Sviluppo di un Framework per Symbolic Partial Model Checking

Marco Malvasio

Tesi presentata per il conseguimento del titolo di

DOTTORE MAGISTRALE
IN
INGEGNERIA INFORMATICA

Relatore	Chiar.^{mo} Prof. Alessandro Armando
Correlatori:	Dott. Gabriele Costa
	Ing. Alessio Merlo

Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi

Università degli Studi di Genova

22 dicembre 2014

Bite it like a bulldog
Bite it to the bone
Never let it go

Indice

1 Introduzione **6**

2 Preliminari tecnici **9**
2.1 Modelli 9
2.1.1 Transition System 9
2.1.2 Labelled Transition System 11
2.1.3 Symbolic Transition System 12
2.1.4 Algebre di Processo 14
2.2 Specifiche di sicurezza 16
2.2.1 Security Automata 16
2.2.2 Linear Temporal Logic 17
2.2.3 μ-calcolo 19
2.3 Model Checking 22
2.3.1 Costo computazionale 24

3 Filosofi al Buffet: un caso di studio **25**

4 Symbolic Partial Model Checking **28**
4.1 Partial Model Checking 28
4.1.1 μ-calcolo equazionale 28
4.1.2 Quotienting 30
4.1.3 Semplificazioni 33
4.2 Symbolic Partial Model Checking 34
4.2.1 Symbolic Quotienting 34

5 Prototipo **38**
5.1 Architettura 38
5.2 Implementazione 38
5.2.1 Symbolic Partial Model Checker 40
5.2.2 Semplificazioni 41
5.2.3 Model Checking 41
5.2.4 Input ed Output 41
5.3 Le tecnologie usate 42
5.3.1 Java Compiler Compiler 42

5.3.2 Simple Promela Interpreter 42
5.3.3 LTL2BA 43

6 Valutazione sperimentale **44**
6.1 Struttura degli esperimenti 44
6.2 Esperimenti e tempi 44

7 Conclusioni **46**
7.1 Sviluppi futuri 46

Appendice A Codice **50**
A.1 Grammatica del π-calcolo 50
A.2 Grammatica del μ-calcolo 50

Elenco delle figure

2.1 Problema dei Filosofi a cena 10
2.2 TS per un possibile Filosofo 11
2.3 LTS per il filosofo 12
2.4 STS per un Filosofo 13
2.5 Security Automata per la proprietà di Esempio 5 17

3.1 STS per il filosofo ingordo 26
3.2 STS per il filosofo testardo 26
3.3 STS per il tavolo 27

5.1 Schema generale dell'architettura del Framework 39

Elenco delle tabelle

4.1 Regole di quotienting 31
4.2 Trasformazioni valide sulle asserzioni 36
4.3 Regole di quotienting simbolico 37

6.1 Caso di studio con 8 agenti di tipo Filosofo testardo 45
6.2 Caso di studio con agenti di tipi alternati 45

Capitolo 1

Introduzione

La verifica automatica dei sistemi (hardware e software) e delle loro proprietà ha ricevuto negli ultimi anni un'attenzione sempre crescente. La principale motivazione per questo interesse è il rapido aumento degli agenti computazionali e delle responsabilità ad essi delegate.

Esistono diverse tecniche, che hanno lo scopo di verificare la correttezza dei sistemi. Per esempio, il *testing* consiste nell'osservare un sistema e le sue reazioni a determinati stimoli. In questo ambito, testare un software significa eseguirlo, fornirgli parametri e valori di input ed osservarne i cambiamenti di stato ed output. Se il software presenta difetti, i test possono effettivamente individuare alcuni di essi. Tuttavia non è possibile in generale testare *tutte* le possibili esecuzioni di un programma (dato che esse possono essere infinite e infinitamente lunghe).

Negli ultimi anni, le tecniche di analisi statica (cioè quelle che verificano un software senza eseguirlo) si sono affermate come un'importante alternativa o complementazione del testing. Tra queste, il *Model Checking* è probabilmente la tecnica di verifica formale più usata. In breve, il Model Checking consiste nella ricerca, tra gli stati di un sistema, di configurazioni che violano una data specifica. Dato che esso utilizza una rappresentazione astratta degli stati del sistema, è possibile utilizzarlo per verificare la correttezza di "esecuzioni" sia finite che infinite.

Il *Model Checking* non è esente da limitazioni, di cui una delle più importanti è l'*esplosione combinatoria degli stati*. Tale problema si verifica in presenza di sistemi concorrenti in cui più agenti computazionali distinti possono effettuare cambiamenti di stato indipendentemente (asincroni) o dipendentemente (sincroni) dagli altri. Sostanzialmente, al crescere *lineare* del numero di tali agenti, il numero degli stati del sistema da analizzare cresce in maniera *esponenziale*. Dato che la complessità computazionale del Model Checking dipende dal numero degli stati da visitare, la loro esplosione combinatoria riduce significativamente il dominio applicativo della tecnica e rende molti problemi di interesse intrattabili.

Per mitigare questo problema, negli ultimi anni sono stati proposti diversi approcci. Tra questi uno dei più promettenti è il *Partial Model Checking*. In breve esso consiste nel trasformare istanze del problema del Model Checking diminuendo il grado di parallelismo del sistema da verificare. Per ottenere questo risultato si utilizzano tecniche di trasformazione che convertono alcuni degli agenti computazionali in formule logiche. Le formule ottenute in questo modo confluiscono nella specifica da verificare. Infine, procedure di semplificazione vengono utilizzate per ridurre la dimensione della specifica così ottenuta. Anche se questo approccio non evita il problema dell'esplosione combinatoria degli stati, in molti casi pratici e di interesse rende le istanze del problema del Model Checking trattabili.

In questa tesi consideriamo un'estensione del Partial Model Checking detto *Symbolic Partial Model Checking*. L'approccio sfrutta la nozione di *esecuzione simbolica* per rappresentare in maniera compatta stati e transizioni di un sistema. Variabili vengono processate al posto dei valori concreti durante la fase di valutazione parziale del Partial Model Checking. Il risultato è una trattazione astratta dei comportamenti del sistema e delle formule della specifica che permette una rappresentazione sintetica di entrambe.

Lo scopo di questa tesi è stato quello di progettare ed implementare una piattaforma per Model Checking che sfruttasse il Symbolic Partial Model Checking per mitigare il problema dell'esplosione combinatoria degli stati. In particolare, sono stati sviluppati componenti per la rappresentazione di modelli computazionali (basate su algebre di processo) e specifiche (basate su logiche temporali). Inoltre è stato implementato un modulo di Symbolic Partial Model Checking che processa modelli e specifiche per ottenere un'istanza del problema del Model Checking. Infine la piattaforma è stata integrata con strumenti allo stato dell'arte per la trattazione di tali problemi.

Struttura della tesi. Questo documento si compone delle seguenti sezioni.

- **Sezione 2.** In questa Sezione presentiamo modelli, politiche e formalismi necessari per descriverli. Inoltre viene descritto il Model Checking ed il problema dell'esplosione combinatoria degli stati.

- **Sezione 3.** In questa Sezione viene presentato il caso di studio scelto. Questo Working Example consiste in una rappresentazione di diversi agenti che interagiscono tra di loro. Tali iterazioni verranno validate rispetto ad un'apposita politica anch'essa descritta in Sezione.

- **Sezione 4.** In questa Sezione viene introdotto il Partial Model Checking e successivamente viene presentata la sua estensione al Symbolic Partial Model Checking. In particolare vengono trattate tutte le tecniche di semplificazione che consentono di limitare il problema dell'esplosione combinatoria.

- **Sezione 6.** La Sezione è dedicata ai risultati sperimentali ottenuti ed alle performance dell'implementazione, dove i risultati ottenuti vengono paragonati a quelli del Model Checking classico.
- **Sezione 7.** Infine in questa Sezione vengono presentate le conclusioni ed i possibili sviluppi futuri.

Capitolo 2

Preliminari tecnici

In questo capitolo saranno spiegati i modelli ed i formalismi necessari per esprimerli. Saranno introdotti anche i linguaggi necessari per modellare agenti e politiche, per poi passare al Model Checking. Verrà inoltre presentato un esempio che ci permetterà in seguito di spiegare agevolmente il nostro caso di studio.

2.1 Modelli

In generale, un *modello* è una rappresentazione astratta del comportamento di un sistema. Usare un modello ha i suoi vantaggi, infatti si possono caratterizzare solamente gli aspetti del sistema che si vogliono analizzare, trascurando quelli non necessari o poco interessanti. Si può avere un modello accurato anche senza dover specificare il sistema nella sua interezza. Nel seguito della sezione vedremo alcuni formalismi che permettono di descrivere in maniera formale i modelli.

2.1.1 Transition System

Un sistema di transizioni, nel seguito *Transition System (TS)*, è una macchina astratta che consiste in un insieme di stati e transizioni tra stati.

Definizione 1. *Un TS è una tupla* $(S, \rightarrow, i)$ *dove:*

- S *è un insieme di stati*
- $\rightarrow \subseteq S \times S$ *è una relazione di transizione*
- $i \in S$ *è lo stato iniziale*

Per semplicità scriviamo $s \rightarrow s'$ per indicare $(s, s') \in \rightarrow$.

Tramite gli stati si rappresenta la configurazione di un sistema, mentre le transizioni indicano i cambi di stato consentiti.

Inoltre chiamiamo *traccia* del TS la sequenza di stati $s_0 \cdot \ldots \cdot s_n$ se e solo se esiste una sequenza di transizioni $s_0 \rightarrow \ldots \rightarrow s_n$ con $s_0 = i$.

Esempio 1. *Si consideri il classico problema dei* Filosofi a cena*, proposto da Edsger Dijkstra per modellare i problemi di accesso a risorse condivise e descritto in [18]. Alcuni Filosofi sono seduti ad un tavolo rotondo, e davanti a ciascuno di loro si trova un piatto di spaghetti (vedi Figura 2.1).*

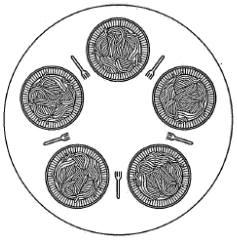


Figura 2.1: Problema dei Filosofi a cena

Dal momento che gli spaghetti sono troppo scivolosi, ogni Filosofo deve avere due forchette per poterli mangiare. Ogni Filosofo inoltre pensa *oppure* mangia*. Se un Filosofo smette di pensare, cerca di prendere dal tavolo la forchetta che sta alla sua sinistra e quella alla sua destra, in ordine arbitrario ed una alla volta. Appena riesce ad impossessarsi di due forchette, il Filosofo mangia per un tempo indefinito. Quando ha finito, le posa entrambe, una alla volta, e torna a pensare.*

Può succedere che ogni filosofo decida di mangiare a tempo con gli altri e prenda una forchetta. A questo punto tutti i filosofi si trovano in una situazione di stallo: vorrebbero raccogliere la seconda forchetta dal tavolo, ma non ci sono più forchette disponibili, per cui ogni Filosofo si trova a pensare per sempre, aspettando invano una posata che non arriverà, senza riuscire a mangiare. Tale situazione è detta deadlock *e si può presentare in un contesto di accesso alle risorse condivise, se non vengono adottate opportune precauzioni.*

Il TS che rappresenta un possibile Filosofo è riprodotto in Figura 2.2. Dal momento che la formulazione del problema lascia spazio ad interpretazione, si potrebbe ad esempio pensare che i Filosofi raccolgano per prima sempre la forchetta alla loro destra. Il deadlock *sarebbe sempre possibile, ma il TS che rappresenta l'agente sarebbe diverso.*

Lo stato a *rappresenta il filosofo* affamato *che non ha ancora una forchetta.* s_1 *ed* s_2 *rappresentano rispettivamente l'acquisizione della forchetta sinistra e della forchetta destra. La transizione successiva, in entrambi gli stati, ci porta all'acquisizione della forchetta mancante e quindi allo stato* m *in cui il filosofo può mangiare. La stessa cosa avviene per la restituzione delle posate* r_1 *e* r_2*: indipendentemente dall'ordine di restituzione, una volta che entrambe le posate sono state restituite, il filosofo si trova nuovamente nello stato* a*, pronto per ricominciare. I loop sugli stati* a *e* m *modellano l'attesa ciclica sullo stesso stato.*

Una possibile traccia di esecuzione è la seguente: $\beta \equiv a \cdot s_1 \cdot m \cdot m \cdot r_1 \cdot a$

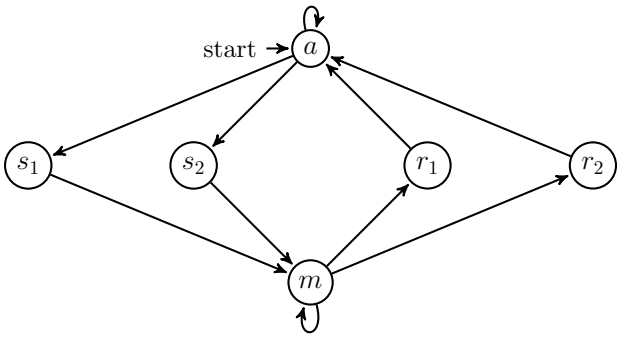


Figura 2.2: TS per un possibile Filosofo

Come si evince dalla Figura 2.2, questo corrisponde al Filosofo che impugna una forchetta, portandosi dallo stato a allo stato s_1, impugna l'altra e arriva in m pronto per mangiare. Rimane nello stato m per due volte e successivamente posa una forchetta alla volta, arrivando prima in r_1 e poi in a (cioè ritorna nello stato iniziale). □

2.1.2 Labelled Transition System

Un formalismo per la modellazione di agenti computazionali sono i *Labelled Transition System (LTS)* [2]. Questa particolare estensione dei TS possiede delle etichette per rappresentare effetti laterali della computazione. Un modo intuitivo per immaginare tali effetti è in termini di eventi visibili da un osservatore esterno.

