, in cui tutte le assegnazioni alle variabili non vengono applicate al processo in esecuzione, ma "registrate" in una sostituzione. Secondo questo punto di vista, una computazione viene eseguita sulla coppia $P * \Theta$, dove si usa * solo per leggibilità, per distinguerlo dalla applicazione della solita sostituzione $P\Theta$ o $(P)\Theta$. Le computazioni che seguono questa linea sono dette *computazioni con sostituzioni esplicite*. La tabella 5.1 contiene le regole di riduzione utilizzanti le sostituzioni esplicite.

La relazione di congruenza con le sostituzioni esplicite è la seguente:
$P * \Theta \equiv Q * \Theta \Leftrightarrow (P)\Theta \equiv (Q)\Theta$.

E' importante osservare che in una computazione con sostituzioni esplicite: $P_0 * \emptyset \rightarrow P_1 * \Theta_1 \rightarrow P_2 * \Theta_2 \ldots \rightarrow P_n * \Theta_n$, poiché le sostituzioni computate dalle sincronizzazioni o

dalle applicazioni di distruttori non vengono applicate al processo corrente, fino a che un operatore di replica non viene eseguito, le azioni di un qualsiasi processo P_i sono già presenti in P_0. Osserviamo il ruolo fondamentale giocato da AoT in questa definizione: se Θ_i contiene $\{M/x\}$ allora P_i contiene al massimo una variabile x alla quale tale assegnazione può venire applicata. Per questo fatto è possibile comporre tutte le sostituzioni calcolate dai differenti step (di sincronizzazione o di applicazione di distruttore) in una sostituzione globale, senza creare ambiguità per quanto riguarda le etichette delle variabili alle quali queste sostituzioni vengono applicate.

Lemma 2 *Consideriamo una computazione AoT con sostituzioni esplicite,* $P_0 * \emptyset \rightarrow P_1 * \Theta_1 \rightarrow P_2 * \Theta_2 \ldots \rightarrow P_n * \Theta_n$*, e assumiamo che ad ogni step* $i \in [1, n]$, $\Theta_i = \delta_i \circ \Theta_{i-1}$ *dove* $\delta_i = \{M_i/x_i\}$*, allora i seguenti fatti sono veri:*

1. *per* $i, j \in [1, n]$ *e* $i \neq j$, $x_i \neq x_j$*;*
2. *per* $i, j \in [1, n]$ *e* $i \neq j$, $x_i = M_j \Rightarrow (i < j)$.

Dimostrazione Il punto 1 è semplicemente una conseguenza del fatto che stiamo considerando computazioni AoT.

Per il punto 2 basta osservare che, per la definizione 10, x_i può essere l'oggetto o di un' azione di input, o l'oggetto dell'applicazione di un distruttore. Vediamo i due casi separatamente:

- In questo caso M_j può essere o l'oggetto di una azione di output o il termine assegnato da una applicazione di distruttore. Nel primo caso sia B l'azione di output. Nel momento in cui B viene eseguito, l'azione di input che riguarda x_i (sia A) si deve già essere sincronizzata con qualche azione di output (con oggetto M_i); per questo concludiamo che $i < j$. Anche nel secondo caso abbiamo che nel momento in cui l'applicazione di ditruttore viene eseguita, l'azione di input che riguarda x_i (sia A) si deve già essere sincronizzata con qualche azione di output (con oggetto M_i); e quindi anche in questo caso concludiamo che $i < j$.

- Nel caso in cui x_i sia l'oggetto dell'applicazione di un distruttore, la sostituzione $\{M_j/x_j\}$ può essere generata o da una sincronizzazione o da una applicazione di distruttore. Nel primo caso, l'azione di sincronizzazione avente come oggetto di output $M_j = x_i$, deve essere preceduta dall'applicazione del distrutture che genera la sostituzione $\{M_i/x_i\}$. Quindi $i < j$. Inoltre anche nel secondo caso, l'applicazione del distruttore che genera la sostituzione $\{M_j/x_j\}$ con $M_j = x_i$, deve essere preceduta dall'applicazione di distruttore che genera la sostituzione $\{M_i/x_i\}$. E quindi $i < j$.

C.v.d.

Il prossimo teorema mostra che le computazioni con sostituzioni esplicite sono semplicemente un modo diverso di rappresentare le computazioni AoT.

Teorema 4 *Esiste una computazione* $C : P_0 \rightarrow P_1 \rightarrow P_2 \ldots \rightarrow P_n \Leftrightarrow$ *esiste una computazione con sostituzioni esplicite* $C' : P_0 * \emptyset \rightarrow P'_1 * \Theta_1 \rightarrow P'_2 * \Theta_2 \ldots \rightarrow P'_n * \Theta_n$*, tale che per ogni* $i \in [1, n]$, $P_i = (P'_i)\Theta_i$.

Dimostrazione Lo dimostriamo per induzione sul numero di step di C e di C'. la base dell'induzione è triviale. Assumiamo quindi che l'induzione valga fino allo step $i-1$ ($i > 0$). Mostreremo che $P_i = (P'_i)\Theta_i$. Per ipotesi di induzione $P_{i-1} = (P'_{i-1})\Theta_{i-1}$. Consideriamo solo il caso in cui lo step $i-$esimo di una delle due computazioni consista solo di regole quali Red I/O o Red Destr 1. Infatti nelle altre regole non vi sono nè nuove sincronizzazioni, nè applicazioni di distruttore, e quindi non computano nuove sostituzioni alle variabili del processo $(i-1)-$esimo, quindi poiché lo step $i-$esimo dell'altra computazione può eseguire uno step $i-$esimo perfettamente corrispondente, abbiamo $\Theta_i = \Theta_{i+1}$. Notiamo inoltre che la regola Red $\equiv$ non crea problemi in quanto non può cambiare termini per α-conversione poiché entrambe le computazioni sono AoT. Consideriamo ora il caso in cui nello step $i-$esimo venga usata la regola Red I/O. Mostriamo separatamente le due direzioni dell'equivalenza. Supponiamo quindi che lo step $i-$esimo di C usi la regola Red I/O e siano $\overline{M} < N >$ e $M(x)$ le azioni che si sincronizzano. Per l'ipotesi di induzione ci sono espressioni concorrenti di P_{i-1} che possono eseguire l'azione corrispondente: $\overline{M'} < N' >$ e $M''(x)$. Per l'ipotesi di induzione $(M')\Theta_{i-1} = (M'')\Theta_{i-1} = M$ (nome), e perciò è possibile applicare la regola Red I/O nello step $P'_{i-1} * \Theta_{i-1} \rightarrow P'_i * \Theta_i$. Questo step computa la sostituzione $\{N'/x\}$ e quindi $\Theta_i = \{N'/x\} \circ \Theta_{i-1}$, e per ipotesi di induzione $(N')\Theta_{i-1} = N$, segue che $(x)\Theta_i = N$. Quindi $P_i = (P'_i)\Theta_i$. L'altra direzione è completamente analoga. Consideriamo ora il caso in cui nello step $i-$esimo venga usata la regola Red Destr1. Mostriamo separatamente le due direzioni dell'equivalenza. Sia $let\, x = g(M_1, \ldots, M_n)$ l'applicazione del distruttore in P_{i-1}. Per l'ipotesi di induzione esiste l'azione corrispondente in P'_{i-1}, sia $let\, x = g(M'_1, \ldots, M'_n)$, con $(M'_j)\Theta_{i-1} = M_j$ per ogni $j \in [1, n]$, risulta quindi possibile applicare la Red Destr 1 nello step $P'_{i-1} * \Theta_{i-1} \rightarrow P'_i * \Theta_i$. Tale applicazione computa la sostituzione $\{\sigma' M'/x\}$ ove $\sigma' M'$ è il valore assunto dal distruttore per una certa sostituzione σ'. Quindi $\Theta_i = \{\sigma' M'/x\} \circ \Theta_{i-1}$. Per ipotesi induttiva $(\sigma' M')\Theta_{i-1} = \sigma M$ (ove σM è il valore assunto dal distruttore nello step P_{i-1}), segue quindi che $(x)\Theta_i = \sigma M$. Perciò $(P'_i)\Theta_i = P_i$. L'altra direzione è completamente analoga.

C.v.d.

Capitolo 6

L'analisi statica

Basandoci sulle assunzioni del capitolo 5 introduciamo ora l'analisi statica. Il punto centrale dell'analisi è il fatto che vengono considerate tutte le azioni di comunicazione (Red I/O) che possono avvenire in qualsiasi possibile esecuzione del processo sotto analisi. Si considerano quindi solo le coppie che effettivamente si possono sincronizzare (e si valutano in un secondo tempo i test che le precedono). La definizione statica di una coppia di azioni che si possono sincronizzare durante l'esecuzione di un processo P è la seguente (diciamo che tale definizione è statica poiché è basata solo sul processo iniziale e non sulle computazioni che possono aver luogo da P):

Definizione 11 *Dato un processo P, le occorrenze in P di una azione di input A e di una azione di output B si dicono in posizioni concorrenti in P quando appartengono a due distinti sottoprocessi di P o quando appartengono a sottoprocessi di P della forma $!R$, cioè quando è presente un operatore di replica. Entrambe queste situazioni sono facilmente individuabili ragionando sull'albero sintattico di P. Questo albero ha azioni e test come foglie ed i suoi nodi interni sono etichettati con gli operatori $|$,., e $!$. Nell'ultimo caso le due azioni complementari sono entrambe nel sottoalbero con radice il nodo etichettato $!$.*

Definizione 12 *Con PAIR(P) si denota l'insieme di tutte le azioni di P che sono in posizioni concorrenti in P.*

Definizione 13 *Con COND(A) si denota l'insieme dei test*
$let\, x = g(M_1, \ldots, M_n)$ che precedono positivamente A in P.

Esempio 5 *Sia $P = A_0 \mid A_1.A_2.let\, x = g(M_1, \ldots, M_n)\, in\, (A_3.A_4)\ else\, (A_5.A_6)$ allora $COND(A_0)$=$COND(A_1)$=$COND(A_2)$=$COND(A_5)$= $COND(A_6)$=$\emptyset$ e $COND(A_3)$= $COND(A_4)$=$\{let\, x = g(M_1, \ldots, M_n)\}$.*

Definizione 14 *Si consideri un qualsiasi insieme finito $\mathcal{W} \subseteq \mathcal{T}$, si dice una associazione sui termini per $\mathcal{W}$ una funzione $\rho : \mathcal{W} \to \mathcal{PS}(\mathcal{W})$.*

Definizione 15 *Data l'associazione sui termini ρ per $\mathcal{W}$, diciamo che l'associazione sui termini ϕ_ρ per $\mathcal{W}$ è un raffinamento per ρ se $\forall w \in \mathcal{W}$, $\phi_\rho(w) = \{e\}$ con $e \in \rho(w)$ se $\rho(w) \neq \emptyset$ e $\phi_\rho(w) = \emptyset$ se $\rho(w) = \emptyset$.*

Definizione 16 *Si dice che il raffinamento ϕ_ρ per l'associazione sui termini ρ per $\mathcal{W}$ con $\{x, M_1, \ldots, M_n\} \in \mathcal{W}$, soddisfa il test let $x = g(M_1, \ldots, M_n)$ se esiste una sostituzione σ ed una equazione $g(M_1\text{'}, \ldots, M_n\text{'}) = M\text{'}$ in def(g) tali che $\phi_\rho(M_i) = \{\sigma M_i'\}$ per ogni $i \in \{\ 1, \ldots, n\ \}$.*

Definizione 17 *Si dice che l'associazione sui termini ρ per $\mathcal{W}$ con $\{x, M_1, \ldots, M_n\} \in \mathcal{W}$, soddisfa il test let $x = g(M_1, \ldots, M_n)$ se esiste un raffinamento ϕ_ρ che lo soddisfa. Inoltre sia K un insieme di test, diciamo che ρ soddisfa K se esiste un raffinamento ϕ_ρ che soddisfa tutti i test in K.*

Ora poiché l'analisi lavora con classi di equivalenza risulta conveniente introdurre una funzione C come segue: $C(x) = [x]$ per ogni $x \in \mathcal{V}$, $C(a) = [a]$ per ogni $a \in \mathcal{N}$, inoltre estendiamo tale funzione alle operazioni del processo applicandola ai loro termini.

Se P come al solito denota il processo sotto analisi , allora una associazione sui termini per P è qualsiasi associazione sui termini per $\mathcal{C}(\mathcal{T}(P))$. Questo insieme viene denotato con TA_P. Risulta facile vedere che TA_P forma un reticolo con l'ordine parziale $\sqsubseteq$, definito dall'inclusione puntuale: $\rho \sqsubseteq \rho'$ se per ogni $M \in \mathcal{C}(\mathcal{T}(P))$, $\rho(M) \subseteq \rho'(M)$. L'unione puntuale e l'intersezione sono rispettivamente le operazioni *join* e *meet* del reticolo $(TA_P, \sqsubseteq)$. Siamo interessati alle associazioni sui termini che sono soluzioni dell'analisi statica per P.

Definizione 18 *Sia P un processo, e ρ una associazione sui termini per P. Allora ρ è una soluzione per P se:*

1. $\forall a \in \mathcal{N}(P)$, $\rho([a]) = \{[a]\}$
2. $\forall x \in \mathcal{V}(P)$, $\rho([x]) \cap \mathcal{C}(\mathcal{V}(P)) = \emptyset$
3. $\forall f(M_1, \ldots, M_n) \in \mathcal{F}(P)\, tale\, che\, \forall i, i \in [1, n], \rho([M_i]) = \emptyset\, allora$ $\rho([f(M_1, \ldots, M_n]) = \emptyset$
4. $\forall f(M_1, \ldots, M_n) \in \mathcal{F}(P)\, tale\, che\, \forall i, i \in [1, n], \rho([M_i]) \neq \emptyset\, allora$ $\rho([f(M_1, \ldots, M_n)]) = \{f([N_1], \ldots, [N_n]) \mid [N_i] \in \rho([M_i]), i \in [1, n]\}$

5. $\forall\, let\, x = g(M_1, \ldots, M_n) \in P$ *(seguito da almeno un'azione che appartenga a qualche coppia in PAIR(P)), per cui esistono un raffinamento* ϕ_ρ*, una sostituzione* σ *ed una equazione* $g(M'_1, \ldots, M'_n) = M'$ *in* $def(g)$*, per cui* $\phi_\rho([M_i]) = \{\sigma M'_i\}\, \forall i \in [1, \ldots, n]$ *con* $g(\sigma M'_1, \ldots, \sigma M'_n) = \sigma M'$*, allora* $\rho([x]) \supseteq \phi_\rho([\sigma M'])$.

6. $\forall (A, B) \in PAIR(P)$*, sia* $A = \overline{M} < N >$ *e* $B = M'(x)$*, se* ρ *soddisfa* $\mathcal{C}(COND(A) \cup COND(B))$ *per il raffinamento* ϕ_ρ *e* $\phi_\rho([M]) \cap \phi_\rho([M']) \neq \emptyset$*, allora* $\rho([x]) \supseteq \phi_\rho([N])$.

Con S_P *denotiamo l'insieme delle soluzioni per P.*

Lemma 3 $(S_P, \sqsubseteq)$ *è una famiglia di Moore in* $(TA_P, \sqsubseteq)$*, cioè è chiuso per intersezione e contiene l'elemento top di* $(TA_P, \sqsubseteq)$.

Dimostrazione Siano $\rho_1, \rho_2 \in S_P$ e sia $\rho = \rho_1 \sqcap \rho_2$. Dobbiamo verificare che $\rho \in S_P$.

1. Poichè sia ρ_1 che ρ_2 soddisfano 1, anche ρ soddisfa 1, in quanto $\rho([a]) = \rho_1([a]) \cap \rho_2([a]) = \{[a]\}$.

2. $\rho([x]) = \rho_1([x]) \cap \rho_2([x])$ e $\rho([x]) \cap \mathcal{C}(\mathcal{V}(P)) = (\rho_1([x]) \cap \rho_2([x])) \cap \mathcal{C}(\mathcal{V}(P)) = (\rho_1([x]) \cap \mathcal{C}(\mathcal{V}(P))) \cap (\rho_2([x])) \cap \mathcal{C}(\mathcal{V}(P))) = \emptyset \cap \emptyset = \emptyset$

3. $\rho([f(M_1, \ldots, M_n)]) = \rho_1([f(M_1, \ldots, M_n)]) \cap \rho_2([f(M_1, \ldots, M_n)]) = \emptyset \cap \emptyset = \emptyset$, $\forall f(M_1, \ldots, M_n) \in \mathcal{F}(P)\, tale\, che\, \forall i, i \in [1, n], \rho([M_i]) = \emptyset$.

4. $\rho_1([f(M_1, \ldots, M_n)]) = \{f([N_1], \ldots, [N_n]) \mid [N_i] \in \rho_1([M_i]), i \in [1, n]\}$, $\rho_2([f(M_1, \ldots, M_n)]) = \{f([N_1], \ldots, [N_n]) \mid [N_i] \in \rho_2([M_i]), i \in [1, n]\}$ $\Rightarrow$ $\rho([f(M_1, \ldots, M_n)]) = (\rho_1([f(M_1, \ldots, M_n)]) \cap \rho_2(f[(M_1, \ldots, M_n)])) = \{f([N_1], \ldots, [N_n]) \mid [N_i] \in (\rho_1 \sqcap \rho_2)([M_i]), i \in [1, n]\}$, $\forall f(M_1, \ldots, M_n) \in \mathcal{F}(P)$ tale che $\forall i, i \in [1, n], (\rho_1 \sqcap \rho_2)([M_i]) \neq \emptyset$ e $(\rho_1 \sqcap \rho_2) = \rho$.

5. $\forall\, let\, x = g(M_1, \ldots, M_n) \in P$ (seguito da almeno un' azione che appartenga a qualche coppia in PAIR(P)), per cui esistono un raffinamento ϕ_ρ, una sostituzione σ ed una equazione $g(M'_1, \ldots, M'_n) = M'$ in $def(g)$, tali che $\phi_\rho([M_i]) = \{\sigma M'_i\}$ $(\forall i \in [1, n])$ e $g(\sigma M_1, \ldots, \sigma M_n) = \sigma M'$, allora ϕ_ρ è un raffinamento che soddisfa il test anche per ρ_1 e ρ_2 in quanto $\rho = \rho_1 \sqcap \rho_2$. Poiché: $\rho_1([x]) \supseteq \phi_\rho([\sigma M'])$ e $\rho_2([x]) \supseteq \phi_\rho([\sigma M'])$, abbiamo che $\rho([x]) = (\rho_1([x]) \cap \rho_2([x])) \supseteq \phi_\rho([\sigma M'])$.

6. $\forall (A, B) \in PAIR(P)$, sia $A = \overline{M} < N >$ *e* $B = M'(x)$, se esiste ϕ_ρ che soddisfa $\mathcal{C}(COND(A) \cup COND(B))$, e $\phi_\rho([M]) \cap \phi_\rho([M']) \neq \emptyset$ allora un tale ϕ_ρ è un raffinamento che soddisfa tali condizioni anche per ρ_1 e ρ_2. Quindi $\rho_1([x]) \supseteq \phi_\rho([N])$ e $\rho_2([x]) \supseteq \phi_\rho([N])$. Perciò $\rho([x]) = (\rho_1([x]) \cap \rho_2([x])) \supseteq \phi_\rho([N])$.

C.v.d.

E' da osservare che, per definizione, una famiglia di Moore ha almeno un elemento che denotiamo con $\sqcap S_P$ in quanto intersezione di tutte le soluzioni. Mostreremo che l'algoritmo di analisi statica applicato a P computa $\sqcap S_P$.

Presentiamo ora l'algoritmo di analisi statica. Se P è il processo che vogliamo analizzare, l'algoritmo calcola una associazione sui termini ρ su $\mathcal{C}(\mathcal{T}(P))$.

6.1 L'algoritmo

L'analisi parte con il calcolo degli insiemi $\mathcal{N}(P)$,$\mathcal{V}(P)$,$\mathcal{F}(P)$,$\mathcal{T}(P)$, computa PAIR(P), per ogni azione di ouput A ed ogni azione di input B calcola COND(A) e COND(B), infine esegue il calcolo della funzione $\rho_0(\mathcal{T}(P))$ come segue: per ogni nome $a \in \mathcal{N}(P)$, $\rho_0([a]) = \{[a]\}$, per ogni variabile $x \in \mathcal{V}(P)$, $\rho_0([x]) = \emptyset$, per ogni costruttore $f(M_1, \ldots, M_n) \in \mathcal{F}(P)$, $\rho_0(f([M_1], \ldots, [M_n])) = \emptyset$ se esiste un $i \in \{1, \ldots, n\}$ tale che $\rho_0([M_i]) = \emptyset$ altrimenti, se non esiste un tale i, $\rho_0(f([M_1], \ldots, [M_n])) = \{f([N_1], \ldots, [N_n]) \mid N_i \in \rho_0([M_i])\, e\, i \in \{1, \ldots, n\}\}$. Pone $i = 1$ e ripete il seguente step finché non si ferma.

Step:

1. poni $\rho_i = \rho_{i-1}$.

2. Per ogni (A,B)$\in$PAIR(P), siano A=$\overline{M} < N >$ e B=$M'(x)$. Per ogni singolo test $let\, y = g(M_1, \ldots, M_n)$ in COND(A) esegui Verifica-test. Per ogni singolo test $let\, y = g(M_1, \ldots, M_n)$ in COND(B) esegui Verifica-test.

   Se ρ_{i-1} soddisfa $C(COND(A) \cup COND(B))$ per una certa $\phi_{\rho_{i-1}}$ e $\phi_{\rho_{i-1}}([M]) \cap \phi_{\rho_{i-1}}([M']) \neq \emptyset$ allora poni

   $\rho_i([x]) = \rho_i([x]) \cup \phi_{\rho_{i-1}}([N])$.

3. Per ogni $f([M_1], \ldots, [M_n]) \in \mathcal{F}(P)$ se per ogni $i \in [1, n]$ $\rho_{i-1}([M_i]) \neq \emptyset$ allora poni $\rho_i(f([M_1], \ldots, [M_n])) = \{f([N_1], \ldots, [N_n]) \mid [N_i] \in \rho_{i-1}([M_i])\ i \in [1, n]\}$.

4. Se $\rho_i = \rho_{i-1}$ allora poni $\rho = \rho_i$ e STOP altrimenti riesegui step.

Verifica-test:

- Verifica il test $let\, y = g(M_1, \ldots, M_n)$.