Definizione 2. *Un LTS è una tupla $(S, Act, \rightarrow, s_0)$ dove*

- *S è un insieme di stati*
- *Act è un insieme di azioni*
- *$\rightarrow \subseteq S \times (Act \cup \overline{Act}) \times S$ è una relazione di transizione*
- *$s_0 \in S$ è lo stato iniziale*

□

Per aumentare la leggibilità, $(s, \alpha, s') \in \rightarrow$ è abbreviato con $s \xrightarrow{\alpha} s'$.

Utilizzando la nozione di azioni osservabili, possiamo descrivere il comportamento di un LTS in termini di *tracce di esecuzione*. Brevemente, diciamo che un LTS $(S, Act, \rightarrow)$ produce (a partire da uno stato iniziale $s_0 \in S$) una traccia $\sigma = a_1 a_2 \dots a_n$ se $\exists s_1, \dots, s_n \in S$ t.c. $s_0 \xrightarrow{a_1} s_1 \dots s_{n-1} \xrightarrow{a_n} s_n$.

Esempio 2. *Si consideri nuovamente l'Esempio 1. Usiamo LTS per modellare il comportamento dell'i-esimo Filosofo. In particolare, definiamo* $t_i = (S, Act, \rightarrow, s_0)$ *dove* $S = \{a, s_1, s_2, r_1, r_2, m\}$, $Act = \{\bar{\mathrm{f}}_i, \mathrm{f}_{i+1}, \tau\}$ *e* $\rightarrow$ *è definita come in Figura 2.3, dove utilizziamo il simbolo* $\bar{\cdot}$ *per l'azione di prendere una forchetta. In questo modo* $\bar{\mathrm{f}}$ *e* f *rappresentano rispettivamente l'acquisizione e la rinuncia ad una forchetta. Le transizioni* τ *rappresentano un'azione interna, che non cambia lo stato in cui si trova il sistema.*

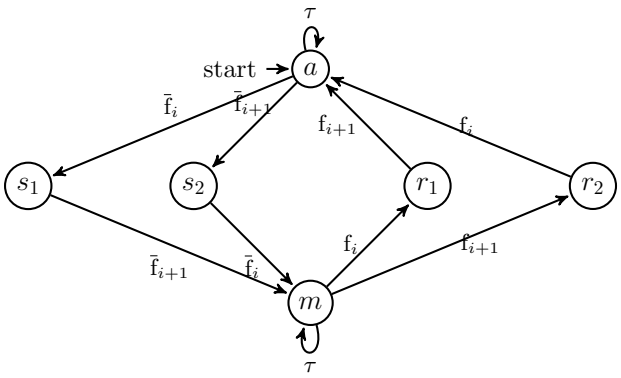


Figura 2.3: LTS per il filosofo

Una possibile traccia per questo agente è la seguente:

$$\beta \equiv \bar{\mathrm{f}}_i \cdot \bar{\mathrm{f}}_{i+1} \cdot \tau \cdot \mathrm{f}_i \cdot \mathrm{f}_{i+1}$$

Infatti esiste una sequenza di stati tali che

$$a \xrightarrow{\bar{\mathrm{f}}_i} s_1 \xrightarrow{\bar{\mathrm{f}}_{i+1}} m \xrightarrow{\tau} m \xrightarrow{\mathrm{f}_i} r_1 \xrightarrow{\mathrm{f}_{i+1}} a$$

□

2.1.3 Symbolic Transition System

Gli STS (*Symbolic Transition System*), aggiungono ulteriori informazioni sulle transizioni, utilizzando delle variabili simboliche su cui si possono porre delle condizioni dette *guardie*.

Definizione 3. *Un STS [3] è una tupla* $(S, Act, Var, \rightarrow, s_0)$ *dove*

- S *è un insieme di stati*
- Act *è un insieme di azioni*
- Var *è un insieme di variabile*

- $\rightarrow \subseteq S \times (Act \cup Var) \times \Phi \times S$ *è una relazione di transizione*
- $s_0 \in S$ *è lo stato iniziale*

Dove Φ è l'insieme di tutte e sole le *guardie* ϕ, ϕ' generabili tramite la seguente grammatica:

$$\phi ::= [\,] \mid [x = y] \mid \neg\phi \mid \phi \wedge \phi'$$

In breve una guardia può essere *vuota* ([ ]), può essere un vincolo di uguaglianza tra (le azioni denotate da) due variabili ($[x = y]$), può essere una negazione ($\neg\phi$) o infine la congiunzione di due sotto-guardie.

Sia $\delta : Var \rightarrow Act*$, indicheremo con $\delta \vdash \phi$ se ϕ è soddisfatta da δ. Diciamo che $\beta \equiv a_1 \dots a_n$ è una traccia di $t = (S, Act, Var, \rightarrow, i)$ sotto δ se $\exists s_0 \dots s_n$ *t.c.* $s_0 \xrightarrow{\alpha_1,\phi_1} s_1 \dots \xrightarrow{\alpha_n,\phi_n} s_n$ e $\forall j \in [1,n]$ *t.c.* $\delta \vdash \phi_j$ e $\delta(\alpha_j) = a_j$, $s_0 = i$.

Esempio 3. *Per mostrare il funzionamento di un STS, riprendiamo l'agente Filosofo dell'Esempio 2. Si consideri la Figura 2.4: siano $\overline{x}$ e $\overline{y}$ due simboli che vengono acquisiti, z e w due simboli che vengono rilasciati. La prima transizione ci dice che il Filosofo prende una forchetta, non importa quale. La seconda transizione invece ha una condizione, ovvero che la forchetta presa non sia la stessa acquisita al passaggio precedente. Dopo aver mangiato (stato m), il Filosofo rilascia una delle due forchette z o w e successivamente rilascia l'altra.*

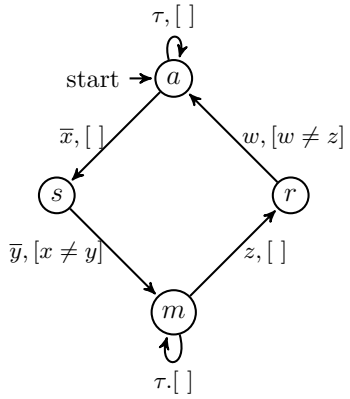


Figura 2.4: STS per un Filosofo

Mostriamo che la traccia presa da Es. 2 (cioè $\beta \equiv \bar{\mathrm{f}}_i \cdot \bar{\mathrm{f}}_{i+1} \cdot \tau \cdot \mathrm{f}_i \cdot \mathrm{f}_{i+1}$) è anche una traccia dell'STS t sotto $\delta = [x \leftarrow\!\shortmid \mathrm{f}_i, y \leftarrow\!\shortmid \mathrm{f}_{i+1}, z \leftarrow\!\shortmid \mathrm{f}_i, w \leftarrow\!\shortmid \mathrm{f}_{i+1}]$.

$$\beta \equiv a \xrightarrow{\overline{x},[\,]} s \xrightarrow{\overline{y},[x\neq y]} m \xrightarrow{\tau,[\,]} m \xrightarrow{z,[\,]} r \xrightarrow{w,[w\neq z]} a$$

Vediamo se δ soddisfa le guardie di t

$$\begin{aligned} \delta &\vdash [\,] \\ \delta \vdash [x \neq y] \text{ infatti } \delta(x) = \mathrm{f}_i \neq \mathrm{f}_{i+1} &= \delta(y) \\ \delta &\vdash [\,] \\ \delta &\vdash [\,] \\ \delta \vdash [w \neq z] \text{ infatti } \delta(w) = \mathrm{f}_i \neq \mathrm{f}_{i+1} &= \delta(z) \end{aligned}$$

Quindi otteniamo che la sequenza di stati diventa

$$\overline{\delta(x)} \cdot \overline{\delta(y)} \cdot \delta(\tau) \cdot \delta(z) \cdot \delta(w) =$$

$$= \bar{\mathrm{f}}_i \cdot \bar{\mathrm{f}}_{i+1} \cdot \tau \cdot \mathrm{f}_i \cdot \mathrm{f}_{i+1} \equiv \beta$$

□

2.1.4 Algebre di Processo

Le Algebre di Processo (in seguito PA, *Process Algebras*) [15], sono un insieme di formalismi che permettono di rappresentare algebricamente gli agenti ed i loro comportamenti. Gli elementi alla base delle PA sono *azioni* ed *operatori* e gli agenti si ottengono tramite la loro composizione. Negli ultimi decenni sono state proposte diverse PA. Alcuni esempi sono il *Communicating Sequential Process (CPS)* [7], il *Calculus of Communicating Systems (CCS)* [11] e il *π-calcolo* [12].

Definizione 4. *La sintassi del π-calcolo è definita come segue:*

$$\begin{aligned} P, Q &::= nil \mid \pi.P \mid X \mid P + Q \mid P|Q \mid recX.P \mid (\nu x)P \\ \pi &::= x!y \mid x?y \mid \tau \end{aligned}$$

nil è l'agente vuoto. $\pi.P$ rappresenta l'agente che esegue π e si riduce a P. X è una variabile che denota un processo. $P + Q$ è l'agente che si comporta come P o Q non deterministicamente. $P|Q$ è l'agente che corrisponde all'esecuzione sincrona di P e Q. $recX.P$ è l'agente che itera il comportamento di P, associandolo alla variabile X. Infine $(\nu x)P$ restringe l'utilizzo di x in P. π indica le azioni di invio e ricezione di un dato y sul canale x (rispettivamente $x!y$ e $x?y$). Si noti che dati e canali sono interscambiabili, ovvero si può inviare o ricevere un canale.

Un modo per definire la semantica operazionale del π-calcolo è in termini di LTS, cioè si definisce un insieme di regole che associano un LTS ad ogni processo.

Definizione 5. *Semantica LTS per il π-calcolo :*

$$ACT: \frac{}{\pi.P \xrightarrow{\pi} P} \qquad SUM1: \frac{P \xrightarrow{\pi} P'}{P+Q \xrightarrow{\pi} P'} \qquad SUM2: \frac{Q \xrightarrow{\pi} Q'}{P+Q \xrightarrow{\pi} Q'}$$

$$PAR1: \frac{P \xrightarrow{\pi} P'}{P|Q \xrightarrow{\pi} P'|Q} \qquad PAR2: \frac{Q \xrightarrow{\pi} Q'}{P|Q \xrightarrow{\pi} P|Q'}$$

$$PAR3: \frac{P \xrightarrow{x!y} P' \qquad Q \xrightarrow{x?z} Q'}{P|Q \xrightarrow{\tau} P'|Q'\{z/y\}}$$

$$REC: \frac{P\{recX.P/X\} \xrightarrow{\pi} P'}{recX.P \xrightarrow{\pi} P'} \qquad RES: \frac{P \xrightarrow{\pi} P'}{(\nu x)P \xrightarrow{\pi} (\nu x)P'}$$

□

In breve, un agente $\pi.P$ esegue l'azione π e si riduce a P (regola ACT). L'agente $P+Q$ può comportarsi (non deterministicamente) come P (regola $SUM1$) o come Q ($SUM2$). Invece un agente $P|Q$ può (i) effettuare un passo di P e ridursi a $P'|Q$ (regola $PAR1$), (ii) effettuare un passo di Q e ridursi a $P|Q'$ ($PAR2$) oppure (iii) effettuare una sincronizzazione tra P e Q (regola $PAR3$). In particolare la sincronizzazione tra due agenti avviene se essi inviano e ricevono (rispettivamente) su uno stesso canale (per esempio x). L'effetto della sincronizzazione è l'azione invisibile τ e il risultato è un nuovo agente in cui, oltre alla riduzione associata all'invio/ricezione, l'agente ricevente sostituisce al proprio parametro attuale (z) il valore ricevuto (y). Infine, un agente $recX.P$ si comporta come P in cui la variabile X viene sostituita dall'agente stesso (regola REC) mentre un agente $(\nu x)P$ può eseguire tutte le transizioni di P che non contengono il simbolo x.

Esempio 4. *Si consideri l'LTS di Es. 2. Diamo una corrispondente specifica in π-calcolo : tale agente è rappresentato dalla formula P_a. Per facilitare la leggibilità, sono state usate le seguenti abbreviazioni:*

$$\begin{aligned}
P_a &\equiv recP_a.(\tau.P_a + \bar{\mathrm{f}}_i.P_{s1} + \bar{\mathrm{f}}_{i+1}.P_{s2}) \\
P_m &\equiv recP_m.(\tau.P_m + \mathrm{f}_i.P_{r1} + \mathrm{f}_{i+1}.P_{r2}) \\
P_{s1} &\equiv \bar{\mathrm{f}}_{i+1}.P_m \qquad & P_{s2} &\equiv \bar{\mathrm{f}}_i.P_m \\
P_{r1} &\equiv \mathrm{f}_{i+1}.P_a \qquad & P_{r2} &\equiv \mathrm{f}_i.P_a
\end{aligned}$$

Inoltre, per semplicità di scrittura definiamo

$$Q = \tau.P_a + \bar{\mathrm{f}}_i.P_{s1} + \bar{\mathrm{f}}_{i+1}.P_{s2}$$

Applicando le regole della Definizione 2, mostriamo un passo dell'esecuzione di P_a:

$$\cfrac{\cfrac{\cfrac{-}{\bar{\mathrm{f}}_i, P_{s_1}\{recP_a.Q/P_a\} \xrightarrow{\bar{\mathrm{f}}_i} P_{s_1}\{recP_a.Q/P_a\}}(SUM)}{Q\{recP_a.Q/P_a\} \xrightarrow{\bar{\mathrm{f}}_i} P_{s_1}\{recP_a.Q/P_a\}}(REC)}{P_a \xrightarrow{\bar{\mathrm{f}}_i} P_{s_1}\{recP_a.Q/P_a\}}$$

Ripetendo lo stesso procedimento per i passi successivi, è facile notare che P_a *produce la traccia* β *dell'Es. 2.* □

2.2 Specifiche di sicurezza

Le specifiche di sicurezza modellano (insiemi di) comportamenti considerati *legali* (o, dualmente, *illegali*). Date una o più tracce di esecuzione (vedi Sezione 2.1.2), diciamo che una certa specifica è soddisfatta se le tracce in questione ricadono nell'insieme dei comportamenti legali. Esistono svariati formalismi per la definizione di specifiche di sicurezza. In questa sezione ne introduciamo brevemente alcune di interesse.

2.2.1 Security Automata

I *Security Automata* [17] (SA) sono una classe di FSA usati per definire proprietà di sicurezza. Intuitivamente un SA riconosce come legali le sequenze che appartengono al linguaggio da esso accettato.

Definizione 6. *Un SA è una tupla* (Q, Act, δ, q_0) *dove*

- Q *è un insieme di stati*
- Act *è un insieme di simboli di input*
- $\delta : (Q \times Act) \to Q$ *è una funzione di transizione*
- $q_0 \in Q$ *è lo stato iniziale*

Data una sequenza di simboli $\beta = \alpha_0, ...\alpha_n \in Act^*$*, diciamo che* β soddisfa *la proprietà rappresentata da SA se esiste uno stato* $q \in Q$ *tale che* $\delta(\delta...(\delta(q_0, \alpha_0), \alpha_1), ..., \alpha_n) = q$. □

Esempio 5. *Si consideri nuovamente l'agente LTS introdotto in Es. 2. Vogliamo definire la seguente proprietà che restringe il comportamento dell'agente. "Dopo aver acquisito entrambe le forchette* ($\bar{\mathrm{f}}_i$ *e* $\bar{\mathrm{f}}_{i+1}$)*, l'agente deve rilasciarne almeno una entro le successive due transizioni". Il SA che modella questa proprietà è dato in Figura 2.5.*

Brevemente, a partire dallo stato iniziale q_0 *l'automa si sposta in* q_1 *in risposta all'azione* $\bar{\mathrm{f}}_i$ *o* $\bar{\mathrm{f}}_{i+1}$ *mentre rimane in* q_0 *con* τ*. Da questo stato può compiere un'azione* f_i *o* f_{i+1} *e tornare in* q_0 *oppure può compiere* $\bar{\mathrm{f}}_i$ *o* $\bar{\mathrm{f}}_{i+1}$ *e raggiungere* q_2 *oppure in alternativa può compiere un'azione* τ *e rimanere in* q_1*. Dallo stato* q_2 *l'automa può andare in* q_1 *con* f_i *o* f_{i+1}*, oppure può compiere una transizione* τ *e finire in* q_3*. Infine, da* q_3 *l'automa può solo percorrere* f_i *o* f_{i+1} *e torna nello stato* q_1*. Si noti che l'automa non ammette transizioni etichettate con* τ *nello stato* q_3*. Quindi se in tale stato viene osservata l'azione* τ*, la traccia processata porta ad una violazione della proprietà.*

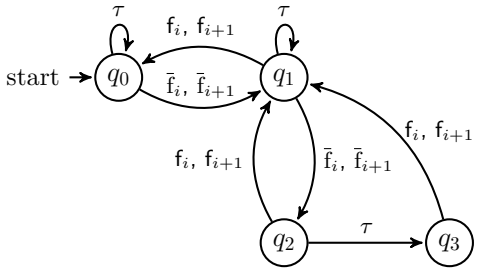


Figura 2.5: Security Automata per la proprietà di Esempio 5

Si consideri nuovamente la traccia

$$\beta \equiv \bar{\mathrm{f}}_i \cdot \bar{\mathrm{f}}_{i+1} \cdot \tau \cdot \mathrm{f}_i \cdot \mathrm{f}_{i+1}$$

di Es. 2. Mostriamo che β soddisfa SA. Infatti dalla Definizione 6 si può vedere che

$$(\delta(\delta(\delta(\delta(q_0, \bar{\mathrm{f}}_i), \bar{\mathrm{f}}_{i+1}), \tau), \mathrm{f}), \mathrm{f}_{i+1}) = \beta$$

□

2.2.2 Linear Temporal Logic

Linear Temporal Logic (LTL) è una logica modale introdotta da Amir Pnueli [16] nel 1976 per la specifica delle proprietà temporali. La sintassi di LTL consiste dei classici operatori della logica proposizionale, uniti ad alcuni operatori modali per la modellazione degli aspetti legati alle sequenze temporali.

Definizione 7. *Le formule LTL sono tutte e sole le formule che si possono ottenere con al seguente grammatica:*

$$\phi, \psi ::= p \mid \neg\phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid X\phi \mid \phi\ U\ \psi \mid G\phi$$

dove p appartiene ad un insieme di proposizioni atomiche AP. □

Brevemente, una formula LTL può essere una proposizione atomica p, la negazione di una sottoformula ϕ, la disgiunzione o congiunzione di due formule ϕ e ψ. Inoltre una formula LTL può essere ottenuta applicando le modalità temporali X, U e G.

La validità di una formula LTL (contro una traccia di esecuzione) dipende dall'interpretazione delle sue proposizioni atomiche. In particolare assumiamo che sia definita una funzione $\mathcal{I}$ che associa ai simboli di una traccia le proposizioni atomiche valide per quel simbolo. Per semplicità, nel seguito assumeremo (quando non diversamente specificato) $\mathcal{I}(p) = \{p\}$.

Definizione 8. *Data una formula LTL ϕ e una traccia di esecuzione β diciamo che β soddisfa ϕ (in simboli $\beta \models \phi$) se valgono le seguenti relazioni.*

- $\beta \models p$ *sse* $p \in \mathcal{I}(\beta(0))$
- $\beta \models \neg\phi$ *sse* $\beta \not\models \phi$
- $\beta \models \phi \vee \psi$ *sse* $\beta \models \phi$ *oppure* $\beta \models \psi$
- $\beta \models \phi \wedge \psi$ *sse* $\beta \models \phi$ *e* $\beta \models \psi$
- $\beta \models X\phi$ *sse* $\beta_1 \models \phi$
- $\beta \models \phi\ U\ \psi$ *sse* $\exists\ i \geq 0$ *t.c.* $\beta_i \models \psi$ *e* $\forall\ 0 \leq k < i,\ \beta_k \models \phi$
- $\beta \models G\ \phi$ *sse* $\forall\ i \geq 0$ *t.c.* $\beta_i \models \phi$

□

Informalmente, β soddisfa p se e solo se p appartiene alle proposizioni che sono vere per $\beta(0)$, β soddisfa $\neg\phi$ se e solo se β non soddisfa ϕ. Inoltre β soddisfa rispettivamente $\phi \vee \psi$ se β soddisfa almeno una tra ϕ e ψ. Allo stesso modo, β soddisfa $\phi \wedge \psi$ se entrambe ϕ e ψ sono soddisfatte. Per quanto riguarda gli operatori temporali, $X\phi$ è soddisfatta dalle tracce β tali per cui il suffisso $beta_1$ soddisfa ϕ, $\phi\ U\ \psi$ è soddisfatta se esiste in indice i tale per cui il suffisso di β a partire da i soddisfa ψ, mentre β soddisfa $G\ \phi$ se ogni suo suffisso, compresa β soddisfa ϕ.

Esempio 6. *Consideriamo la seguente formula LTL, riferita all'agente in figura 2.3*

$$\phi \equiv G(\neg m \vee (m \wedge X\ \neg m))$$

Questa formula, in linguaggio naturale, significa che globalmente non vogliamo che il Filosofo si trovi due volte consecutive nello stato m.

Verifichiamo se la traccia $\beta \equiv a \cdot s_1 \cdot m \cdot m \cdot r_1 \cdot a$ soddisfa ϕ.

Applicando le regole della semantica (vedi Definizione 8) otteniamo

$$\beta \models \phi \iff \forall i \geq 0.\beta_i \models (\neg m \vee (m \wedge X\neg m))$$

Dato che β è una traccia finita, possiamo riscrivere questa relazione come segue

$$\beta \models \phi \iff \bigwedge_{i=0}^{5} (\beta_i \models \neg m \vee (m \wedge X\neg m))$$

Risolvendo i sotto casi separatamente otteniamo

- $\beta_0 \models (\neg m \vee (m \wedge X\neg m))$. *Infatti* $\beta_0 \models \neg m$ *dato che* $\beta_0(0) = a \neq m$ *e quindi* $\beta_0 \not\models m$.
- $\beta_1 \models (\neg m \vee (m \wedge X\neg m))$. *Per lo stesso ragionamento del passo precedente, semplicemente sostituendo* s_1 *ad* a.

- $\beta_2 \not\models (\neg m \vee (m \wedge X \neg m))$. *In questo caso,* $\beta_2 \not\models \neg m$ *(dato che* $\beta_2(0) = m$*). Inoltre* $\beta_2 \not\models m \wedge X \neg m$*. Infatti* $\beta_2 \not\models X \neg m$ *come segue immediatamente dal fatto che* $\beta_3 \not\models \neg m$*, essendo* $\beta_3(0) = m$*.*

Dato che il soddisfacimento di ϕ *equivale alla congiunzione dei casi precedenti (oltre ai rimanenti non svolti), possiamo concludere che* $\beta \not\models \phi$*.* □

Chiaramente, la nozione di validità può essere estesa ad insiemi di tracce come segue

$$B \models \phi \Longleftrightarrow \forall \beta \in B . \beta \models \phi$$

2.2.3 μ-calcolo

Il μ-calcolo modale è un'estensione della logica modale proposizionale che vanta la proprietà di essere un linguaggio Turing equivalente, cioè ha la capacità di simulare una qualunque macchina di Turing. E' stato sviluppato da Dana Scott e Jaco de Bakker, mentre la forma usata ai giorni nostri deriva dal lavoro di Dexter Kozen [8].

Definizione 9. *La sintassi del* μ*-calcolo è così definita:*

$$\psi ::= F \mid \neg\psi \mid \psi_0 \vee \psi_1 \mid \langle\alpha\rangle\psi \mid X \mid \mu X . \psi$$

□

F è la formula che non è mai soddisfatta da nessuno stato, $\neg\psi$ è la formula che è soddisfatta da tutti e soli gli stati che non soddisfano ψ, $\psi_0 \vee \psi_1$ rappresenta la formula che è soddisfatta da tutti gli stati che soddisfano ψ_0 oppure ψ_1, $\langle\alpha\rangle\psi$ è la formula verificata dagli stati tali per cui esiste almeno un'α-transizione che porta ad uno stato per cui ψ è soddisfatta. Infine $\mu X.\psi$ è la formula soddisfatta dagli stati che ricadono nell'insieme che è minimo punto fisso delle soluzioni (cioè degli stati che soddisfano) ψ ricorrendo su X. Inoltre introduciamo le seguenti abbreviazioni standard. $T \equiv \neg F$, $\psi_0 \wedge \psi_1 \equiv \neg(\neg\psi_0 \vee \neg\psi_1)$, $[\alpha]\psi \equiv \neg\langle\alpha\rangle\neg\psi$ ed infine $\nu X.\psi \equiv \neg\mu X.\neg\psi[\neg X/X]$.

Formalmente, la semantica (denotazionale) del μ-calcolo può essere data come segue.

Definizione 10. *Dato un LTS* $t = (S, Act, \rightarrow)$ *ed un* mapping ρ *da variabili in insiemi di stati di LTS,* $[\![\cdot]\!]_\rho : \phi \rightarrow 2^S$ *è la funzione definita dalle regole seguenti.*

- $[\![F]\!]^t_\rho = \emptyset$
- $[\![\phi \vee \psi]\!]^t_\rho = [\![\phi]\!]^t_\rho \cup [\![\psi]\!]^t_\rho$
- $[\![\neg\phi]\!]^t_\rho = S \setminus [\![\phi]\!]^t_\rho$
- $[\![\langle\alpha\rangle\phi]\!]^t_\rho = \{s \in S \mid \exists s' . s \xrightarrow{\alpha} s' \wedge s' \in [\![\phi]\!]^t_\rho\}$
- $[\![X]\!]^t_\rho = \rho(X)$

- $[\![\mu Z.\phi]\!]^t_\rho = \bigcap\{U \subseteq S \mid [\![\phi]\!]^t_{\rho[Z \leftarrow\!\rightarrow U]} \subseteq U\}$

F non è verificata in nessuno stato, $\phi \vee \psi$ è la formula verificata in tutti gli stati in cui ϕ o ψ sono verificate. $\neg\phi$ è verificata in tutti gli stati in cui ϕ non è verificata. $\langle\alpha\rangle\phi$ è verificata in uno stato s se esiste un'α-transizione che da s porta in uno stato dove ϕ è verificata. X è verificato in tutti gli stati in cui sono verificati tutti gli stati successivi a X. $\mu Z.\phi$ è verificata in ogni stato in ogni insieme U tale che quando la variabile Z è fissata a U, allora ϕ è verificata per ogni U, dove $\rho[Z \leftarrow\!\rightarrow U]$ mappa Z in U preservando le corrispondenze di ρ altrove.