  Il test $let\, y = g([M_1], \ldots, [M_n])$ viene soddisfatto se esistono un raffinamento $\phi_{\rho_{i-1}}$, una sostituzione σ ed una equazione $g(M_1', \ldots, M_n') = M'$ in def(g) tali che $\phi_{\rho_{i-1}}([M_i]) = \{\sigma M_i'\}$ per ogni $i \in \{\ 1, \ldots, n\ \}$ con $g(\sigma M_1', \ldots, \sigma M_n') = \sigma M'$.

- Se il test viene soddisfatto allora per ognuna di tali terne poni $\rho_i([y]) = \rho_i([y]) \cup \phi_{\rho_{i-1}}([\sigma M'])$.

L'algoritmo in alcuni casi può non terminare in quanto gli insiemi di definizione dei distruttori possono essere infiniti, oppure perchè i costruttori sono tali da creare infiniti nuovi termini. Dimostriamo, comunque, che quando l'algoritmo termina allora ha certamente computato la soluzione minima per P.

Teorema 5 *L'associazione sui termini ρ che l'algoritmo di analisi statica computa quando riesce a terminare, è la soluzione minima per il processo P considerato.*

Dimostrazione Proviamo i seguenti punti:

1. La funzione ρ calcolata dall'algoritmo è un elemento di S_P;
2. Tale ρ è l'elemento minimo di S_P, cioè $\rho = \sqcap S_P$.

Procediamo con la dimostrazione di ciascun punto:

1. Affinché ρ sia una soluzione deve soddisfare i 6 punti della definizione 18. Il punto 1 è soddisfatto perchè $\rho_0([a]) = \{[a]\}$ e durante tutto l'agoritmo tale valore non viene mai modificato in quanto non si calcola mai il valore di ρ_i applicato ad un nome.

   Il punto 3 viene soddisfatto in quanto vale per ρ_0 per assegnazione iniziale, durante tutto l'algoritmo il valore della ρ_i applicato ad un costruttore che continui a soddisfare tali ipotesi non viene mai modificato (punto 3 dell'algoritmo) e quando l'algoritmo termina $\rho_i = \rho_{i-1}$.

   Il punto 4 viene soddisfatto in quanto vale per ρ_0, durante tutto l'algoritmo il valore della ρ_i applicato ad un costruttore che soddisfi tali ipotesi viene modificato dal punto 3 dell'algoritmo e poiché quando termina $\rho_i = \rho_{i-1}$, l'uguaglianza rimane vera.

   Il punto 5 viene soddisfatto dalla routine Verifica-test, tenendo presente che quando l'algoritmo termina $\rho_i = \rho_{i-1}$.

   Il punto 6 viene soddisfatto dal punto 2 dell'algoritmo sempre tenendo presente che alla fine $\rho_i = \rho_{i-1}$.

   Il punto 2 viene soddisfatto in quanto per ogni i è vero che $\rho_i([x]) \cap \mathcal{C}(\mathcal{V}(P)) = \emptyset$ Lo dimostriamo per induzione. Per $i = 0$ abbiamo che $\rho_0([x]) = \emptyset$, e quindi il passo base vale. Supponiamo allora che l'affermazione sia vera per $i \leq n$ e

dimostriamola per $i+1$. Ora $\rho_{i+1}([x]) \subseteq \rho_i([x]) \cup_j \{\rho_i([T_j])\}$ ove i T_j sono i termini che la variabile può ricevere da applicazioni di distruttore o da sincronizzazioni. Ora $\rho_i([T_j])$ per induzione e per i punti 1,3,4 della Definizione 18 (che abbiamo visto essere soddisfatti) non può contenere variabili e nemmeno $\rho_i([x])$ per induzione può contenere variabili, e quindi $\rho_{i+1}([x])$ non contiene variabili.

2. Dimostreremo per induzione sul numero di iterazioni i effettuate dall'algoritmo che, per ogni i, $\rho_i \sqsubseteq \sqcap S_P$, ed in particolare che $\rho \sqsubseteq \sqcap S_P$ che, assieme al punto precedente, implica che $\rho = \sqcap S_P$. La base dell'induzione è $i = 0$. Chiaramente $\rho_0 \sqsubseteq \sqcap S_P$ perchè:

   $\rho_0(a) = S_P(a)$ con $a \in C(\mathcal{N}(P))$;

   $\rho_0(x) = \emptyset \subseteq S_P(x)$ con $x \in C(\mathcal{V}(P))$;

   $\rho_0(f(M_1, \ldots, M_n)) \subseteq S_P(f(M_1, \ldots, M_n))\, con\, f(M_1, \ldots, M_n) \in C(\mathcal{F}(P))$.

   Assumiamo che $\rho_{i-1} \sqsubseteq \sqcap S_P$, dobbiamo provare che anche $\rho_i \sqsubseteq \sqcap S_P$. Abbiamo bisogno di una seconda induzione sulla sequenza delle coppie in $\mathcal{C}(PAIR(P))$ che vengono considerate durante lo step i-esimo dell'algoritmo di analisi statica. (Ci riferiremo alla prima induzione con il termine $1^a - ind$). Assumiamo che $\rho_i \sqsubseteq \sqcap S_P$ rimanga vera per un numero fissato di coppie analizzate, e che la prossima coppia sotto analisi sia $(A, B) = \mathcal{C}(A', B')$, $con\, A = \overline{M} < N > e\, B = M'(x)$. Chiaramente l'analisi di (A, B) può modificare ρ_i se almeno uno dei test $let\, y = g(M_1, \ldots, M_n)$ in $C(COND(A) \cup COND(B))$ viene soddisfatto. Per ognuno di tali test soddisfatti (sia $\phi_{\rho_{i-1}}$ un raffinamento che lo soddisfa) l'algoritmo computa la seguente assegnazione:

   $\rho_i(y) = \rho_i(y) \cup \phi_{\rho_{i-1}}(\sigma M')$

   per ogni $\sigma M'$ calcolato nella procedura Verifica-test. Inoltre se tutti i test vengono soddisfatti da un unico $\phi_{\rho_{i-1}}$ tale che

   $\phi_{\rho_{i-1}}(M) \cap \phi_{\rho_{i-1}}(M') \neq \emptyset$

   abbiamo che :

   $\rho_i(x) = \rho_i(x) \cup \phi_{\rho_{i-1}}(N)$.

   Dobbiamo dimostrare che dopo queste assegnazioni effettuate per ogni test che viene verificato, rimane vero che:

   $\rho_i(y) \subseteq \sqcap S_P(y)$,

ed inoltre se tutti i test vengono soddisfatti per un certo $\phi_{\rho_{i-1}}$ e $\phi_{\rho_{i-1}}(M) \cap \phi_{\rho_{i-1}}(M') \neq \emptyset$, dobbiamo dimostrare che rimane vero:

$\rho_i(x) \subseteq \sqcap S_P(x)$.

A tal fine, per le ipotesi della seconda induzione, basta mostrare che:

$\phi_{\rho_{i-1}}(\sigma M') \subseteq \sqcap S_P(y)$ e che (se tutti i test vengono soddisfatti da un unico raffinamento per il quale la sincronizzazione può avvenire)

$\phi_{\rho_{i-1}}(N) \subseteq \sqcap S_P(x)$.

Mostriamo questo fatto come segue: per le ipotesi della $1^a - ind$, $\rho_{i-1} \sqsubseteq \sqcap S_P$ e perciò se ρ_{i-1} soddisfa qualcuno (o tutti) i test visti, anche $\sqcap S_P$ li soddisfa. Ora poichè $\sqcap S_P$ è una soluzione, abbiamo che:

$\sqcap S_P(N) \subseteq \sqcap S_P(x)$ e

$\sqcap S_P(\sigma M') \subseteq \sqcap S_P(y)$.

Perciò per le ipotesi della $1^a - ind$ troviamo:

$\phi_{\rho_{i-1}}(\sigma M') \subseteq \sqcap S_P(y)$ e che

$\phi_{\rho_{i-1}}(N) \subseteq \sqcap S_P(x)$

come desiderato.

C.v.d.

6.2 Correttezza dell'analisi

Per dimostrare la correttezza dell'algoritmo di analisi statica, useremo la nozione di computazioni con sostituzioni esplicite [16]. Questo può essere fatto senza perdita di generalità per il Teorema 4.

Lemma 4 *Consideriamo una computazione AoT con sostituzioni esplicite,* $P_0 * \emptyset \rightarrow P_1 * \Theta_1 \ldots \rightarrow P_n * \Theta_n$, *allora per ogni* $i \in [1, n]$*:*
$\mathcal{C}(PAIR(P_0)) \supseteq \mathcal{C}(PAIR(P_i))$*;*
e se denotiamo con TEST(P) l'insieme di tutti i test in P, allora:
$\mathcal{C}(TEST(P_0)) \supseteq \mathcal{C}(TEST(P_i))$.

Dimostrazione Se in P_0 non è presente nessun operatore di replica, il risultato è semplice. Osserviamo le regole nella Tabella 5.1, è immediato verificare che nessuna di queste regole può introdurre nuove coppie di azioni concorrenti o nuovi operatori di

composizione parallela e nemmeno nuovi test. Inoltre poiché consideriamo computazioni AoT, questo ci garantisce che non vengono applicate α-conversioni. In questo caso la funzione $\mathcal{C}$ è ininfluente: la conclusione vale anche senza applicare $\mathcal{C}$. Al contrario, tale funzione è importante quando P_0 contiene il sottoprocesso $!R$. In questo caso, per la definizione 9, ogni copia di R ha i propri nomi e le proprie variabili rinominate nella stessa classe di equivalenza dei corrispondenti nomi e delle corrispondenti variabili usate in R. Perciò, per ognuna di tali copie R', è vero che $\mathcal{C}(R) = \mathcal{C}(R')$. Quindi, copie multiple di R non producono nuove azioni rispetto ad una copia (quando $\mathcal{C}$ viene applicata), e perciò non possono introdurre nuove coppie concorrenti e nemmeno nuovi test.

C.v.d.

Il lemma precedente ha un'importante conseguenza:

Corollario 1 *Consideriamo una qualsiasi computazione (con sostituzioni esplicite) come nel Lemma 4, supponiamo che nello step $i-$esimo vengano applicate o la regola Red I/O o la regola Red Destr 1. Nel primo caso sia (A,B) la coppia di azioni che si sincronizzano, allora:*

$\mathcal{C}(A, B) \in \mathcal{C}(PAIR(P_0))$;

nel secondo caso sia $let\, x = g(M_1, \ldots, M_n)$ il test che viene soddisfatto, allora:

$\mathcal{C}(let\, x = g(M_1, \ldots, M_n)) \in \mathcal{C}(TEST(P_0))$.

Dimostrazione Entrambi i risultati seguono direttamente dal Lemma 4, infatti nel primo caso la coppia (A,B) si sincronizza in Red I/O quindi A e B sono in posizioni concorrenti in P_{i-1}; nel secondo caso poiché viene applicata la RedDestr1 allora $let\, x = g(M_1, \ldots, M_n) \in TEST(P_{i-1})$.

C.v.d.

Per dimostrare la correttezza dell'algoritmo di analisi statica, abbiamo bisogno di un ultimo risultato tecnico che ci dice che durante una computazione, quando una azione A è pronta per essere eseguita, tutti i test in COND(A) devono essere soddisfatti.