Date le precedenti, scriviamo che

$$t \models \phi \Leftrightarrow s_0 \in [\![\phi]\!]^t_\emptyset$$

cioè, dato un LTS $t = (S, Act, \rightarrow, s_0)$ esso soddisfa ϕ se e solo se s_0 appartiene all'insieme di stati $[\![\phi]\!]^t_\emptyset$. □

Esempio 7. *Si consideri la seguente formula (corrispondente della specifica data in LTL nell'Esempio 6).*

$$\phi = \mu X.(([\overline{f}_i][\overline{f}_{i+1}][\tau]F \wedge [\overline{f}_{i+1}][\overline{f}_i][\tau]F) \wedge [\tau]X)$$

Si consideri il LTS $t_i = (S, Act, \rightarrow, s_0)$ di Esempio 2.3, utilizzando le formule della semantica denotazionale in Definizione 10 e dato lo stato di partenza a, vogliamo verificare se $t_i \models \phi$ cioè se $a \in [\![\phi]\!]^{t_i}_\emptyset$.

Iniziamo applicando la regola per gli operatori di punto fisso, da cui ci riduciamo al problema di verificare

$$a \in [\![\mu X.(([\overline{f}_i][\overline{f}_{i+1}][\tau]F \wedge [\overline{f}_{i+1}][\overline{f}_i][\tau]F) \wedge [\tau]X)]\!]^{t_i}_\emptyset$$

che vale se e solo se

$$a \in U^* = \bigcap\{U \subseteq S \mid [\![(([\overline{f}_i][\overline{f}_{i+1}][\tau]F \wedge [\overline{f}_{i+1}][\overline{f}_i][\tau]F) \wedge [\tau]X)]\!]^{t_i}_{\emptyset[X \leftarrow\!\rightarrow U]} \subseteq U\}$$

L'insieme U^* può essere calcolato applicando il Teorema del punto fisso di Tarski [19]. Quindi calcoliamo induttivamente il punto fisso della catena U_n come segue.

- *$U_0 = S$. In questo caso abbiamo*

  $$U_1 = [\![([\overline{f}_i][\overline{f}_{i+1}][\tau]F \wedge [\overline{f}_{i+1}][\overline{f}_i][\tau]F) \wedge [\tau]X]\!]^{t_i}_{\emptyset[X \leftarrow\!\rightarrow U_0]}$$

  Applicando la regola per la congiunzione otteniamo

  $$U_1 = [\![[\overline{f}_i][\overline{f}_{i+1}][\tau]F \wedge [\overline{f}_{i+1}][\overline{f}_i][\tau]F]\!]^{t_i}_{\emptyset[X \leftarrow\!\rightarrow U_0]} \cap [\![[\tau]X]\!]^{t_i}_{\emptyset[X \leftarrow\!\rightarrow U_0]}$$

  Applicando la regola per la modalità $[\cdot]$ (immediatamente inferibile da quella per $\langle\cdot\rangle$) al lato destro della disgiunzione otteniamo

  $$[\![[\tau]X]\!]^{t_i}_{\emptyset[X \leftarrow\!\rightarrow U_0]} = \{s \in S \mid \forall s'.s \xrightarrow{\tau} s' \Longrightarrow s' \in [\![X]\!]^{t_i}_{\emptyset[X \leftarrow\!\rightarrow U_0]}\}$$

che può essere ulteriormente semplificata in

$$[\![[\tau]X]\!]^{t_i}_{\emptyset[X\leftarrow U_0]} = \{s \in S \mid \forall s'.s \xrightarrow{\tau} s' \Longrightarrow s' \in S\}$$

Dalla definizione di t_i segue immediatamente

$$[\![[\tau]X]\!]^{t_i}_{\emptyset[X\leftarrow U_0]} = S$$

Invece il lato sinistro può essere ulteriormente ridotto utilizzando la regola per la congiunzione.

$$[\![[\overline{f}_i][\overline{f}_{i+1}][\tau]F \wedge [\overline{f}_{i+1}][\overline{f}_i][\tau]F]\!]^{t_i}_{\emptyset[X\leftarrow U_0]} =$$

$$[\![[\overline{f}_i][\overline{f}_{i+1}][\tau]F]\!]^{t_i}_{\emptyset[X\leftarrow U_0]} \cap [\![[\overline{f}_{i+1}][\overline{f}_i][\tau]F]\!]^{t_i}_{\emptyset[X\leftarrow U_0]}$$

La prima delle due sotto formule può essere ridotta come segue (non mostriamo le riduzioni per la seconda formula, ma esse sono facilmente derivabili per simmetria).

$$[\![[\overline{f}_i][\overline{f}_{i+1}][\tau]F]\!]^{t_i}_{\emptyset[X\leftarrow U_0]} = \{s \in S \mid \forall s'.s \xrightarrow{\overline{f}_i} s' \Longrightarrow s' \in [\![[\overline{f}_{i+1}][\tau]F]\!]^{t_i}_{\emptyset[X\leftarrow U_0]}$$

Ripetendo lo stesso ragionamento e riapplichiamo due volte la regola per $[\cdot]$ e infine la regola per F in modo da ottenere

$$[\![[\overline{f}_i][\overline{f}_{i+1}][\tau]F]\!]^{t_i}_{\emptyset[X\leftarrow U_0]} = \{s \in S \mid \forall s', s'', s'''.s \xrightarrow{\overline{f}_i} s' \xrightarrow{\overline{f}_{i+1}} s'' \xrightarrow{\tau} s''' \Longrightarrow s''' \in \emptyset\}$$

Questo insieme può essere ridefinito come

$$\{s \in S \mid \nexists s', s'', s'''.s \xrightarrow{\overline{f}_i} s' \xrightarrow{\overline{f}_{i+1}} s'' \xrightarrow{\tau} s'''\}$$

Osservando le transizioni di t_i in riferimento alla Figura 2.3, questo insieme si riduce a $S \setminus \{a\}$. Per concludere il caso, basta osservare che $U_1 = (S \setminus \{a\}) \cap (S \setminus \{a\}) \cap S = S \setminus \{a\}$.

- *U_1. Senza ripetere i passaggi precedenti, si noti che, dato che essa non dipende dalla variabile X, le formula $[\overline{f}_i][\overline{f}_{i+1}][\tau]F \wedge [\overline{f}_{i+1}][\overline{f}_i][\tau]F$ denota lo stesso insieme di stati del caso precedente ($S \setminus \{a\}$). L'unica differenza è data dalla seguente riduzione*

$$[\![[\tau]X]\!]^{t_i}_{\emptyset[X\leftarrow U_1]} = \{s \in S \mid \forall s'.s \xrightarrow{\tau} s' \Longrightarrow s' \in S \setminus \{a\}\}$$

Di conseguenza $U_2 = (S \setminus \{a\}) \cap (S \setminus \{a\}) = S \setminus \{a\}$. E quindi il minimo punto fisso $U^* = U_1$.

Concludendo, osserviamo che $a \notin U^*$ e quindi $t_i \not\models \phi$. □

2.3 Model Checking

Il Model Checking [4] (MC nel seguito) è una tecnica automatica per la verifica di modelli finiti. Ha lo scopo di verificare che un modello dato soddisfi una specifica. Modelli e specifiche possono essere descritti tramite i formalismi visti nelle sezioni precedenti. Formalmente, data una proprietà ψ, un modello M ed un suo stato s, si vuole verificare se, a partire da s, M soddisfa ψ, in simboli

$$M, s \models \psi$$

Se il MC ha successo, si ha la garanzia che il modello soddisfa la data proprietà. Altrimenti, il MC restituisce una traccia che viola la specifica e che appartiene ai possibili comportamenti descritti dal modello. In seguito, la traccia può essere utilizzata in vari modi, per esempio per guidare procedure automatiche di testing o per riprogettare il sistema in analisi.

Il MC si può effettuare con diversi approcci algoritmici. Alcuni esempi sono la *State Space Enumeration* in cui viene eseguita una ricerca esaustiva (*Breadth First Search* [13] oppure *Depth First Search*) sugli stati del modello oppure metodi basati su *Language Inclusion* o sulla verifica di relazioni di *simulazione* o *bisimulazione* [10].

Esempio 8. *Si consideri nuovamente la proprietà espressa in LTL dell'Esempio 6*

$$\phi = G(\neg m \vee (m \wedge X\neg m))$$

Inoltre si consideri il TS t di Esempio 1. Applichiamo la procedura di ricerca in profondità (Depth First Search) per verificare se $t, a \models \phi$.

Vogliamo verificare che

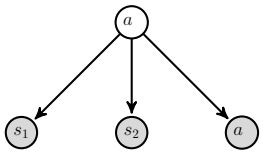
$$t, a \models G(\neg m \vee (m \wedge X\neg m))$$

Indichiamo con $FS(p)$ *la funzione che restituisce tutti i nodi raggiungibili da p con una transizione (*forward star *di p), altrimenti definita come* $FS(p) = \{q \mid p \rightarrow q\}$.

Ad ogni passo la procedura $MC(q, \beta, \bar{\phi})$ *considera lo stato corrente q, il percorso* β *dallo stato iniziale fino allo stato corrente e la proprietà da verificare* $\bar{\phi}$. *Il risultato di tale valutazione può essere* (i) *la soluzione del problema del MC o* (ii) *una o più chiamate ricorsive della procedura su nuove (sotto) istanze del problema (ad ogni passo evidenziamo tramite sottolineatura quale istanza viene valutata al passo seguente, inoltre etichettiamo con* ▲ *e* ▼ *i casi banalmente verificati positivamente e negativamente). Il procedimento si svolge come mostrato nei seguenti passi.*

- *Configurazione:* $q = a$, $\beta = \cdot$, $\bar{\phi} = \phi$.

$$FS(q) = FS(a) = \{s_1, s_2, a\}$$

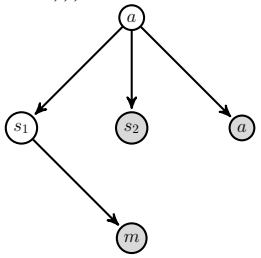


Nuove istanze (congiunzione):

$MC(s_1, a, \neg m \vee (m \wedge X\neg m))$ ▲
$MC(s_2, a, \neg m \vee (m \wedge X\neg m))$ ▲
$MC(a, a, \neg m \vee (m \wedge X\neg m))$ ▲
$\underline{MC(s_1, a, G(\neg m \vee (m \wedge X\neg m)))}$
$MC(s_2, a, G(\neg m \vee (m \wedge X\neg m)))$
$MC(a, a, G(\neg m \vee (m \wedge X\neg m)))$

- *Configurazione:* $q = s_1$, $\beta = a$, $\bar{\phi} = G(\neg m \vee (m \wedge X\neg m)))$.

$$FS(s_1) = \{m\}$$

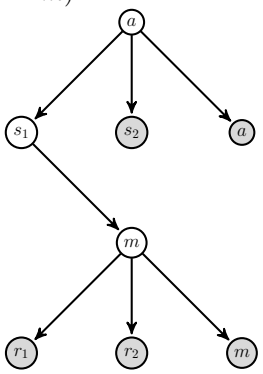


Nuove istanze (congiunzione):

$$\frac{MC(m, a \cdot s_1, \neg m \vee (m \wedge X\neg m))}{MC(m, a \cdot s_1, G(\neg m \vee (m \wedge X\neg m)))}$$

- *Configurazione:* $q = m$, $\beta = a \cdot s_1$, $\bar{\phi} = \neg m \vee (m \wedge X\neg m)$.

$$FS(m) = \{r_1, r2, m\}$$



Nuove istanze (disgiunzione):

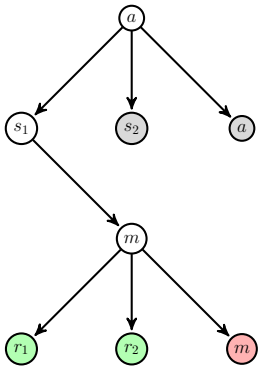
$MC(m, a \cdot s_1, \neg m)$ ▼
$\underline{MC(m, a \cdot s_1, (m \wedge X\neg m))}$

- *Configurazione:* $q = m$, $\beta = a \cdot s_1$, $\bar{\phi} = m \wedge X\neg m$.

$$FS(m) = \{r_1, r2, m\}$$

Nuove istanze (congiunzione):

$MC(m, a \cdot s_1, m)$ ▲
$MC(r_1, a \cdot s_1 \cdot m, \neg m)$ ▲
$MC(r_2, a \cdot s_1 \cdot m, \neg m)$ ▲
$MC(m, a \cdot s_1 \cdot m, \neg m)$ ▼



Di conseguenza la procedura termina individuando come traccia violante la sequenza composta da $\beta \cdot q = a \cdot s_1 \cdot m \cdot m$. □

2.3.1 Costo computazionale

Fin dalle prime proposte di algoritmi per model checking, il problema del costo computazionale di tali procedure ha ricevuto particolare attenzione. Molti autori hanno effettuato analisi della complessità computazionale degli algoritmi per model checking mostrando che, sotto assunzioni comuni, non esistono procedure efficienti per la risoluzione del problema. Uno dei primi studi sul MC del μ-calcolo è stato [5], in cui si mostra un algoritmo per il model checking, di complessità $O((mn)^{l+1})$, dove m è la lunghezza della formula, n è la dimensione del modello ed l è il numero massimo di operatori di minimo e massimo punto fisso annidati. Da qui deriva che, per quanto riguarda la complessità computazionale, la dimensione della politica domina quella del modello in analisi. Anche se altri algoritmi sono stati proposti in letteratura, questa considerazione rimane di carattere generale per ognuno di essi.

Da un punto di vista pragmatico, alcuni autori (per esempio [9]) hanno osservato come generalmente le istanze del problema del model checking trattino la verifica di modelli di grandi dimensioni contro formule piuttosto compatte. Di conseguenza, anche se teoricamente è la formula a contribuire maggiormente alla complessità del problema del model checking, in molti casi di interesse l'intrattabilità deriva dall'eccessiva dimensione dei modelli considerati.

In questo senso la principale causa della rapida crescita dei modelli è data dalla necessità di comporre in parallelo il comportamento di diversi agenti che compongono un unico sistema. In presenza di parallelismo, la dimensione dei modelli cresce esponenzialmente con il numero degli agenti considerati. Tale fenomeno è noto come *esplosione combinatoria degli stati*.

Capitolo 3

Filosofi al Buffet: un caso di studio

Allo scopo di illustrare le tecniche presentate in questa tesi, introduciamo il seguente caso di studio. Un certo numero di filosofi partecipa ad un buffet. I *Filosofi al buffet* attendono di fronte ad un tavolo. Sul tavolo sono collocati k forchette e k piatti. Ad ogni istante un filosofo p può avvicinarsi al tavolo (azione $join!p$) per mangiare. Prima di mangiare (azione $eat!p$) il filosofo deve raccogliere esattamente un piatto (azione $d_i!p$ con $i \in \{1, \ldots, k\}$) ed una forchetta (azione $f_i!p$ con $i \in \{1, \ldots, k\}$). Quando un filosofo detiene una forchetta (un piatto, rispettivamente) può rilasciarla (appoggiandola sul tavolo) tramite l'azione duale $f_i?p$ ($d_i?p$ rispettivamente)[1]. Come per il caso dei Filosofi a cena, i filosofi possono essere modellati tramite sistemi di transizione. In Figura 3.1 e in Figura 3.2 mostriamo due possibili STS di Filosofi al buffet. Anche se entrambi ricadono nella definizione generale del comportamento di un filosofo al buffet, essi hanno caratteristiche diverse (che saranno discusse nel seguito).

Inoltre possiamo modellare tramite un opportuno STS anche l'agente che rappresenta il comportamento del tavolo. Chiaramente la struttura di tale agente dipende dal numero di piatti e forchette (il valore di k). Per esempio, in Figura 3.3 mostriamo il STS per il tavolo con un piatto ed una forchetta ($k = 1$). Si noti che tale STS sfrutta le condizioni sui nomi simbolici per disambiguare tra le transizioni causate da agenti (filosofi) distinti.

Una possibile proprietà di interesse (mutuata dal problema dei filosofi a cena) è l'assenza di deadlock (vedi Esempio 1). Una variante di tale proprietà è l'assenza di *starvation*, cioè quella condizione per cui uno o più agenti, pur non essendo in deadlock, non riescono ad accedere ad una determinata risorsa o a raggiungere un loro determinato stato. Per esempio possiamo modellare l'assenza di starvation per un filosofo x rispetto all'azione $eat!$ con la seguente formula del μ-calcolo .

[1] Nel seguito di questa tesi, useremo la notazione $a!b$ e $a?b$ in alternativa a $a(b)$ e $\bar{a}(b)$ (rispettivamente) per le azioni parametriche ed i loro duali.

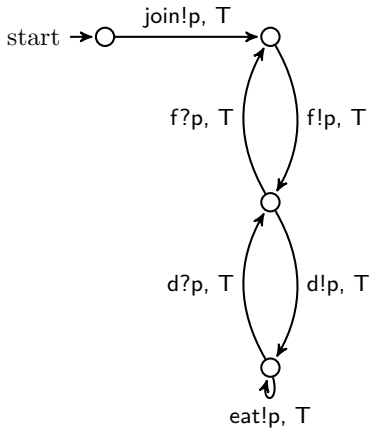


Figura 3.1: STS per il filosofo ingordo

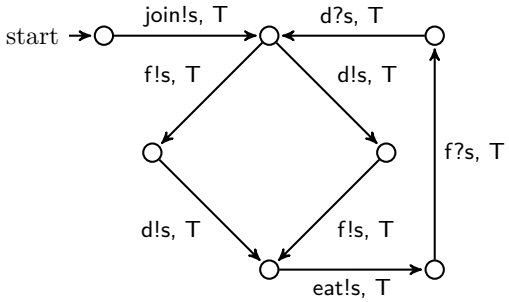


Figura 3.2: STS per il filosofo testardo

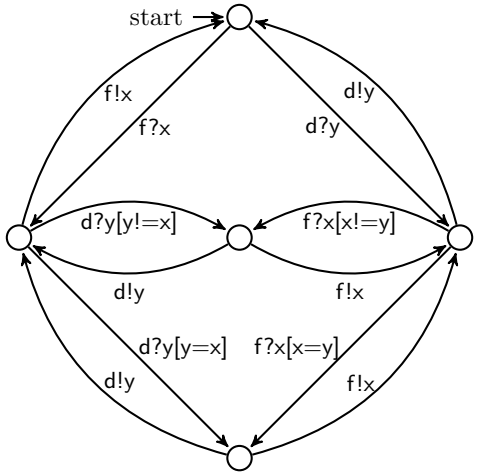


Figura 3.3: STS per il tavolo

$$\phi \equiv \mu X.(\langle join?x \rangle (\mu Y.(\langle \tau \rangle Y \vee \langle eat?x \rangle T) \vee \langle \tau \rangle X)$$

Negli esempi presentati nel resto della tesi, ci riferiremo agli elementi di questa sezione per illustrare le metodologie utilizzate.

Capitolo 4

Symbolic Partial Model Checking

In questa sezione presentiamo il *Symbolic Partial Model Checking*. In particolare in Sez. 4.1 iniziamo introducendo il *Partial Model Checking* come presentato in [1]. Quindi, in Sez. 4.2 mostriamo come il Symbolic Partial Model Checking viene ottenuto tramite l'estensione del Partial Model Checking.

4.1 Partial Model Checking

Il *Partial Model Checking* [1] (nel seguito *PMC*) è una tecnica di trasformazione per convertire istanze del problema del MC. Più precisamente l'idea alla base del PMC consiste nel rimuovere (sotto alcune condizioni) dai modelli gli operatori di composizione parallela, che sono la principale causa del fenomeno dell'esplosione combinatoria degli stati (vedi Sez. 2.3.1). La rimozione del parallelismo avviene tramite un processo di valutazione parziale che causa una specializzazione delle proprietà di sicurezza. Anche se da un punto di vista puramente teorico il PMC non evita il problema dell'esplosione combinatoria degli stati, in molti casi concreti esso può effettivamente abbattere il costo computazionale per il MC.

4.1.1 μ-calcolo equazionale

La procedura di valutazione parziale alla base del PMC si applica alle *asserzioni* (cioè le formule prive di operatori di punto fisso) del μ-calcolo . Per evitare l'annidamento arbitrario degli operatori di punto fisso, viene utilizzata la rappresentazione in formato *equazionale*. Le due rappresentazioni hanno lo stesso potere espressivo (cioè ogni formula del μ-calcolo può essere portata in formato equazionale e viceversa), però il μ-calcolo equazionale separa gli operatori di punto fisso dalle asserzioni.

Definizione 11. *I sistemi del μ-calcolo equazionale E, E' sono tutti e soli quelli generabili tramite la seguente grammatica.*

$$E ::= X =_\mu AE' \mid X =_\nu AE' \mid \epsilon$$

dove ϵ è la sequenza vuota, X, Y sono nomi di variabili (nel senso di Def. 9), μ e ν sono i due operatori di punto fisso e A sono asserzioni generate con la grammatica di Def. 9 escluse le produzioni per gli operatori di punto fisso. Inoltre è richiesto che tutte le variabili X, Y che appaiono in tutti i lati destri (cioè nelle asserzioni) delle equazioni di E appaiano una ed una sola volta nei lati sinistri.

Una formula del μ-calcolo equazionale è una coppia $(E \downarrow X)$ dove E è un sistema di equazioni ed X è la variabile iniziale. □

Informalmente, una specifica nel μ-calcolo equazionale si compone di un sistema finito di equazioni della forma

$$E = \begin{cases} X_1 =_\lambda A_1 \\ \vdots \\ X_n =_\lambda A_n \end{cases}$$

dove $\lambda \in \{\mu, \nu\}$. La variabile iniziale indica l'equazione di partenza per la valutazione della specifica. Con questa rappresentazione ogni equazione rappresenta esattamente una operazione di punto fisso. Anche se non presente esplicitamente, l'annidamento di tali operazioni è rappresentato dalla possibilità di avere nelle asserzioni variabili, cioè puntatori ad altre equazioni.

Possiamo quindi definire la semantica del μ-calcolo equazionale in maniera simile a quanto fatto in Def. 10 per il μ-calcolo modale.

Definizione 12. *Dato un LTS $t = (S, Act, \rightarrow, s_0)$ e un mapping ρ da variabili in 2^S, definiamo $[\![A]\!]^t_\rho$ come la funzione induttivamente definita dalle seguenti regole.*

- $[\![F]\!]^t_\rho = [\ ]$
- $[\![\neg A]\!]^t_\rho = S \setminus [\![A]\!]^t_\rho$
- $[\![A \vee A']\!]^t_\rho = [\![A]\!]^t_\rho \cup [\![A']\!]^t_\rho$
- $[\![\langle\alpha\rangle A]\!]^t_\rho = \{s \in S \mid \exists s'.s \xrightarrow{\alpha} s' \wedge s' \in [\![A]\!]^t_\rho\}$
- $[\![X]\!]^t_\rho = \rho(X)$

Abusando leggermente della notazione, definiamo $[\![E]\!]^t_\rho$ tramite le seguenti regole.

- $[\![\epsilon]\!]^t_\rho = [\ ]$
- $[\![X =_\lambda AE]\!]^t_\rho = [X \leftarrow\!\shortmid U'] \star [\![E]\!]^t_{(\rho \star [X \leftarrow\!\shortmid U'])}$

dove $U' = \lambda U.[\![A]\!]^t_{(\rho\star[X\leftarrow\!\!\shortmid U]\star\rho'(U))}$ *e* $\rho'(U) = [\![E]\!]^t_{(\rho\star[X\leftarrow\!\!\shortmid U])}$ *in cui il simbolo* $\star$ *è utilizzato per evidenziare l'operazione di composizione del mapping, per evitare ambiguità di lettura.*

Infine definiamo $[\![(E \downarrow X)]\!]^t = [\![E]\!]^t_{[\ ]}(X)$*. Valgono le stesse semplificazioni della Definizione 10.* □

Come per il μ-calcolo modale (vedi Sez. 2.2.3), anche per quello equazionale l'interpretazione di un'asserzione A risulta in un sottoinsieme degli stati di S (cioè gli stati che soddisfano A). Per esempio, nessuno stato soddisfa F (cioè $[\![F]\!]^t_\rho = \emptyset$) e una disgiunzione di due asserzioni è soddisfatta da tutti gli stati che soddisfano almeno una delle due.

Invece l'interpretazione di un sistema di equazioni E produce un mapping ρ che assegna alle variabili di E quei sottoinsiemi di S che soddisfano le asserzioni ad esse associate. In generale un sistema E genera un mapping i cui assegnamenti soddisfano simultaneamente tutte le equazioni.

Infine, possiamo utilizzare il mapping generato dal sistema E per dare un'interpretazione della specifica $(E \downarrow X)$. In generale diciamo che un LTS $t = (S, Act, \rightarrow, s_0)$ soddisfa una specifica $(E \downarrow X)$ se e solo se $s_0 \in [\![(E \downarrow X)]\!]^t$. In tal caso scriviamo $t \models (E \downarrow X)$.

Esempio 9. *Si consideri il seguente sistema composto da una sola equazione (equivalente alla formula di Esempio 7).*

$$E = \{X =_\mu ([\overline{f}_i][\overline{f}_{i+1}][\tau]F \wedge [\overline{f}_{i+1}][\overline{f}_i][\tau]F) \wedge [\tau]X\}$$

Verifichiamo se $t_i \models (E \downarrow X)$ *(dove* t_i *è il LTS di Esempio 2) applicando le regole di Definizione 12. Per prima cosa calcoliamo*

$$[\![X =_\mu ([\overline{f}_i][\overline{f}_{i+1}][\tau]F \wedge [\overline{f}_{i+1}][\overline{f}_i][\tau]F) \wedge [\tau]X; \epsilon]\!]^{t_i}_{[\ ]} = [X \leftarrow\!\!\shortmid U'] \star [\ ]$$

Per definizione $U' = \mu U.[\![([\overline{f}_i][\overline{f}_{i+1}][\tau]F \wedge [\overline{f}_{i+1}][\overline{f}_i][\tau]F) \wedge [\tau]X]\!]^{t_i}_{[X\leftarrow\!\!\shortmid U]}$*. Chiaramente* U' *può essere ottenuta applicando il Teorema di Tarski come fatto in Esempio 7, da cui segue (senza mostrare nuovamente i passaggi)* $U' = S \setminus \{a\}$*. Quindi abbiamo che* $[\![E]\!]^{t_i}_{[\ ]} = [X \leftarrow\!\!\shortmid S \setminus \{a\}]$*. Dato che* $s_0 \notin [X \leftarrow\!\!\shortmid S \setminus \{a\}](X) = [\![E \downarrow X]\!]^{t_i}$*, concludiamo che* $t_i \not\models (E \downarrow X)$*.* □

4.1.2 Quotienting

L'operatore di *quotienting* [1] $(E \downarrow X)/\!\!/_{L,M}\ t$ è definito tramite una funzione che trasforma una specifica $(E \downarrow X)$ eseguendone la valutazione parziale contro un LTS t. Le variabili L e M indicano rispettivamente l'insieme delle azioni di t che sono *visibili* e *mascherate*. Brevemente, se t è ristretto sulle azioni visibili L questo significa che tutte le azioni non in L (anche indicato da $\bar{L}$) esclusa l'azione τ, non possono essere generate da t. Invece tutte le azioni generate da t e contenute in M vengono trasformate in τ.