Lemma 5 *Consideriamo una computazione (con sostituzioni esplicite) $P_0 * \emptyset \rightarrow P_1 * \Theta_1 \ldots \rightarrow P_n * \Theta_n$, e sia A l'azione di P_{i-1} eseguita durante lo step $i-$esimo; allora deve essere vero che Θ_{i-1} soddisfi tutti i test in COND(A).*

Dimostrazione Basta osservare che prima che A venga eseguita tutti i test i COND(A) devono essere eliminati, e questo fatto è possibile solo se la corrente sostituzione permette di dire che il distruttore del test in considerazione è definito (in tale caso si aggiorna poi la sostituzione). Rimane da verificare che se Θ_i soddisfa un test, allora anche Θ_{i+k} lo soddisfa (per qualsiasi $k > 0$), e questo fatto è una facile conseguenza del Lemma 2.

C.v.d.

Teorema 6 *Consideriamo un processo P_0, e sia ρ la funzione computata dall'algoritmo di analisi statica con P_0 come input, allora per ogni computazione AoT con sostituzioni esplicite $P_0 * \emptyset \rightarrow P_1 * \Theta_1 \ldots \rightarrow P_n * \Theta_n$, vale:*

1. *Per ogni nome $a \in \mathcal{N}(P_i)$, $i \in [1, n]$, $[(a)\Theta_i] = \rho([a])$.*
2. *Per ogni variabile $x \in \mathcal{V}(P_i)$, $i \in [1, n]$, se in Θ_i ci sono sostituzioni che riguardano tale variabile, e tali sostituzioni non sono generate applicazioni di distruttori che non precedono un'azione contenuta in qualche coppia di PAIR(P_0), allora tali sostituzioni sono incluse in $\rho([x])$ cioè:*

   $\forall x \in \mathcal{V}(P_i)$, $(x)\Theta_i \neq x \Rightarrow [(x)\Theta_i] \in \rho([x])$.
3. *per ogni costruttore $f(M_1, \ldots, M_n) \in \mathcal{F}(P_i)$, $i \in [1, n]$, tale che per ogni $M_i \in \mathcal{V}(P)$, $(M_i)\Theta_i \neq M_i$ ($i \in [1, n]$) e tali sostituzioni non sono generate da applicazioni di distruttori che non precedono un'azione contenuta in qualche coppia di PAIR(P_0), allora $[(f(M_1, \ldots, M_n))\Theta_i] = [f((M_1)\Theta_i, \ldots, (M_n)\Theta_i)] \in \rho([f(M_1, \ldots, M_n)])$.*

Dimostrazione

1. Tale punto viene dal fatto che ai nomi non vengono mai eseguite sostituzioni e che per ogni nome a, $\rho([a]) = [a]$.
2. Lo dimostriamo per induzione. La sostituzione iniziale è vuota e perciò per ogni $x \in \mathcal{V}(P_0)$, $(x)\emptyset = x$ quindi la tesi è banalmente vera per $i = 0$. Nell'analisi degli step successivi consideriamo solo quelli che usano le regole Red I/O e RedDestr 1. Il motivo è il seguente: consideriamo un qualsiasi step $P_i * \Theta_i \rightarrow P_{i+1} * \Theta_{i+1}$ che non usi nè Red I/O nè Red Destr 1, allora tale step non crea nessuna nuova sostituzione e perciò $\Theta_i = \Theta_{i+1}$. Eventualmente un tale step può introdurre in P_{i+1} nuovi termini (tramite le regole di congruenza ove $!P \equiv P \mid !P$) ma solo se tali termini appartengono a classi di equivalenza già presenti. Inoltre solo la Red I/O e la RedDestr 1 possono istanziare tali termini, perciò finchè tali regole non vengono usate rimane vero che $(x)\Theta_i = (x)\Theta_{i+1}$, per ogni variabile x già presente, e che $(x')\Theta_{i+1} = x'$, per ogni nuova variabile introdotta. Osserviamo che in tale caso $\rho([x'])$ è definita dall'algoritmo di analisi statica perchè esiste $x \in \mathcal{V}(P_0)$ tale che $[x] = [x']$. Consideriamo un passo $P_i * \Theta_i \rightarrow P_{i+1} * \Theta_{i+1}$ che contenga la regola Red I/O (il caso della regola RedDestr 1 lo consideriamo successivamente). Siano $A =$

$\overline{N} < M >$ e $B = N'(x)$ le azioni che si sincronizzano. Per il Lemma 5, Θ_i soddisfa tutte i test in $COND(A) \cup COND(B)$ ed inoltre $(N)\Theta_i = (N')\Theta_i$ con $(N)\Theta_i$ nome. Per l'ipotesi induttiva anche ρ soddisfa tutti i test in $COND(A) \cup COND(B)$ per una certa ϕ_ρ tale che $\phi_\rho(N) \cap \phi_\rho(N') \neq \emptyset$, quindi la coppia $\mathcal{C}(A, B)$ viene considerata dall'algoritmo e perciò $\phi_\rho(M) \subseteq \rho(x)$. Ora per ipotesi induttiva abbiamo che $\{(M)\Theta_i\} = \phi_\rho(M)$ per qualche ϕ_ρ, infatti se $[M] \in C(\mathcal{V}(P_i)) \Rightarrow [(M)\Theta_i] \in \rho(M)$ e quindi consideriamo il raffinamento tale per cui $\phi_\rho([M]) = \{[(M)\Theta_i]\}$; se $[M] \in C(\mathcal{N}(P_i))$ per il punto 1 abbiamo che $\phi_\rho([M]) = \{[(M)\Theta_i]\} = \rho([M])$; se $[M] \in C(\mathcal{F}(P_i))$ sia $[M] = [f(N_1, \ldots, N_n)]$, abbiamo che, per ipotesi induttiva e per il Teorema 5, $\phi_\rho([M]) = \{[(M)\Theta_i]\}$ per qualche ϕ_ρ.

Perciò $[(x)\{M/x\} \circ \Theta_i] \in \rho(x)$.

Consideriamo un passo $P_i * \Theta_i \rightarrow P_{i+1} * \Theta_{i+1}$ che applichi la regola RedDestr 1 e che tale applicazione sia seguita da un'azione che appartenga ad una coppia in PAIR(P_0). Sia $let\, x = g(M_1, \ldots, M_n)$ il test che viene soddisfatto da Θ_i. Per il Lemma 5, Θ_i soddisfa tutti i test che precedono tale azione, ed inoltre $g(M_1, \ldots, M_n) * \Theta_i$ risulta definita e supponiamo valga M. Ora per ipotesi induttiva abbiamo che $\{(M)\Theta_i\} = \phi_\rho(M)$ per qualche ϕ_ρ, infatti se $[M] \in C(\mathcal{V}(P_i)) \Rightarrow [(M)\Theta_i] \in \rho(M)$ e quindi consideriamo il raffinamento tale per cui $\phi_\rho([M]) = \{[(M)\Theta_i]\}$; se $[M] \in C(\mathcal{N}(P_i))$ per il punto 1 abbiamo che $\phi_\rho([M]) = \{[(M)\Theta_i]\} = \rho([M])$; se $[M] \in C(\mathcal{F}(P_i))$ sia $[M] = [f(N_1, \ldots, N_n)]$, abbiamo che, per ipotesi induttiva e per il Teorema 5, $\phi_\rho([M]) = \{[(M)\Theta_i]\}$ per qualche ϕ_ρ.

Perciò $[(x)\{M/x\} \circ \Theta_i] \in \rho(x)$.

3. Tale punto è una conseguenza dei due punti precedenti e del Teorema 5.

C.v.d.

Esempio 6 *Applichiamo l'algoritmo di analisi statica al protocollo di comunicazione proposto in [3]. Modifichiamo tale protocollo in modo che soddisfi le ipotesi del cap. 6. Sia*

$P = (\nu sK_A)(\nu sK_B)\, let\, pK_A = id(pk(sK_A))\, in\, let\, pK_B = id(pk(sK_B))\, in\, (A \mid B)$,

dove

$A = (\nu k)\overline{e} < pencrypt((k, pK_A)), pK_B)) > .e(z_a).\ let\, (x_a, y_a) = pdecrypt(z_a, sK_a)\, in\ let\, z = equals(x_a, k)\, in\ \overline{e} < sencrypt(s, y_a) >$,

e

$B = e(z_b).let\, (x_b, y_b) = pdecrypt(z_b, sK_B)\, in\, (\nu K_{AB}).\ \overline{e} < pencrypt((x_b, K_{AB}), y_b)) > .e(z_{b'})$.

Questo protocollo stabilisce una sessione di comunicazione tra A e B con chiave K_{AB} che usa per inviare il messaggio segreto s da A a B. Si basa sulle chiavi pubbliche pK_A per A e pK_B per B. Per prima cosa A crea una 'nonce' (k) che invia a B assieme alla sua chiave pubblica, criptati con la chiave pubblica di B. Poi B risponde con lo stesso 'nonce' e la chiave K_{AB}, criptati con la chiave pubblica di A. Quando A riceve questo messaggio, riconosce k; è così sicuro che la chiave K_{AB} è stata creata da B. Infine A manda il messaggio segreto s criptato con chiave K_{AB}. (Si è usato $let\,(x,y) = M\,in\,Q$ piuttosto che $let\,z = M\,in\,let\,x = 1th_2(z)\,in\,let\,y = 2th_2(z)\,in\,Q$ usando il pattern-matching sulle n-uple. Le chiavi sK_A e sK_B sono le chavi di decriptazione che corrispondono a pK_A e pK_B, rispettivamente. e è un canale pubblico.)

Vediamo ora come lavora l'analisi statica su P.

- $\mathcal{N}(P) = \{e, sK_A, sK_B, k, s, K_{AB}\}$;
  $\mathcal{V}(P) = \{pK_A, pK_B, z_a, x_a, y_a, z, z_b, x_b, y_b, z_{b'}\}$;
  $\mathcal{F}(P) = \{pk(sK_A), pk(sK_B), pencrypt((k, pK_A), pK_B),$
  $sencrypt(s, y_a), pencrypt((x_b, K_{AB}), y_b)\}$.

- $PAIR(P) = \{\ (\overline{e} < pencrypt((k, pK_A), pK_B) >, e(z_b))$;
  $(\overline{e} < pencrypt((k, pK_A), pK_B) >, e(z_{b'}))$;
  $(\overline{e} < sencrypt(s, y_a) >, e(z_b))$;
  $(\overline{e} < sencrypt(s, y_a) >, e(z_{b'}))$;
  $(\overline{e} < pencrypt((x_b, K_{AB}), y_b) >, e(z_a))\}$

- $COND(\overline{e} < pencrypt((k, pK_A), pK_B) >) = \{let\,pK_A = id(pk(sK_A)), let\,pK_B = id(pk(sK_B))\}$;
  $COND(e(z_a)) = \{let\,pK_A = id(pk(sK_A)), let\,pK_B = id(pk(sK_B))\}$;
  $COND(e(z_b)) = \{let\,pK_A = id(pk(sK_A)), let\,pK_B = id(pk(sK_B))\}$;
  $COND(\overline{e} < sencrypt(s, y_a) >) = \{let\,pK_A = id(pk(sK_A)), let\,pK_B = id(pk(sK_B)),$
  $let\,(x_a, y_a) = pdecrypt(z_a, sK_A), let\,z = equals(x_a, k)\}$;
  $COND(\overline{e} < pencrypt((x_b, K_{AB}), y_b) >) = \{let\,pK_A = id(pk(sK_A)), let\,pK_B = id(pk(sK_B)), let\,(x_b, y_b) = pdecrypt(z_b, sK_B)\}$;
  $COND(e(z_{b'})) = \{let\,pK_A = id(pk(sK_A)), let\,pK_B = id(pk(sK_B)),$
  $let\,(x_b, y_b) = pdecrypt(z_b, sK_B)\}$;
  tutte le altre azioni non sono precedute da test.