La proprietà fondamentale dell'operatore di quotienting è la seguente.

$$(E \downarrow X)/\!\!/_{L,M}\ t = (E/\!\!/_{L,M}\ t) \downarrow X_i$$

$$\epsilon/\!\!/_{L,M}\ t = \epsilon$$

$$(X =_\sigma\ A\ E)/\!\!/_{L,M}\ t = \begin{cases} X_{s_1} & =_\sigma\ A/\!\!/_{L,M}\ s_1 \\ \ldots & \\ X_{s_n} & =_\sigma\ A/\!\!/_{L,M}\ s_n \\ E/\!\!/_{L,M}\ t & \end{cases}$$

$$X/\!\!/_{L,M}\ s = X_s$$

$$\langle\alpha\rangle A/\!\!/_{L,M}\ s = \langle\alpha\rangle(A/\!\!/_{L,M}\ s)\ \vee \bigvee_{s \xrightarrow{\alpha} s'} A/\!\!/_{L,M}\ s' \text{ se } \alpha \neq \tau$$

$$\langle\tau\rangle A/\!\!/_{L,M}\ s = \langle\tau\rangle(A/\!\!/_{L,M}\ s)\ \vee \bigvee_{s \xrightarrow{\tau} s'} A/\!\!/_{L,M}\ s'\ \vee \bigvee_{\beta \in \overline{L} \cap M, s \xrightarrow{\beta} s'} \langle\overline{\beta}\rangle(A/\!\!/_{L,M}\ s')$$

$$(A_1 \vee A_2)/\!\!/_{L,M}\ s = (A_1/\!\!/_{L,M}\ s)\ \vee\ (A_2/\!\!/_{L,M}\ s)$$

$$F/\!\!/_{L,M}\ s = F$$

Tabella 4.1: Regole di quotienting

Teorema 1. *([1]) Dati due LTS t e t' e una specifica del μ-calcolo equazionale $(E \downarrow X)$, vale che*

$$t'\ ||_{L,M}\ t \models E \downarrow X \iff t'\ \models (E \downarrow X)/\!\!/_{L,M}\ t$$

A parole, Teorema 1 stabilisce che la specifica $(E \downarrow X)/\!\!/_{L,M}\ t$, ottenuta applicando l'operatore di quotienting rispetto a t, è soddisfatta da tutti e soli gli LTS t' che, se messi in parallelo con t, soddisfano anche $(E \downarrow X)$. Se è possibile mantenere sotto controllo la crescita della nuova asserzione principale , cioè quella ottenuta tramite quotienting, allora il risultato è una semplificazione del problema del MC.

L'operatore di quotienting è definito dalle regole in Tabella 4.1. Dall'alto verso il basso, il quotienting di una asserzione principale $(E \downarrow X)$ è dato da una nuova asserzione principale che ha come variabili di ingresso X_{s_0} (cioè una nuova variabile ottenuta etichettando X con il nome dello stato iniziale di t) e come equazioni quelle ottenute tramite quotienting di E. Il quotienting di un sistema vuoto di equazioni ϵ è sempre un sistema vuoto. Invece, il quotienting genera, per ogni equazione, un nuovo sistema di equazioni tali per cui le variabili sui lati sinistri sono ottenute tramite etichettatura della variabile (della equazione) originale con tutti gli stati di t e le asserzioni sui lati destri vengono prodotte

tramite quotienting dell'asserzione originale rispetto allo stato che etichetta la variabile al lato sinistro.

Abusando leggermente la notazione definiamo l'operatore $\cdot /\!\!/ \cdot$ anche per le asserzioni. Si noti che per questo nuovo operatore, la valutazione parziale viene fatta rispoetto ad uno specifico stato del LTS. Il quotienting per una singola variabile è X_s che dipende dallo stato dell'automa specializzato su uno stato s. Il quotienting di $\langle\alpha\rangle A /\!\!/_{L,M}\ s$ è il quotienting di due possibili componenti. Quello a sinistra considera l'azione α compiuta da t e verifico che almeno un passo da α ad s soddisfi la specifica. L'altra parte considera invece il caso in cui α è compiuta da un altro agente. Il quotienting di $\langle\tau\rangle A /\!\!/_{L,M}\ s$ è simile per le prime due parti, mentre la terza rappresenta il caso in cui nessuno degli agenti coinvolti compie τ, ma entrambi si sincronizzano su un canale β, cioè una qualunque azione in comune tra i due (che è l'intersezione tra quelle non ristrette $\overline{L}$ e quelle in M). In altre parole, si considera un canale in cui l'agente t si può sincronizzare e tutti gli altri agenti che possono sincronizzare su di esso. La regola per $(A_1 \vee A_2) /\!\!/_{L,M}\ s$ dice che il quotienting della disgiunzione tra due asserzioni, è la disgiunzione del quotienting delle due. Infine la costante F è invariante rispetto all'operatore di quotienting. Da queste possiamo derivare immediatamente le regole duali per $[\alpha]A$, $\wedge$ e T come segue.

$$[\alpha]A /\!\!/_{L,M}\ s = [\alpha](A /\!\!/_{L,M}\ s) \ \wedge \bigwedge_{s \xrightarrow{\alpha} s'} A /\!\!/_{L,M}\ s' \text{ se } \alpha \neq \tau$$

$$[\tau]A /\!\!/_{L,M}\ s = [\tau](A /\!\!/_{L,M}\ s) \ \wedge \bigwedge_{s \xrightarrow{\tau} s'} A /\!\!/_{L,M}\ s' \ \wedge \bigwedge_{\beta \in \overline{L} \cap M, s \xrightarrow{\beta} s'} [\overline{\beta}](A /\!\!/_{L,M}\ s')$$

$$(A_1 \wedge A_2) /\!\!/_{L,M}\ s = (A_1 /\!\!/_{L,M}\ s) \ \wedge \ (A_2 /\!\!/_{L,M}\ s)$$

$$T /\!\!/_{L,M}\ s = T$$

Esempio 10. *Si considerino nuovamente il sistema di Esempio 9 ed il LTS di Esempio 2. Assumendo* $L = \{join, eat, \overline{join}, \overline{eat}\}$ *e* $M = \{f_i, d_i, f_{i+1}, d_{i+1}, \bar{f}_i, \bar{d}_i, \bar{f}_{i+1}, \bar{d}_{i+1}\}$*, mostriamo l'applicazione dell'operatore di quotienting.*

$$E \downarrow X /\!\!/_{L,M}\ t_i = E /\!\!/_{L,M}\ t_i \downarrow X_a$$

dove $E /\!\!/_{L,M}\ t_i$ *è il sistema di equazioni ottenuto come*

$$\begin{cases} X_a & =_\mu ((([\overline{f}_i][\overline{f}_{i+1}][\tau]F \wedge [\overline{f}_{i+1}][\overline{f}_i][\tau]F) \wedge [\tau]X) /\!\!/_{L,M}\ a \\ X_{s_1} & =_\mu ((([\overline{f}_i][\overline{f}_{i+1}][\tau]F \wedge [\overline{f}_{i+1}][\overline{f}_i][\tau]F) \wedge [\tau]X) /\!\!/_{L,M}\ s_1 \\ X_{s_2} & =_\mu ((([\overline{f}_i][\overline{f}_{i+1}][\tau]F \wedge [\overline{f}_{i+1}][\overline{f}_i][\tau]F) \wedge [\tau]X) /\!\!/_{L,M}\ s_2 \\ X_m & =_\mu ((([\overline{f}_i][\overline{f}_{i+1}][\tau]F \wedge [\overline{f}_{i+1}][\overline{f}_i][\tau]F) \wedge [\tau]X) /\!\!/_{L,M}\ m \\ X_{r_1} & =_\mu ((([\overline{f}_i][\overline{f}_{i+1}][\tau]F \wedge [\overline{f}_{i+1}][\overline{f}_i][\tau]F) \wedge [\tau]X) /\!\!/_{L,M}\ r_1 \\ X_{r_2} & =_\mu ((([\overline{f}_i][\overline{f}_{i+1}][\tau]F \wedge [\overline{f}_{i+1}][\overline{f}_i][\tau]F) \wedge [\tau]X) /\!\!/_{L,M}\ r_2 \end{cases}$$

Applicando l'operatore di quotienting in Tabella 4.1, alla prima equazione otteniamo i seguenti passaggi.

$$[\overline{f}_i][\overline{f}_{i+1}][\tau]F /\!\!/_{L,M}\ a = [\overline{f}_i]([\overline{f}_{i+1}][\tau]F /\!\!/_{L,M}\ a) \wedge ([\overline{f}_{i+1}][\tau]F /\!\!/_{L,M}\ s_1)$$

Questa formula viene ulteriormente ridotta riapplicando le opportune regole di quotienting alle sotto-formule ancora da processare. Per brevità mostriamo solo il caso per $[\overline{f}_{i+1}][\tau]F/\!\!/_{L,M}\ a$ *che si riduce a*

$$[\overline{f}_{i+1}][\tau]F/\!\!/_{L,M}\ a \wedge [\tau]F/\!\!/_{L,M}\ s_2$$

Nuovamente, riapplicando le regole solo a $[\tau]F/\!\!/_{L,M}\ a$ *otteniamo*

$$[\tau](F/\!\!/_{L,M}\ a) \wedge [\tau](F/\!\!/_{L,M}\ a) \wedge [f_i](F/\!\!/_{L,M}\ s_1) \wedge [f_{i+1}](F/\!\!/_{L,M}\ s_2)$$

Anche senza mostrare interamente il risultato della valutazione parziale delle asserzioni del sistema, è facile osservare come l'operatore di quotienting causi una crescita significativa della dimensione della formula.

□

La dimensione di $E/\!\!/_{L,M}\ t$ è facilmente calcolabile come $\mid E \mid (\mid\!\rightarrow\mid + \mid S \mid)$. $\mid E \mid$ è il numero degli operatori, delle variabili e delle costanti in E, $\mid\!\rightarrow\mid$ è il numero delle transizioni un $\rightarrow$ ed $\mid S \mid$ è il numero di stati in S. Applicando ripetutamente l'operazione di quotienting in un sistema contenente n_i stati ed m_i transizioni, si ottiene un'asserzione risultante di dimensione non maggiore di $\mid E \mid \Pi_{i=1}^{N}(n_i + m_i)$. Questa espressione mostra che c'è il rischio di esplosione combinatoria della dimensione della formula, che si può evitare tramite opportune semplificazioni, rendendo efficace il quotienting ripetuto.

4.1.3 Semplificazioni

Come visto nella sezione precedente, il quotienting è un'operazione che generalmente causa una rapida crescita delle formule su cui è applicato. Per cercare di contrastare questo fenomeno è possibile applicare una processazione della formula per verificare la possibilità di applicare semplificazioni. Le semplificazioni hanno lo scopo di identificare *pattern* all'interno delle formule logiche e di sostituirli con formule equivalenti di dimensione minore, eventualmente portando all'eliminazione di parti di asserzioni o di intere equazioni. Le tecniche di semplificazione originariamente proposte in [1], sono riportate in Tabella 4.2.

A parole, la *Simple Evaluation* consiste nel ricercare la presenza di elementi neutri e assorbenti nelle congiunzioni (e nelle disgiunzioni). L'elemento neutro (es. T per la congiunzione) può essere rimosso, mentre l'elemento assorbente permette di collassare l'intera formula. Un ragionamento simile si applica anche agli operatori modali. Infatti le formule $\langle\alpha\rangle F$, $[\alpha]T$ sono banalmente equivalenti ad F e T rispettivamente.

La *Constant Propagation* consiste nel sostituire le istanze di una variabile con la costante F (T rispettivamente) nel caso in cui nel sistema sia presente un'equazione che semplicemente assegna tale costante alla variabile.

La *Unguardedness Removal* consiste ne sostituire all'interno delle asserzioni del sistema le istanze di una variabile con l'asserzione a cui la variabile è associata nel caso in cui l'istanza non ricada sotto operatori di modalità. Questa

operazione corrisponde all'*unfolding* del sistema di equazioni e ha lo scopo di favorire il riconoscimento di equazioni non raggiungibili (vedi *Reachability Analysis)*. Chiaramente questo processo può degenerare in un unfolding infinito nel caso siano presenti cicli. Per evitare questa evenienza i lati destri delle asserzioni coinvolte vengono processati per individuare la presenza di un sotto-sistema fortemente connesso. Se tale sotto-sistema esiste le asserzioni coinvolte vengono riscritte in modo da preservarne la semantica ma interrompendo il ciclo.

La *Trivial Equation Elimination*, si può facilmente osservare che equazioni del tipo $X =_\mu \langle\alpha\rangle X$ e $Y =_\nu [\alpha]Y$ sono equivalenti rispettivamente a $X =_\mu F$ e $Y =_\nu T$. Infatti la prima $X =_\mu \langle\alpha\rangle X$ ammette come soluzione il punto fisso della catena $F \vee \langle\alpha\rangle X$, $F \vee \langle\alpha\rangle(F \vee \langle\alpha\rangle X)$, ... Tale punto fisso può essere solo F. Il ragionamento analogo si applica nell'altro caso.

La *Equivalence Reduction* ha lo scopo di individuare all'interno del sistema E la presenza di equazioni ridondanti. Due equazioni sono ridondanti se accettano le stesse soluzioni nel sistema. Per esempio, se tale evenienza si verifica per due equazioni di minimo, una delle due può essere scritta come la disgiunzione dei lati destri delle equazioni originali, mentre la seconda può essere sostituita da un assegnamento della seconda variabile nella prima. Analogamente per le equazioni di massimo, una delle due può essere scritta come la congiunzione delle due originali, mentre l'altra come un'associazione della seconda variabile con la prima.

Un'ulteriore semplificazione consiste nell'effettuare una *Reachability Analysis* sul grafo associato al sistema di equazioni. Il grafo può essere costruito associando un nodo ad ogni variabile del sistema e una transizione direzionata tra due nodi se la variabile associata allo stato di destinazione appare nel lato destro dell'equazione della variabile associata allo stato di origine. Se non esiste un percorso dallo stato associato alla variabile di ingresso del sistema fino ad una variabile data, tale variabile risulta essere isolata dal sistema. Di conseguenza l'equazione ad essa associata può essere rimossa senza modificare le soluzioni del sistema.

4.2 Symbolic Partial Model Checking

Il *Symbolic Partial Model Checker* (nel seguito *SPMC*) è un'estensione del PMC che prevede l'applicazione (di una versione modificata) dell'operatore di quotienting a STS invece che a LTS. Lo scopo è di trarre vantaggio dalla rappresentazione simbolica degli stati di un sistema per ottenere formule specializzate di dimensioni ancora più compatte.

4.2.1 Symbolic Quotienting

Ridefiniamo l'operatore di quotienting di Sezione 4.1, modificandone le regole per permetterne l'applicazione a STS. La definizione del nuovo operatore di quotienting è data in Tabella 4.3.

Le regole in Tabella 4.3 seguono la struttura di quelle del quotienting in Tabella 4.1. In breve il nuovo operatore di quotienting si comporta come quello visto per il PMC (con l'unica differenza che ora t denota un STS e non un LTS) a meno delle regole per le asserzioni che contengono operatori modali. In particolare, la regola per $\langle\alpha\rangle A /\!\!/_{L,M}\ s$ stabilisce che la nuova formula consiste della disgiunzione di $\langle\alpha\rangle(A /\!\!/_{L,M}\ s)$ (come per il PMC) e la disgiunzione delle formule $(\gamma A /\!\!/_{L,M}\ s')$. In tali formule, appare un nuovo stato s' raggiungibile tramite una transizione $s \xrightarrow{z,\phi} s'$ e una sostituzione γ che è il *most general unifier* (mgu) del sistema di vincoli $\phi; [x = z]$. Intuitivamente tali elementi rappresentano i casi in cui il sistema t ammetta una transizione etichettata con il nome z. Se esiste una soluzione che, oltre a soddisfare i vincoli in ϕ, unifica i due nomi x e z, significa che il sistema può effettivamente compiere tale transizione. Chiaramente l'asserzione residua, oltre ad essere parzialmente valutata tramite quotienting, viene aggiornata applicando la sostituzione γ.

Infine, la regola per $\langle\tau\rangle A /\!\!/_{L,M}\ s$ genera un'asserzione che si costituisce della disgiunzione (i) di $\langle\tau\rangle(A /\!\!/_{L,M}\ s)$, ($ii$) delle formule $\gamma A /\!\!/_{L,M}\ s'$ e (iii) delle formule $\langle x\rangle(\gamma' A /\!\!/_{L,M}\ s')$. I tre elementi sono analoghi a quelli visti per il PMC. La differenza principale è data dalla presenza degli unificatori γ e γ' ottenuti come gli mgu dei vincoli che appaiono nelle transizioni coinvolte e degli opportuni vincoli di uguaglianza tra i nomi che etichettano tali transizioni.

Simple Evaluation

SE1	$X =_\sigma \bigwedge\{F, A_1, ..., A_k\}$	$\Rightarrow$	$X =_\sigma F$
SE2	$X =_\sigma \bigwedge\{T, A_1, ..., A_k\}$	$\Rightarrow$	$X =_\sigma \bigwedge\{A_1, ..., A_k\}$
SE3	$X =_\sigma \langle\alpha\rangle F$	$\Rightarrow$	$X =_\sigma F$

Constant Propagation

CP	$X =_\sigma A$		$X =_\sigma A[F/Y]$
	...	$\Rightarrow$	...
	$Y =_{\sigma'} F$		$Y =_{\sigma'} F$

Unguardedness Removal

UR	$X =_\sigma B$		$X =_\sigma B[A/Y]$
	...	$\Rightarrow$	...
	$Y =_{\sigma'} A$		$Y =_{\sigma'} A$

(Y senza guardie in B)

Trivial Equation Elimination

TEE$_\mu$	$X =_\mu \langle\alpha\rangle X$	$\Rightarrow$	$X =_\mu F$
TEE$_\nu$	$X =_\nu [\alpha] X$	$\Rightarrow$	$X =_\nu T$

Equivalence Reduction

ER$_\mu$	$X =_\mu A$		$X =_\mu A \vee B$
		$\Rightarrow$	
	$Y =_\mu B$		$Y =_\mu X$
ER$_\nu$	$X =_\nu A$		$X =_\nu A \wedge B$
		$\Rightarrow$	
	$Y =_\nu B$		$Y =_\nu X$

Tabella 4.2: Trasformazioni valide sulle asserzioni

$$(E \downarrow X)/\!\!/_{L,M}\ t = (E/\!\!/_{L,M}\ t \downarrow X_i)$$

$$(\epsilon)/\!\!/_{L,M}\ t = \epsilon$$

$$X =_\sigma A\ E/\!\!/_{L,M}\ t = \begin{cases} X_{s_1} =_\sigma A_1/\!\!/_{L,M}\ s_1 \\ \cdots \\ X_{s_n} =_\sigma A_n/\!\!/_{L,M}\ s_n \\ E/\!\!/_{L,M}\ t \end{cases}$$

$$X/\!\!/_{L,M}\ s = X_s$$

$$\langle x\rangle A/\!\!/_{L,M}\ s = \langle x\rangle(A/\!\!/_{L,M}\ s) \vee \bigvee_{s \xrightarrow{z,\phi} s'} (\gamma A/\!\!/_{L,M}\ s')$$
$$\text{dove } \gamma = \text{mgu}(\phi\,;\,[x = z])$$

$$\langle \tau\rangle A/\!\!/_{L,M}\ s = \langle \tau\rangle(A/\!\!/_{L,M}\ s) \vee \bigvee_{s \xrightarrow{\tau,\phi} s'} (\gamma A/\!\!/_{L,M}\ s') \vee \bigvee_{s \xrightarrow{z,\phi'} s'}^{x \in \bar{L} \cap M} \langle x\rangle(\gamma' A/\!\!/_{L,M}\ s')$$
$$\text{dove } \begin{array}{l} \gamma = \text{mgu}(\phi) \text{ e} \\ \gamma' = \text{mgu}(\phi'\,;\,[x = z]) \end{array}$$

$$A_1 \vee A_2/\!\!/_{L,M}\ s = A_1/\!\!/_{L,M}\ s \vee A_2/\!\!/_{L,M}\ s$$

$$F/\!\!/_{L,M}\ s = F$$

Tabella 4.3: Regole di quotienting simbolico

Capitolo 5

Prototipo

In questa sezione viene trattata l'implementazione del Framework nel dettaglio, iniziando con una panoramica generale e concludendo con l'analisi dettagliata dei singoli componenti sviluppati.

5.1 Architettura

Riassumendo i passaggi visti in precedenza, il nostro *tool* si comporta come segue. Dato un certo numero di STS in composizione parallela $(t_1, \ldots, t_N)$ contro la specifica da verificare $(E \downarrow X)$ si genera una nuova versione del problema del MC che è una specializzazione della politica in ingresso. Formalmente il nostro problema è dimostrare che

$$(t_1, \ldots, t_N) \models E \downarrow X$$

Quello che vogliamo è una specializzazione tale per cui

$$t_N \models E \downarrow X /\!\!/_{t_1} /\!\!/_{\ldots} /\!\!/_{t_{N-1}}$$

dove la parte a destra del simbolo $\models$ è stata ricavata dal SPMC con eventuali semplificazioni applicate. La parte destra e quella sinistra vengono ora fornite al MC come input da analizzare.

Per uno schema di massima, si veda la Figura 5.1.

5.2 Implementazione

In questa sezione vedremo nel dettaglio l'implementazione del Framework, seguendo lo schema di Figura 5.1 ed entreremo nel dettaglio dei singoli blocchi. Si parte definendo il problema che vogliamo analizzare, nel nostro caso abbiamo implementato il Working Example riportato in Capitolo 3. In particolare, i dati in ingresso (vedi Figura 5.1) sono STS degli agenti di tipo Filosofo (vedi Figura 3.2) e la formula in μ-calcolo equazionale è una rappresentazione della

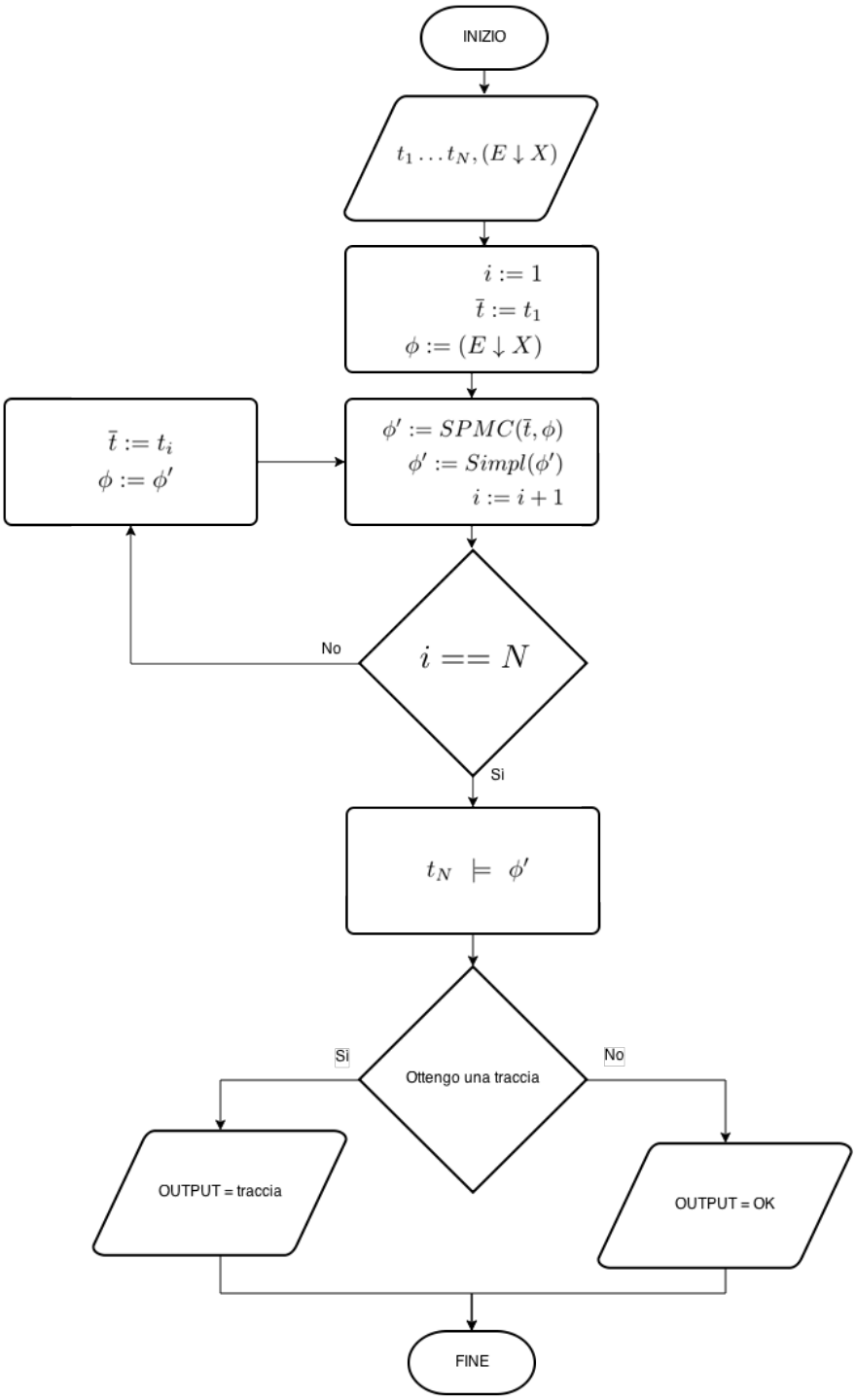


Figura 5.1: Schema generale dell'architettura del Framework

politica ivi descritta. Informalmente, vogliamo verificare che i Filosofi interagendo col Tavolo, non si trovino mai in una situazione di *starvation*. Per poter testare il comportamento del nostro Framework, diamo come input al nostro sistema un vettore di STS, in modo da poterne verificare le prestazioni ed il comportamento al crescere del numero degli agenti coinvolti. Per poter fornire un input decifrabile al nostro sistema, si sono convertiti gli agenti descritti dal π-calcolo in STS. Per far ciò, si è dovuta implementare una rappresentazione del linguaggio del π-calcolo che accogliesse i dati interpretati dal parser del π-calcolo (detto JavaCC, vedi Sezione. 5.3.1) ed inserirli in tale rappresentazione. Allo steso modo si è proceduto per interpretare le formule del μ-calcolo che descrivono la politica: le si è lette con un parser il quale le ha inserite in una struttura per consentirne la manipolazione e l'analisi. Le classi Java dedicate a questo compito sono state chiamate "JavaPi" e "JavaMu". Ciascuna di esse implementa una struttura per accogliere i dati, modellata sulle Definizioni 4 e 9, oltre che le azioni di invio e ricezione necessarie per permettere ai processi di comunicare (ad esempio, l'azione di *prendere una forchetta*). I processi e le azioni del μ-calcolo (e analogamente anche quelle del π-calcolo) sono stati implementati a partire da una classe astratta e da un'interfaccia che forniscono i metodi comuni per tutte le operazioni ricorrenti ed i tipi di dato disponibili. Ad esempio, per quel che riguarda il μ-calcolo , si forniscono strutture dati e metodi per determinare il contenuto del canale ed il tipo di argomento. A partire da questa classe astratta vengono poi implementate tutte le classi che modellano il μ-calcolo come descritto in Definizione 9 ed una serie di metodi comuni a tutte queste classi per effettuare operazioni come il confronto o la sostituzione dei nomi di variabile.