- *Calcoliamo ρ_0:*

$\rho_0([c] = \{[e]\};$ $\rho_0([sK_A] = \{[sK_A]\};$
$\rho_0([k] = \{[k]\};$ $\rho_0([sK_B] = \{[sK_B]\};$
$\rho_0([s] = \{[s]\};$ $\rho_0([K_{AB}] = \{[K_{AB}]\};$
$\rho_0([pK_A]) = \emptyset;$ $\rho_0([pK_B] = \emptyset;$
$\rho_0([z_a] = \emptyset;$ $\rho_0([x_a] = \emptyset;$
$\rho_0([y_a] = \emptyset;$ $\rho_0([z] = \emptyset;$
$\rho_0([z_b] = \emptyset;$ $\rho_0([x_b] = \emptyset;$
$\rho_0([y_b] = \emptyset;$ $\rho_0([z_{b'}] = \emptyset;$
$\rho_0([\rho_0([pk(sK_A)]) = \{pk([sK_A])\};$ $\rho_0([pk(sK_B)]) = \{pk([sK_B])\};$
$\rho_0([pencrypt((k, pK_A), pK_B)]) = \emptyset;$ $\rho_0([sencrypt(s, y_a)]) = \emptyset;$
$\rho_0([pencrypt((x_b, K_{AB}), y_b)]) = \emptyset;$

- *Calcoliamo ρ_1:*

$\rho_1([c] = \{[e]\};$ $\rho_1([sK_A] = \{[sK_A]\};$
$\rho_1([k] = \{[k]\};$ $\rho_1([sK_B] = \{[sK_B]\};$
$\rho_1([s] = \{[s]\};$ $\rho_1([K_{AB}] = \{[K_{AB}]\};$
$\rho_1([\mathbf{pK_A}]) = \{\mathbf{pk}([\mathbf{sK_A}])\};$ $\rho_1([\mathbf{pK_B}] = \{\mathbf{pk}([\mathbf{sK_B}])\}$ *;*
$\rho_1([z_a] = \emptyset;$ $\rho_1([x_a] = \emptyset;$
$\rho_1([y_a] = \emptyset;$ $\rho_1([z] = \emptyset;$
$\rho_1([z_b] = \emptyset;$ $\rho_1([x_b] = \emptyset;$
$\rho_1([y_b] = \emptyset;$ $\rho_1([z_{b'}] = \emptyset;$
$\rho_1([pk(sK_A)]) = \{pk([sK_A])\};$ $\rho_1([pk(sK_B)]) = \{pk([sK_B])\};$
$\rho_1([pencrypt((k, pK_A), pK_B)]) = \emptyset;$ $\rho_1([sencrypt(s, y_a)]) = \emptyset;$
$\rho_1([pencrypt((x_b, K_{AB}), y_b)]) = \emptyset;$

- *Calcoliamo ρ_2:*

$\rho_2([c] = \{[e]\};$ $\rho_2([sK_A] = \{[sK_A]\};$
$\rho_2([k] = \{[k]\};$ $\rho_2([sK_B] = \{[sK_B]\};$
$\rho_2([s] = \{[s]\};$ $\rho_2([K_{AB}] = \{[K_{AB}]\};$
$\rho_2([pK_A]) = \{pk([sK_A])\};$ $\rho_2([pK_B] = \{pk([sK_B])\};$
$\rho_2([z_a] = \emptyset;$ $\rho_2([x_a] = \emptyset;$
$\rho_2([y_a] = \emptyset;$ $\rho_2([z] = \emptyset;$
$\rho_2([z_b] = \emptyset;$ $\rho_2([x_b] = \emptyset;$
$\rho_2([y_b] = \emptyset;$ $\rho_2([z_{b'}] = \emptyset;$
$\rho_2([pk(sK_A)]) = \{pk([sK_A])\};$ $\rho_2([pk(sK_B)]) = \{pk([sK_B])\};$
$\rho_2([\mathbf{pencrypt}((\mathbf{k}, \mathbf{pK_A}), \mathbf{pK_B})]) = \{\mathbf{pencrypt}(([\mathbf{k}], \mathbf{pk}([\mathbf{sK_A}])), \mathbf{pk}([\mathbf{sK_B}]))\};$
$\rho_2([sencrypt(s, y_a)]) = \emptyset;$ $\rho_2([pencrypt((x_b, K_{AB}), y_b)]) = \emptyset;$

- *Calcoliamo ρ_3:*

$\rho_3([c] = \{[e]\};$ $\rho_3([sK_A] = \{[sK_A]\};$
$\rho_3([k] = \{[k]\};$ $\rho_3([sK_B] = \{[sK_B]\};$
$\rho_3([s] = \{[s]\};$ $\rho_3([K_{AB}] = \{[K_{AB}]\};$
$\rho_3([pK_A]) = \{pk([sK_A])\};$ $\rho_3([pK_B] = \{pk([sK_B])\};$
$\rho_3([z_a] = \emptyset;$ $\rho_3([x_a] = \emptyset;$
$\rho_3([y_a] = \emptyset;$ $\rho_3([z] = \emptyset;$
$\rho_3([\mathbf{z_b}] = \{\mathbf{pencrypt}(([\mathbf{k}], \mathbf{pk}([\mathbf{sK_A}])), \mathbf{pk}([\mathbf{sK_B}]))\};$
$\rho_3([y_b] = \emptyset;$ $\rho_3([z_{b'}] = \emptyset;$
$\rho_3([x_b] = \emptyset;$
$\rho_3([pk(sK_A)]) = \{pk([sK_A])\};$ $\rho_3([pk(sK_B)]) = \{pk([sK_B])\};$
$\rho_3([pencrypt((k, pK_A), pK_B)]) = \{pencrypt(([k], pk([sK_A])), pk([sK_B]))\};$
$\rho_3([sencrypt(s, y_a)]) = \emptyset;$ $\rho_3([pencrypt((x_b, K_{AB}), y_b)]) = \emptyset;$

- *Calcoliamo ρ_4:*

$\rho_4([c] = \{[e]\};$ $\rho_4([sK_A] = \{[sK_A]\};$
$\rho_4([k] = \{[k]\};$ $\rho_4([sK_B] = \{[sK_B]\};$
$\rho_4([s] = \{[s]\};$ $\rho_4([K_{AB}] = \{[K_{AB}]\};$
$\rho_4([pK_A]) = \{pk([sK_A])\};$ $\rho_4([pK_B] = \{pk([sK_B])\};$
$\rho_4([z_a] = \emptyset;$ $\rho_4([x_a] = \emptyset;$
$\rho_4([y_a] = \emptyset;$ $\rho_4([z] = \emptyset;$
$\rho_4([z_b] = \{pencrypt(([k], pk([sK_A])), pk([sK_B]))\};$
$\rho_{\mathbf{4}}([\mathbf{z_{b'}}] = \{\mathbf{pencrypt}(([\mathbf{k}], \mathbf{pk}([\mathbf{sK_A}])), \mathbf{pk}([\mathbf{sK_B}]))\};$
$\rho_{\mathbf{4}}([\mathbf{x_b}] = \{[\mathbf{k}]\};$
$\rho_{\mathbf{4}}([\mathbf{y_b}] = \{\mathbf{pk}([\mathbf{sK_A}])\};$
$\rho_4([pk(sK_A)]) = \{pk([sK_A])\};$ $\rho_4([pk(sK_B)]) = \{pk([sK_B])\};$
$\rho_4([pencrypt((k, pK_A), pK_B)]) = \{pencrypt(([k], pk([sK_A])), pk([sK_B]))\};$
$\rho_4([sencrypt(s, y_a)]) = \emptyset;$ $\rho_4([pencrypt((x_b, K_{AB}), y_b)]) = \emptyset;$

- *Calcoliamo ρ_5:*

$\rho_5([c] = \{[e]\};$ $\rho_5([sK_A] = \{[sK_A]\};$
$\rho_5([k] = \{[k]\};$ $\rho_5([sK_B] = \{[sK_B]\};$
$\rho_5([s] = \{[s]\};$ $\rho_5([K_{AB}] = \{[K_{AB}]\};$
$\rho_5([pK_A]) = \{pk([sK_A])\};$ $\rho_5([pK_B] = \{pk([sK_B])\};$
$\rho_5([z_a] = \emptyset;$ $\rho_5([x_a] = \emptyset;$
$\rho_5([y_a] = \emptyset;$ $\rho_5([z] = \emptyset;$
$\rho_5([z_b] = \{pencrypt(([k], pk([sK_A])), pk([sK_B]))\};$
$\rho_5([z_{b'}] = \{pencrypt(([k], pk([sK_A])), pk([sK_B]))\};$
$\rho_5([x_b] = \{[k]\};$
$\rho_5([y_b] = \{pk([sK_A])\};$
$\rho_5([pk(sK_A)]) = \{pk([sK_A])\};$ $\rho_5([pk(sK_B)]) = \{pk([sK_B])\};$
$\rho_5([pencrypt((k, pK_A), pK_B)]) = \{pencrypt(([k], pk([sK_A])), pk([sK_B]))\};$
$\rho_{\mathbf{5}}([\mathbf{pencrypt}((\mathbf{x_b}, \mathbf{K_{AB}}), \mathbf{y_b})]) = \{\mathbf{pencrypt}(([\mathbf{k}], [\mathbf{K_{AB}}]), \mathbf{pk}([\mathbf{sK_A}]))\};$
$\rho_5([sencrypt(s, y_a)]) = \emptyset;$

- *Calcoliamo ρ_6:*

$\rho_6([c] = \{[e]\};$ $\rho_6([sK_A] = \{[sK_A]\};$
$\rho_6([k] = \{[k]\};$ $\rho_6([sK_B] = \{[sK_B]\};$
$\rho_6([s] = \{[s]\};$ $\rho_6([K_{AB}] = \{[K_{AB}]\};$
$\rho_6([pK_A]) = \{pk([sK_A])\};$ $\rho_6([pK_B] = \{pk([sK_B])\};$
$\boldsymbol{\rho_6([\mathbf{z_a}] = \mathbf{pencrypt}(([\mathbf{k}], [\mathbf{K_{AB}}]), \mathbf{pk}([\mathbf{sK_A}]));}$
$\rho_6([y_a] = \emptyset;$ $\rho_6([z] = \emptyset;$
$\rho_6([z_b] = \{pencrypt(([k], pk([sK_A])), pk([sK_B]))\};$
$\rho_6([z_{b'}] = \{pencrypt(([k], pk([sK_A])), pk([sK_B]))\};$
$\rho_6([x_b] = \{[k]\};$
$\rho_6([y_b] = \{pk([sK_A])\};$ $\rho_6([x_a] = \emptyset;$
$\rho_6([pk(sK_A)]) = \{pk([sK_A])\};$ $\rho_6([pk(sK_B)]) = \{pk([sK_B])\};$
$\rho_6([pencrypt((k, pK_A), pK_B)]) = \{pencrypt(([k], pk([sK_A])), pk([sK_B]))\};$
$\rho_6([pencrypt((x_b, K_{AB}), y_b)]) = \{pencrypt(([k], [K_{AB}]), pk([sK_A]))\};$
$\rho_6([sencrypt(s, y_a)]) = \emptyset;$

- *Calcoliamo ρ_7:*

$\rho_7([c] = \{[e]\};$ $\rho_7([sK_A] = \{[sK_A]\};$
$\rho_7([k] = \{[k]\};$ $\rho_7([sK_B] = \{[sK_B]\};$
$\rho_7([s] = \{[s]\};$ $\rho_7([K_{AB}] = \{[K_{AB}]\};$
$\rho_7([pK_A]) = \{pk([sK_A])\};$ $\rho_7([pK_B] = \{pk([sK_B])\};$
$\rho_7([z_a] = pencrypt(([k], [K_{AB}]), pk([sK_A]));$
$\boldsymbol{\rho_7([\mathbf{y_a}] = \{[\mathbf{K_{AB}}]\};}$ $\rho_7([z] = \emptyset;$
$\rho_7([z_b] = \{pencrypt(([k], pk([sK_A])), pk([sK_B]))\};$
$\rho_7([z_{b'}] = \{pencrypt(([k], pk([sK_A])), pk([sK_B]))\};$
$\rho_7([x_b] = \{[k]\};$
$\rho_7([y_b] = \{pk([sK_A])\};$ $\boldsymbol{\rho_7([\mathbf{x_a}] = \{[\mathbf{k}]\};}$
$\rho_7([pk(sK_A)]) = \{pk([sK_A])\};$ $\rho_7([pk(sK_B)]) = \{pk([sK_B])\};$
$\rho_7([pencrypt((k, pK_A), pK_B)]) = \{pencrypt(([k], pk([sK_A])), pk([sK_B]))\};$
$\rho_7([pencrypt((x_b, K_{AB}), y_b)]) = \{pencrypt(([k], [K_{AB}]), pk([sK_A]))\};$
$\rho_7([sencrypt(s, y_a)]) = \emptyset;$

- *Calcoliamo ρ_8:*

$\rho_8([c] = \{[e]\};$ $\rho_8([sK_A] = \{[sK_A]\};$
$\rho_8([k] = \{[k]\};$ $\rho_8([sK_B] = \{[sK_B]\};$
$\rho_8([s] = \{[s]\};$ $\rho_8([K_{AB}] = \{[K_{AB}]\};$
$\rho_8([pK_A]) = \{pk([sK_A])\};$ $\rho_8([pK_B] = \{pk([sK_B])\};$
$\rho_8([z_a] = pencrypt(([k], [K_{AB}]), pk([sK_A]));$
$\rho_8([y_a] = \{[K_{AB}]\};$ $\rho_{\mathbf{8}}([\mathbf{z}] = \{[\mathbf{k}]\};$
$\rho_8([z_b] = \{pencrypt(([k], pk([sK_A])), pk([sK_B]))\};$
$\rho_8([z_{b'}] = \{pencrypt(([k], pk([sK_A])), pk([sK_B]))\};$
$\rho_8([x_b] = \{[k]\};$
$\rho_8([y_b] = \{pk([sK_A])\};$ $\rho_8([x_a] = \{[k]\};$
$\rho_8([pk(sK_A)]) = \{pk([sK_A])\};$ $\rho_8([pk(sK_B)]) = \{pk([sK_B])\};$
$\rho_8([pencrypt((k, pK_A), pK_B)]) = \{pencrypt(([k], pk([sK_A])), pk([sK_B]))\};$
$\rho_8([pencrypt((x_b, K_{AB}), y_b)]) = \{pencrypt(([k], [K_{AB}]), pk([sK_A]))\};$
$\rho_{\mathbf{8}}([\mathbf{sencrypt}(\mathbf{s}, \mathbf{y_a})]) = \{\mathbf{sencrypt}([\mathbf{s}], [\mathbf{K_{AB}}])\};$

- *Calcoliamo ρ_9:*

$\rho_9([c] = \{[e]\};$ $\rho_9([sK_A] = \{[sK_A]\};$
$\rho_9([k] = \{[k]\};$ $\rho_9([sK_B] = \{[sK_B]\};$
$\rho_9([s] = \{[s]\};$ $\rho_9([K_{AB}] = \{[K_{AB}]\};$
$\rho_9([pK_A]) = \{pk([sK_A])\};$ $\rho_9([pK_B] = \{pk([sK_B])\};$
$\rho_9([z_a] = pencrypt(([k], [K_{AB}]), pk([sK_A]));$
$\rho_9([y_a] = \{[K_{AB}]\};$ $\rho_9([z] = \{[k]\};$
$\rho_{\mathbf{9}}([\mathbf{z_b}] = \{\mathbf{pencrypt}(([\mathbf{k}], \mathbf{pk}([\mathbf{sK_A}])), \mathbf{pk}([\mathbf{sK_B}])), \mathbf{sencrypt}([\mathbf{s}], [\mathbf{K_{AB}}])\};$
$\rho_{\mathbf{9}}([\mathbf{z_{b'}}] = \{\mathbf{pencrypt}(([\mathbf{k}], \mathbf{pk}([\mathbf{sK_A}])), \mathbf{pk}([\mathbf{sK_B}])), \mathbf{sencrypt}([\mathbf{s}], [\mathbf{K_{AB}}])\};$
$\rho_9([x_b] = \{[k]\};$
$\rho_9([y_b] = \{pk([sK_A])\};$ $\rho_9([x_a] = \{[k]\};$
$\rho_9([pk(sK_A)]) = \{pk([sK_A])\};$ $\rho_9([pk(sK_B)]) = \{pk([sK_B])\};$
$\rho_9([pencrypt((k, pK_A), pK_B)]) = \{pencrypt(([k], pk([sK_A])), pk([sK_B]))\};$
$\rho_9([pencrypt((x_b, K_{AB}), y_b)]) = \{pencrypt(([k], [K_{AB}]), pk([sK_A]))\};$
$\rho_9([sencrypt(s, y_a)]) = \{sencrypt([s], [K_{AB}])\};$

- *Calcoliamo* $\rho_{10} = \rho$*:*

$\rho_{10}([c] = \{[e]\};$ $\rho_{10}([sK_A] = \{[sK_A]\};$
$\rho_{10}([k] = \{[k]\};$ $\rho_{10}([sK_B] = \{[sK_B]\};$
$\rho_{10}([s] = \{[s]\};$ $\rho_{10}([K_{AB}] = \{[K_{AB}]\};$
$\rho_{10}([pK_A]) = \{pk([sK_A])\};$ $\rho_{10}([pK_B] = \{pk([sK_B])\};$
$\rho_{10}([z_a] = pencrypt(([k], [K_{AB}]), pk([sK_A]));$
$\rho_{10}([y_a] = \{[K_{AB}]\};$ $\rho_{10}([z] = \{[k]\};$
$\rho_{10}([z_b] = \{pencrypt(([k], pk([sK_A])), pk([sK_B])), sencrypt([s], [K_{AB}])\};$
$\rho_{10}([z_{b'}] = \{pencrypt(([k], pk([sK_A])), pk([sK_B])), sencrypt([s], [K_{AB}])\};$
$\rho_{10}([x_b] = \{[k]\};$
$\rho_{10}([y_b] = \{pk([sK_A])\};$ $\rho_{10}([x_a] = \{[k]\};$
$\rho_{10}([pk(sK_A)]) = \{pk([sK_A])\};$ $\rho_{10}([pk(sK_B)]) = \{pk([sK_B])\};$
$\rho_{10}([pencrypt((k, pK_A), pK_B)]) = \{pencrypt(([k], pk([sK_A])), pk([sK_B]))\};$
$\rho_{10}([pencrypt((x_b, K_{AB}), y_b)]) = \{pencrypt(([k], [K_{AB}]), pk([sK_A]))\};$
$\rho_{10}([sencrypt(s, y_a)]) = \{sencrypt([s], [K_{AB}])\};$

L'analisi mostra le varie comunicazioni che istanziano le varibili. Chiaramente l'analisi non garantisce che tali comunicazioni possono sempre avvenire, ma ci garantisce che le comunicazioni che in realtà avverranno sono comprese in quelle computate. Perciò tale analisi risulta utile per trovare eventuali errori nel protocollo, ad esempio può far rilevare che alcune comunicazioni all'interno del protocollo non possono mai avvenire; inoltre può rilevare anche la situazione opposta in cui un protocollo permette comunicazioni non previste.

6.3 Proprietà di sicurezza

Utilizziamo ora l'analisi statica per determinare proprietà di sicurezza per un processo P.

Sia S un insieme. Per vedere se il processo P preserva la sicurezza dei dati segreti da S, dovremmo simulare tutti i possibili S-avversari con i quali tale processo può interagire [17].

Sia Spia(P) un processo che soddisfi le seguenti proprietà:

1. Spia(P) conosce tutti gli $a \in S$;

2. Se Spia(P) conosce M e conosce N allora può mandare N lungo M;

3. Se Spia(P) conosce M e lungo M viene spedito il messaggio N, allora Spia(P) conosce N;

4. Per ogni costruttore di arità n, se Spia(P) conosce $M_1, \ldots, M_n$ allora conosce anche $f(M_1, \ldots, M_n)$;

5. Per ogni distruttore g e per ogni equazione $g(M_1, \ldots, M_n) = M$ in def(g), se Spia(P) conosce $M_1, \ldots, M_n$ allora conosce anche M.

Supponiamo ora che l'analisi statica applicata al processo (P| Spia(P)) termini e che ρ sia l'associazione sui termini trovata. Per il Teorema 6 abbiamo che P preserva la segretezza di s da S per l'S-avversario Spia(P) se per ogni $[M] \in \mathcal{T}(Spia(P))$, s non è in $\rho([M])$.

In tale caso è possibile effettuare una assegnazione dei tipi che soddisfa le ipotesi del Teorema 1.

6.3.1 Assegnazione dei tipi

Per far ciò abbiamo bisogno di una ulteriore funzione: $conv(T)$, che calcoliamo in questo modo:

- Per ogni azione di output $\overline{M} < N >$ e per ogni $[M'] \in \rho([M])$ poni $conv([M']) = conv([M']) \cup \rho([N])$;
- Per ogni azione di input $M(x)$ e per ogni $[M'] \in \rho([M])$ poni $conv([M']) = conv([M']) \cup \rho([x])$.

Definiamo la grammatica dei tipi in questo modo:

- T: tipi
  - $[a]$: nome.
  - Per ogni $[f(M_1, \ldots, M_n)] \in \mathcal{F}(P \mid Spia(P))$, poni per ogni $f(T_1, \ldots, T_n) \in \rho([f(M_1, \ldots, M_n)])$,
    $f(T_1, \ldots, T_n)$: applicazione di costruttore.
- $T_{Public} = \{T \mid esiste\, [M] \in \mathcal{T}(Spia(P))\, tale\, che\, T \in \rho([M])\}$.
- $conveys(T) = \{T' \mid T' \in conv(T)\}$.
- $O_f(T_1, \ldots . T_n) = f(T_1, \ldots . T_n)$.

- $O_g(T_1, \ldots, T_n) = \{[M] \mid$ *esiste una equazione* $g(M_1', \ldots, M_n') = M'$ *in* $def(g)$ *tale che* $\sigma M_i' = [M_i] \in \rho([N_i])$ *per qualche* $[N_i] \in \mathcal{T}(P \mid Spia(P))$, *con* $[M_i] : T_i$ *e* $[M] = \sigma M'\}$.

Verifichiamo ora che tale assegnazione soddisfa le ipotesi del Teorema 1. Sia $T \in T_{Pubblic}$ quindi esiste $[M] \in \mathcal{T}(Spia(P))$ tale che $T \in \rho([M])$. Ora sappiamo che se la spia conosce [M] di tipo T allora conosce anche tutto quello che può essere inviato o ricevuto tramite [M] e quindi abbiamo che, per come vien definita la conv([M]), essendo [M] un nome o un costruttore conv([M]) risulta essere un insieme di nomi e costruttori ed essendo $[M] \in T_{Pubblic}$, $conveys(T) = T_{Pubblic}$. Quindi la proprietà P0 viene soddisfatta.

Sia ora per ogni $i \in [1, n]$ $T_i \in T_{Pubblic}$. Ora poiché $T_i \in T_{Pubblic}$, Spia(P) conosce anche $[f(T_1, \ldots, T_n)]$ e quindi $O_f(T_1, \ldots, T_n)$ è definito e $O_f(T_1, \ldots, T_n) \in T_{Pubblic}$. Quindi la proprietà P1 viene soddisfatta.

Sia ora per ogni $i \in [1, n]$ $T_i \in T_{Pubblic}$. Ora poiché $T_i \in T_{Pubblic}$, Spia(P) conosce anche $g(T_1, \ldots, T_n)$ nei casi in cui è definito e quindi sia $T \in O_g(T_1, \ldots, T_n)$ allora $T \in T_{Pubblic}$. Quindi la proprietà P2 viene soddisfatta.

La proprietà P3 viene banalmente soddisfatta per come viene definito $O_g(T_1, \ldots, T_n)$.

Quindi per poter applicare il teorema di segretezza per il sistema a tipi rimane da verificare che $E \vdash P$ con $E = \{a : a \mid a \in fn(P)\}$.

Lemma 6 $E \vdash P$ *con* $E = \{a : a \mid a \in fn(P)\}$.

Dimostrazione. (Utilizziamo le regole della Tabella 4.1) Verifichiamo per prima cosa che $E = \{a : a \mid a \in fn(P)\} \vdash \diamond$. Ora per la regola 1 abbiamo che $0 \vdash \diamond$. Per la regola 2 abbiamo che $0, a : a \vdash \diamond$ (con a $\in fn(P)$) in quanto abbiamo $0 \vdash \diamond$ e $a \notin dom(0)$. Quindi l'ambiente $E' = \{a : a\} \vdash \diamond$ (con a $\in fn(P)$).

Quindi proseguendo in questo modo, aggiungendo cioè un elemento alla volta nell'ambiente ben formato appena trovato, arriviamo a costruire $E = \{a : a \mid a \in fn(P)\}$ con $E \vdash \diamond$.

Verifichiamo per induzione che $E \vdash P$, con queste ipotesi:

- Il processo che consideriamo non contiene l'operatore di replica, in quanto dimostrare che $E \vdash !P$, equivale a dimostrare che $E \vdash P$.

- Il processo che consideriamo non contiene l'operatore di composizione parallela in quanto se $P = A.B.C.D.(G \mid H)$ e noi vogliamo verificare che $E \vdash P$, allora questo equivale a verificare che $E \vdash P'$, e che $E \vdash P''$ con $P' = A.B.C.D.G$ e $P'' = A.B.C.D.H$.

- Il processo che consideriamo non contiene applicazioni di distruttore nella forma $let\, x = g(M_1, \ldots, M_n)\, in\, A\, else\, B$, ma solo nella forma $let\, x = g(M_1, \ldots, M_n)\, in\, A$ in quanto se $P = A.B.let\, x = g(M_1, \ldots, M_n) in\, C\, else\, D$ e vogliamo verificare che $E \vdash P$, allora questo equivale a verificare che $E \vdash A.B.let\, x = g(M_1, \ldots, M_n)\, in\, C$ e che e che $E \vdash A.B.D$.

Passo base:

Sia Q un processo che soddisfi le nostre ipotesi. Supponiamo che Q sia formato da un'unica azione.

- Sia $Q = (\nu a)$. Ora $a : a$ e quindi $E \vdash Q$ se $E, a : a \vdash 0$. Per la regola 7, $E' \vdash 0$ se $E' \vdash \diamond$. Ora $E' = E, a : a$ e risulta $E' \vdash \diamond$ per la regola 2 in quanto abbiamo visto che $E \vdash \diamond$ ed essendo a un nome bound $a \notin dom(E)$.

- Sia $Q = \overline{M} < N >$. Ora essendo la prima istruzione, abbiamo solo due casi: o sia M che N sono nomi liberi, oppure M è nome libero ed N è un costruttore. Vediamo il primo caso: essendo entrambi nomi liberi abbiamo che $E \vdash M : M$ e $E \vdash N : N$, inoltre per come abbiamo definito la conveys, $N \in conveys(M)$ e $E \vdash 0$ per la regola 7. Vediamo il secondo caso: come prima $E \vdash M : M$. Sia $N = f(N_1, \ldots, N_n)$ ove gli N_i possono essere o nomi liberi oppure applicazione di costruttori. Affinché $E \vdash f(N_1, \ldots, N_n) : O_f(N_1, \ldots, N_n)$ dobbiamo avere $E \vdash \diamond$, e questo l'abbiamo già dimostrato, inoltre per ogni $i \in [1, n]\, E \vdash N_i : T_i$. Ora se N_i è un nome $T_i = N_i$, invece se N_i è un costruttore si applica tale procedimento ricorsivamente sui suoi termini. Essendo N un costruttore privo di variabili, $f(N_1, \ldots, N_n) : f(N_1, \ldots, N_n)$, ed inoltre per come abbiamo definito la conveys, $N \in conveys(M)$. Quindi anche la regola 5 viene soddisfatta.

- Sia $Q = M(x)$. Come al solito in questa prima azione M può essere solo un nome libero e quindi $E \vdash M : M$. In questo caso, $conveys(M) = \emptyset$, perciò per la regola 7 abbiamo che $E \vdash 0$, quindi anche la regola 6 viene soddisfatta.

- Sia $Q = let\, x = g(M_1, \ldots, M_n)$. Come prima ciascuno degli M_i può essere o un nome o un costruttore (senza variabili), quindi per ogni M_i, $E \vdash M_i : M_i$. Per la regola 7 abbiamo che $E \vdash 0$ ed inoltre, per la regola 2, $E, x : T \vdash 0$ in quanto per ogni $T \in O_g(M_1, \ldots, M_n)$, $x \notin dom(E)$. Quindi anche la regola 11 viene soddisfatta.

Passo induttivo.

Sia quindi $Q = Q_1. \ldots .Q_n.Q_{n+1}. \ldots .Q_m$ il processo in analisi (che soddisfa le nostre ipotesi). Supponiamo quindi che $E \vdash Q_1. \ldots .Q_n$ e dimostriamo che $E \vdash Q_1. \ldots .Q_n.Q_{n+1}$ con i Q_i processi composti da singole azioni. Sia inoltre E' l'ambiente ben formato ottenuto partendo da E analizzando le prime n azioni.

1. Sia $Q_{n+1} = (\nu a)$. Questa azione non crea nessun problema in quanto essendo $E' \vdash \diamond$, ed essendo $a \notin dom(E')$, $E', a : a \vdash \diamond$. Quindi $E' \vdash Q_{n+1}$.

2. Sia $Q_{n+1} = \overline{M} < N >$. Per M abbiamo questi tre casi: M nome libero, M nome bound o M varibile. Se M è un nome libero abbiamo che $E' \vdash M : M$ in quanto $E \subseteq E'$. Se M è un nome bound, allora certamente una azione precedente sarà stata $Q_i = (\nu M)$ per qualche $i \leq n$, quindi $M \in E'$ e perciò abbiamo che $E' \vdash M : M$. Se M è una variabile, allora certamente una azione precedente sarà stata $Q_i = M'(M)$ oppure $Q_i = let\, M = g(M_1, \ldots, M_n)$ con $i \leq n$. In entrambi i casi si è assegnato un tipo a M, e la si è inclusa nell'ambiente. Quindi $M : T$ per qualche T ed inoltre $M \in E'$. Per N abbiamo questi quattro casi: N nome libero, N nome bound, N variabile o N costruttore. Se N è un nome libero abbiamo che $E' \vdash N : N$ in quanto $E \subseteq E'$. Se N è un nome bound, allora certamente una azione precedente sarà stata $Q_i = (\nu N)$ per qualche $i \leq n$, quindi $N \in E'$ e perciò abbiamo che $E' \vdash N : N$. Se N è una variabile, allora certamente una azione precedente sarà stata $Q_i = N'(N)$ oppure $Q_i = let\, N = g(N_1, \ldots, N_n)$ con $i \leq n$. In entrambi i casi si è assegnato un tipo a N, e la si è inclusa nell'ambiente. Quindi $N : T$ per qualche T ed inoltre $N \in E'$. Se N è un costruttore del tipo $f(N_1, \ldots, N_n)$, dobbiamo verificare se $E' \vdash N_i : T_i$ per qualche T_i ed inoltre che $O_f(T_1, \ldots, T_n)$ risulti definito. Come abbiamo appena visto se N_i è un nome libero o bound, o una variabile non c'è nessun problema in quanto tale T_i esiste e $E' \vdash N_i : T_i$, se invece N_i è a sua volta un costruttore applichiamo tale procedimento ricorsivamente. Abbiamo perciò che $E' \vdash N_i : T_i$ per qualche T_i ($i \in [1, n]$) ed inoltre che $O_f(T_1, \ldots, T_n)$ è definito, quindi anche in questo caso tale azione non crea problemi. Quindi, per come abbiamo definito la conveys, il tipo di N appartiene alla conveys calcolata sul tipo di M. Perciò nemmeno questa azione crea problemi: $E' \vdash Q_{n+1}$.

3. Sia $Q_{n+1} = M(x)$. Facciamo lo stesso ragionamento del punto precedente su M e otteniamo che $E' \vdash M : T$ per qualche T; inoltre aggiungiamo a E', x di tipo T' per ogni $T' \in conveys(T)$; per la regola 2 abbiamo che $E'x : T' \vdash \diamond$. Anche in questo caso quindi $E' \vdash Q_{n+1}$.

4. Sia $Q_{n+1} = let\, x = g(M_1, \ldots, M_n)$. Facciamo lo stesso ragionamento fatto per

N nel punto 2, quindi analizziamo in cui M_i sia un nome (libero o bound), una variabile o un costruttore. Abbiamo visto che per tutti e quattro questi casi esiste T_i tale che $E' \vdash M_i : T_i$ con $i \in [1, n]$. Aggiungiamo poi a E', x di tipo T per ogni $T \in O_g(T_1, \ldots, T_n)$; per la regola 2 abbiamo che $E'x : T \vdash \diamond$. Anche in questo caso quindi $E' \vdash Q_{n+1}$.

5. Sia $Q_{n+1} = 0$. Abbiamo che $E' \vdash Q_{n+1}$ in quanto $E' \vdash \diamond$.

Quindi per induzione $E \vdash P$ per ogni processo P, analizzato e tipato secondo la nostra analisi statica. Perciò possiamo concludere che $E = \{a : a \mid a \in fn(P)\}$ è un ambiente ben formato per il processo P. C.v.d.

Abbiamo dunque il seguente teorema:

Teorema 7 *Se P preserva la segretezza di s da S per l'S-avversario Spia(P), allora P preserva la segretezza di s da S per ogni S-avversario.*

Quindi un processo che abbia le stesse proprietà di Spia(P) simula il peggior contesto possibile per il processo P. Purtroppo però la proprietà 4 fa in modo che Spia(P) continui a richiamare costruttori su costruttori, creando così infiniti nuovi termini, ed in questo modo l'analisi di $(P \mid Spia(P))$ non termina mai. Perciò affinché l'analisi termini dobbiamo richiedere una proprietà diversa dalla 4 che impedisca questo ciclo infinito. Realizziamo tale nuova proprietà stabilendo delle regole per i costruttori:

- Per ogni costruttore $f(M_1, \ldots, M_n)$ decidiamo a priori quali sono le n-uple "buone" in questo senso: $(M_1, \ldots, M_n)$ è una n-upla buona se soddisfa certe condizioni logiche; ad esempio (M) è buona per pk se M è una sk; (M_1, M_2) è buona per *pencrypt* se M_2 è una pk e M_1 è un messaggio qualsiasi;

- Stabiliamo le n-uple buone in modo che non si creino cicli infiniti applicando i vari costruttori a tali n-uple. Ad esempio se diciamo che (M_1, M_2) è buona per *pencrypt* se M_2 è una pk e M_1 è un messaggio qualsiasi allora si può formare il ciclo infinito $pencrypt(pencrypt(\ldots(pencrypt(M, pk))\ldots, pk), pk)$. Perciò per evitare un tale ciclo escludiamo dalle coppie buone per *pencrypt* quelle dove il primo elemento della coppia è *pencrypt*;

- Aggiungiamo ad S un nuovo nome m tale che $m \notin \mathcal{T}(P \mid Spia(P))$;

- Per ogni costruttore $f(M_1, \ldots, M_n)$ stabiliamo che: $f(M_1, \ldots, M_n)$=$f(M_1, \ldots, M_n)$ se $(M_1, \ldots, M_n)$ è una n-upla buona, e che $f(M_1, \ldots, M_n)$=m altrimenti.

In questo modo eliminiamo il problema della creazione di infiniti nuovi termini.

Esempio 7 *Se supponiamo che l'insieme dei costruttori sia formato dai seguenti elementi: $pk(M)$, $pencrypt(M, pk(N))$ e $sencrypt(M, N)$, allora possiamo stabilire che (M) è buona per $pk(M)$ se M è una chiave segreta (sk) (come in [7] p. 9), la coppia (M,N) è buona per pencrypt se N è una chiave pubblica (pk) e M è diverso da pencrypt e da sencrypt, ed infine che (M,N) è una coppia buona per sencrypt se N è una chiave (k) e M è diverso sia da pencrypt che da sencrypt.*

Ovviamente con tale ipotesi la nostra Spia(P) non simulerà più tutti i possibili S-avversari, ma cercherà di simulare un processo abbastanza cattivo dal punto di vista logico.

Perciò sia $\mathcal{F}$ l'insieme dei costruttori che consideriamo, e supponiamo che tali costruttori soddisfino le regole viste. Costruiamo quindi il processo Spia(P) come segue:

- Per ogni $a \in S$ sia Q_a il seguente processo:

  $Q_a = !\overline{c} < a > | !c(z_1).c(z_2).\overline{z_1} < z_2 >$;

- Sia $R_1 = !c(z_3).z_3(z_4).!\overline{c} < z_4 >$;

- Per ogni costruttore $f \in \mathcal{F}$ di arità n sia

  $F_f = !(c(x_1). \ldots .c(x_n).!\overline{c} < f(x_1, \ldots, x_n) >$;

- Per ogni distruttore g di arità n sia

  $G_g = !(c(x_1). \ldots .c(x_n).let\, y = (g(x_1, \ldots, x_n).!\overline{c} < y >))$;

- Infine supponiamo siano r gli elementi di S, t tutti i costruttori ed u tutti i distruttori;

- Definiamo perciò $Spia(P) = (\nu c)(Q_{a_1} \mid \ldots \mid Q_{a_r} \mid R_1 \mid F_1 \mid \ldots \mid F_t \mid G_1 \mid \ldots \mid G_u)$.

Applichiamo l'analisi al processo $(P \mid Spia(P))$. Se l'algoritmo termina allora tale risultato segue direttamente dal Teorema 7:

Teorema 8 *Il protocollo P preserva la segretezza dei suoi dati segreti da S, per ogni S-avversario i cui costruttori soddisfano le ipotesi viste, se per ogni $[M] \in \mathcal{T}(Spia(P))$, nessuno di tali dati segreti si trova in $\rho([M])$.*

Capitolo 7

Conclusioni

L'algoritmo di analisi statica proposto risulta quindi uno strumento utile per analizzare protocolli di comunicazione. L'analisi calcola staticamente tutte le possibili comunicazioni che istanziano le variabili all'interno del protocollo, e ci garantisce che le comunicazioni che in realtà avvengono sono comprese in quelle computate. Questo risulta utile, ad esempio, per rilevare che alcune comunicazioni, che si auspica avvengano, in realtà sono impossibili; oppure può far notare la situazione opposta in cui il protocollo permette comunicazioni non gradite. Inoltre tale analisi è molto utile soprattutto quando si analizza il protocollo tenendo conto del contesto nel quale viene eseguito: vedendo tale contesto come processo e componendolo in parallelo con il protocollo in questione, l'analisi statica permette di ricavare importanti informazioni sulla dinamica delle comunicazioni senza, peraltro, eseguire il sistema.

Ringraziamenti

Vorrei ringraziare innanzittutto il Prof. Gilberto Filé per avermi generosamente dedicato il suo tempo per spiegazioni, commenti e delucidazioni, il Prof. Livio Colussi per avere gentilmente letto il mio lavoro, e tutti gli altri insegnanti che in questi quattro anni mi hanno condotto lungo la via dell'apprendimento. Vorrei ringraziare mio marito che mi ha ascoltata e sostenuta per tutto questo tempo, i miei genitori, mia sorella e mio fratello per aver sempre creduto in me e per avermi sempre aiutata. Inoltre un grazie speciale va a Maria Silvia Pini che si è resa sempre disponibile a chiarire i miei dubbi, ad Angela Grassi e a Sara De Marchi che con la loro simpatia e generosità hanno reso piacevoli le ore di studio al Paolotti.

Bibliografia

[1] Martín Abadi.

Secrecy by typing in security protocols. In *Proceedings of Theoretical Aspects of Computer Software, Third International Symposium LNCS 1281*, 1997.

[2] Martín Abadi.

Security protocols and their properties. In F. Bauer e R. Steinbrueggen, editor, *Foundations of Secure Computation*, NATO Science Series, pagine 36-60, IOS Press, 2000.

[3] Martín Abadi e Bruno Blanchet.

Analyzing Security Protocols with Secrecy Types and Logic Programs. In *29th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2002)*, pagine 33-34, ACM Press, 2002.

[4] Martín Abadi e Bruno Blanchet.

Secrecy types for asymmetric communication. In F. Honsell e M. Miculan, editor, *Foundations of Software Science and Computation Structures (FoSSaCS 2001)*, volume 2030 of *Lecture Notes in Computer Science*, pagine 25-41, Springer-Verlag, 2001.

[5] Martín Abadi e C. Fournet.

Mobile values, new names, and secure communication. In *Proceedings of the 28th Annual ACM Symposium on Principles of Programming Languages (POPL'01)*, pagine 104-115, 2001.

[6] Martín Abadi e Andrew Gordon.

A calculus for cryptographic protocols: the spi-calculus. *Information and Computation*, 148(4):1-70, 1999.

[7] Bruno Blanchet.

An efficient cryptographic protocol verifier based on Prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pagine 82-96, 2001.

[8] Chiara Bodei, Pierpaolo Degano, Flemming Nielson e Hanne Riis Nielson.

Static analysis for the π-caclulus with application to security. *Information and Computation*,165:68-92, 2001.

[9] Gerard Boudol e Ilaria Castellani.

Noninterference for concurrent programs and thread systems. *Theoretical Computer Science*, 281(1):109-130, 2002.

[10] L. Cardelli, G. Ghelli e A. Gordon.

Secrecy and group creation. In C. Palamidessi, editor, *CONCUR 2000: Concurrency Theory*, volume 1877 di *Lecture Notes in Computer Science*, pagine 365-379, Springer-Verlag, 2000 .

[11] L. Cardelli e A. Gordon.

Mobile ambients: foundations of system specification and computation structures. *Lecture Notes in Computer Science Vol 1378*, pagine 140-155, Springer-Verlag, 1998.

[12] I. Cervesato, N. A. Durgin, P. D. Lincoln, J. C. Mitchell e A. Scedrov.

A meta-notation for protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW'99)*, pagine 55-69, 1999.

[13] Patrick Cousot and Radhia Cousot.

Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of 4th ACM POPL*, pagine 238-252, 1977.

[14] M. Debabbi, M. Mejri, N. Tawbi e I. Yahmadi.

A new algorithm for the automatic verification of authentication protocols: From specifications to flaws and attack scenarios. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security protocols*, Rutgers University, New Jersey, 1997.

[15] G. Denker, J. Meseguer e C. Talcott.

Protocol specification and anlysis in Maude. In N. Haintze e J. Wing, editor, *Proceedings of Workshop on Formal Methods and Security Protocols*, 1998.

[16] Gilberto Filé e Alberto Griggio.

A simple control flow anlysis of the π-calculus. *Manoscritto*, 2003.

[17] Gilberto Filé e Alberto Griggio.

How to account for the context while checking secrecy of π-calculus processes. *Manoscritto*, 2003.

[18] Riccardo Focardi, Roberto Gorrieri e Fabio Martinelli.

Noninterference for the analysis of criptographic protocols. In Ugo Montanari, editor, *Proceedings of the 27th ICALP*,numero 1853 in LNCS, 2000.

[19] A. Gordon e A. Jeffrey.

Authenticity by typing for security protocols. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pagine 145-159, 2001.

[20] M. Hennesy e J. Riely.

Information flow vs. resource access in the asyncronus π-calculus. In *Proceedings of the 27th International Colloquium on Automata, Language e Programming*, Lecture Notes in Computer Science, pagine 415-427, Springer-Verlag, 2000.

[21] Kohei Honda, Vasco Vasconcelos e Nobuko Yoshida.

Secure information flow as typed program behaviour. In Smolka E., editor, *Proceedings of the 9th ESOP*,numero 1782 in LNCS, Springer-Verlag, 2000.

[22] Robin Milner.

A calculus of Communicating Systems, volume 92 di *Lecture Notes in Computer Science*, Springer-Verlag, 1980.

[23] Robin Milner.

Communication and Concurrency, Prentice Hall, 1989.

[24] Robin Milner.

Communicating and mobile systems: the π-calculus, Cambridge University Press, 1999.

[25] Robin Milner, Joachim Parrow e David Walker.

A calculus of mobile processes (i e ii). *Information and Computation*, 100(1):1-77, 1992.

[26] Robin Milner, Joachim Parrow e David Walker.

Modal Logics for mobile processes. *Theoretical Computer Science*, 114(1):149-171, 1993.

[27] Flemming Nielson, Hanne Riis Nielson e Chris Hankin.

Principles of Program Analysis, Springer-Verlag, 1999.

[28] Davide Sangiorgi e David Walker.

The π-calculus: a Theory of Mobile Processes, Cambridge University Press, 2001.

[29] C. Weidenbach.

Towards an automatic analysis of security protocols in first-order logic. In H. Ganzinger, editor, *16th International Conference on Automated Deduction (CADE-16)*, volume 1632 di *Lecture Notes in Artificial Intelligence*, pagine 314-328, Springer-Verlag, 1999.