5.2.1 Symbolic Partial Model Checker

In riferimento alla Figura 5.1, il blocco che esegue il SPMC riceve in ingresso un vettore di STS ed una formula in μ-calcolo equazionale. Per prima cosa questi dati vengono passati ad una funzione che si occupa di ricavarne un file in linguaggio Promela, pronto per essere analizzato in SPIN. Il file Promela in uscita, conterrà quindi le specifiche di tutti gli agenti coinvolti e della politica, tradotte in un linguaggio comprensibile al Model Checker. A questo punto si procede con il SPMC vero e proprio. Per prima cosa viene chiamata una funzione che si occupa di fare il *quotienting* della politica, come visto in Definizione 4.2.1. La politica viene analizzata, un'equazione alla volta ed una nuova politica viene creata. Nel dettaglio, il cuore di questo sistema è la funzione "quotient(MuEquation E, Set<PiAction> coLcapM, STS t)" che prende in ingresso la formula in μ-calcolo equazionale della politica, un STS ed i nomi dei canali per ricavarne appunto il quotienting. Questa funzione viene chiamata per ogni equazione del sistema E in ingresso (distinguendo tra equazioni di punto fisso massimo e minimo) e per ogni equazione che compone ciascuna di queste, viene chiamata ricorsivamente una seconda funzione che riduce ogni componente, fino ad averli scansionati tutti . A questo punto la nuova politica può essere passata al modulo di semplificazione.

5.2.2 Semplificazioni

La politica appena ridotta dal SPMC viene sottoposta a tutte le semplificazioni viste nella Sezione 4.1.3. In particolare si cerca di semplificare la politica quanto più possibile. Una volta esaurite tutte le semplificazioni, se almeno una delle semplificazioni è andata a buon fine, le si ripete tutte ancora una volta fino a che nessuna delle semplificazioni può essere applicata. Questa scelta implementativa è stata fatta per cercare di ridurre al massimo la dimensione della politica. Le semplificazioni, pur se iterate, richiedono un tempo che cresce in modo lineare con le semplificazioni fatte. Considerando la natura esponenziale dell'aumento della complessità di calcolo, è conveniente effettuare quante più semplificazioni possibili. La classe Java astratta "Simplifier" è implementata in modo da poter aggiungere eventuali future semplificazioni con un intervento minimo. Se si volesse aggiungere una semplificazione, sarebbe sufficiente implementarla in una classe Java separata e poi crearne una nuova istanza nel metodo "getSimplifiers()". Una volta effettuate tutte le semplificazioni possibili, si procede con il secondo STS e così via. La nuova politica costruita con il SPMC della politica di partenza e dei primi $N-1$ STS viene passata al MC insieme al STS N-esimo.

Anche in questo caso si utilizza una classe astratta "Simplifier" che si occupa di implementare una semplificazione in *batch*, assicurandosi di semplificare quanto più possibile, come spiegato sopra.

5.2.3 Model Checking

Una volta che gli $(N-1)$ STS e la politica di partenza sono stati ridotti e semplificati, è il momento di passarli al Model Checker, insieme al STS N-esimo. Si chiama il tool SPIN tramite riga di comando e si misurano i tempi tra le varie operazioni da esso compiute, in modo da poter facilmente vedere i tempi effettivi impiegati. Viene anche osservato l'output in console per tenere traccia di eventuali errori avvenuti in fase di lettura della politica (un eventuale errore di traduzione) oppure durante una delle altre fasi. Altro dato importante che si vuole osservare è se SPIN individua o meno delle violazioni nella politica a lui passata. Una volta completata l'operazione, si possono confrontare le dimensioni delle politiche prima e dopo la semplificazione e di conseguenza i tempi impiegati da SPIN.

5.2.4 Input ed Output

Come già accennato in precedenza, per quel che riguarda gli input del nostro sistema, sono sufficienti una politica in μ-calcolo equazionale, ed una descrizione in π-calcolo di un agente.

Si può notare come la politica di *starvation* venga implementata in un sistema di equazioni di minimo punto fisso e come venga stabilita la variabile x come punto di ingresso (in Appendice A.2 si può vedere un dettaglio del pareser dedicato). Per quel che riguarda invece gli agenti, il Tavolo ed il Filosofo vengono implementati a partire dal π-calcolo , come riportato in Appendice A.1.

Per quel che riguarda il formato di uscita, il tool crea e salva i file Promela prima e dopo le semplificazioni. In tal modo gli esperimenti possono essere ripetuti, eventualmente su macchine più potenti. Inoltre ciò consente di poter effettuare ulteriori comparazioni in un secondo tempo, come ad esempio, dimensioni del file, della politica, profondità di risoluzione raggiunta da SPIN e così via. Questo ci consente anche di utilizzare un piccolo tool creato per facilitare le prime fasi della costruzione del Framework, che consente di trasformare agenti STS in formato "dot". Tale formato consente attraverso appositi strumenti di creare una rappresentazione grafica degli agenti, il tutto in maniera totalmente automatica.

5.3 Le tecnologie usate

Per questa tesi si è scelto di utilizzare *Java* come linguaggio di programmazione ed *Eclipse* come IDE (*Integrated Development Environment*. Queste scelte sono dovute alla flessibilità del linguaggio, che perfettamente si adatta alle nostre esigenze e alla grande disponibilità di *plugin* e librerie per *Eclipse*. Un altro importante punto a favore del linguaggio scelto è il fatto che *Java* è un linguaggio indipendente dalla piattaforma utilizzata. Questo significa che i *tool* prodotti sono compatibili con qualunque sistema operativo, rendendo il risultato finale fruibile da chiunque, senza vincoli *software*. Di riflesso, l'uso di *Eclipse* come IDE è stata una scelta abbastanza naturale: essendo scritto in *Java*, anche l'ambiente diventa svincolato da uno specifico sistema operativo. Questa caratteristica è stata determinante, nel momento in cui in uno stadio piuttosto avanzato del lavoro, ci si è trovati a dover utilizzare una libreria di terze parti (descritta nella sezione 5.3.3) e disponibile solo per sistema operativo *Linux*. La flessibilità del linguaggio e dell'IDE, ha permesso di continuare il lavoro su una macchina *Linux*, senza dover riscrivere nuovamente il codice prodotto fino a quel momento.

5.3.1 Java Compiler Compiler

Il *tool* prodotto ha la capacità di leggere espressioni in π-calcolo (vedi: sez. 2.1.4) e in μ-calcolo (vedi sez. 2.2.3) direttamente da *file* di testo. Per raggiungere questo risultato sono stati scritti due *parser* tramite un *plugin* di *Eclipse*: *Java Compiler Compiler* [14] (*JavaCC* nel seguito). JavaCC è un *software* che genera *parser* partendo da una grammatica definita dall'utente. Tramite i *parser* si popolano delle classi *Java* che rappresentano le strutture del π-calcolo e del μ-calcolo e che forniscono metodi per compiere operazioni su di esse.

5.3.2 Simple Promela Interpreter

Simple Promela Interpreter,noto come *Spin*, è un *tool* sviluppato a partire dal 1980 nei laboratori Dell da Gerard J. Holzmann e dal gruppo Unix del Computing Sciences Research Center e dal 1991 è *open source* ed è in continuo aggior-

namento. *Spin* è un software usato per la verifica formale della correttezza di modelli, in maniera rigorosa e in gran parte automatizzata. I sistemi da verificare devono essere descritti in *Process Meta Language*, comunemente abbreviato con *Promela*. Questo linguaggio consente di modellare algoritmi asincroni distribuiti ed automi non deterministici. Le proprietà che devono essere verificate sono espresse in LTL (vedi sez. 2.2.2), negate e quindi convertite in BA. Questo processo fa parte dell'algoritmo di MC. Oltre a questa caratteristica, Spin può agire come simulatore, seguendo i possibili percorsi di esecuzione e presentando all'utente una traccia di esecuzione. Al contrario di altri MC, Spin genera codice sorgente C per descrivere uno specifico problema. Grazie a questa tecnica si risparmia memoria e si migliorano le performance.

5.3.3 LTL2BA

Utilizziamo una libreria sviluppata da P. Gastin e descritta in [6]. Questa libreria consente la generazione di BA partendo da formule LTL ed usando un BA generalizzato come passo intermedio. Sostanzialmente, si trasforma una formula caratteristica (la politica)in un agente. Questo agente *caratteristico* è tale per cui se analizzato in *Sipn* produce una traccia, allora rispetta la politica, mentre se non la produce, la viola. Se si invertono le azioni di invio e ricezione dell'agente caratteristico, allora si ottiene l'agente che è in grado di verificare chi rispetta la traccia da chi non la rispetta.

Capitolo 6

Valutazione sperimentale

In questa Sezione riportiamo gli esperimenti effettuati per il nostro caso di studio, discusso nel Capitolo 3.

6.1 Struttura degli esperimenti

Per validare sperimentalmente il nostro approccio, abbiamo applicato il nostro prototipo agli agenti e alle specifiche mostrate in precedenza in questa tesi. In particolare abbiamo considerato due casi. Nel primo abbiamo considerato il problema di verificare tramite MC la composizione di k agenti t del tipo mostrato in Figura 3.2 contro una specifica E data in μ-calcolo equazionale. Al crescere di k abbiamo comparato il comportamento del model checker sul problema di verificare $t_1 \mid \ldots \mid t_k \mid \bar{t} \models E$ (dove $\bar{t}$ è l'agente in Figura 3.3) contro il comportamento dello stesso model checker sul problema $\bar{t} \models E /\!\!/_{L,M}\ t_1 \ldots /\!\!/_{L,M}\ t_k$. I risultati tengono conto del tempo impiegato nelle varie fasi del MC, del numero di stati visitati dal MC

6.2 Esperimenti e tempi

I Tabella 6.1 mostriamo i risultati degli esperimenti. In particolare la tabella si compone di due parti con i dati relativi al MC ed al MC applicato dopo il PMC. Le colonne per il MC mostrano la profondità raggiunta dal MC ed il tempo richiesto per la verifica (con time out impostato a cinque minuti di analisi). Si nota immediatamente che già con $k = 3$ il problema risulta intrattabile nel tempo dato. Inoltre al crescere di k il model checker raggiunge sempre minor profondità di visita entro la soglia temporale. Invece il partial model checking permette di mantenere costante la profondità di visita, dato che ad essere analizzato è sempre un solo agente alla volta. La dimensione della politica a cui viene applicato l'operatore di quotienting cresce rapidamente (colonna Dim. E, valori tra parentesi), ma tale crescita viene efficacemente contrastata dall'applicazione

	Senza PMC		Con PMC		
k	Profondità	Tempo (ms)	Dim E	Profondità	Tempo (ms)
1	528	841	19 (82)	43	1156
2	102327	2983	20 (138)	43	1091
3	>9999999	>300000	21 (145)	43	13953
4	553709	>300000	22 (152)	43	3546
5	56005	>300000	23 (159)	43	1179
6	16544	>300000	24 (166)	43	1057
7	8271	>300000	25 (173)	43	1200

Tabella 6.1: Caso di studio con 8 agenti di tipo Filosofo testardo

	Senza PMC		Con PMC		
k	Profondità	Tempo (ms)	Dim E	Profondità	Tempo (ms)
1	528	1038	19 (82)	43	1130
2	110498	4178	20 (100)	43	1056
3	5273069	>300000	21 (145)	43	7518
4	>9999999	>300000	22 (110)	43	4475
5	7320633	>300000	23 (159)	43	1091
6	50977	>300000	24 (120)	43	1145
7	5956	>300000	25 (173)	43	1196

Tabella 6.2: Caso di studio con agenti di tipi alternati

delle procedure di semplificazione (valori fuori dalle parentesi). Il risultato è una sostanziale riduzione dei tempi di verifica per la nuova istanza del problema.

I risultati in Tabella 6.2 sostanzialmente confermano i risultati ottenuti precedentemente. In particolare, possiamo osservare che le risorse computazionali necessarie per il MC dopo l'applicazione del quotienting sono sostanzialmente inferiori a quelle richieste quando si applica direttamente in MC.

Capitolo 7

Conclusioni

In questa tesi abbiamo mostrato il SPMC, un'estensione del PMC. Abbiamo mostrato come questa estensione porti un vantaggio nella rappresentazione compatta degli stati. Infine abbiamo implementato una piattaforma per il MC che trae vantaggio dal SPMC per cercare di ridurre il problema dell'esplosione combinatoria degli stati e abbiamo applicato il tutto ad un caso di studio, per verificarne i vantaggi e proporne possibili sviluppi.

7.1 Sviluppi futuri

L'attuale implementazione del SPMC può essere oggetto di ulteriori migliorie, soprattutto in termini di prestazioni. Con alcune modifiche al codice si potrebbe rendere il calcolo *multi core*, riducendo ulteriormente i tempi. Ciò non risolverebbe il problema dell'esplosione combinatoria degli stati, ma comunque ridurrebbe il calcolo, sfruttando a pieno la potenza dei moderni processori *multi core*. Non sarebbe corretto "spezzare" una singola esecuzione su più processi, in quanto si rischia di perdere alcune combinazioni di stati, però si potrebbero affidare due esecuzioni a due processi distinti. Le singole esecuzioni non sarebbero più veloci, ma in parallelo, il tempo totale dedicato al calcolo si potrebbe ridurre.

Un altra possibile miglioria potrebbe essere sviluppare una euristica per le semplificazioni proposte in [1]. Al momento le semplificazioni vengono eseguite in serie, ripetendole se almeno una semplificazione è stata effettuata. Questo perchè eseguendo una certa semplificazione, si potrebbe ricreare una situazione semplificata al passaggio precedente. Il problema è che con n semplificazioni, senza un algoritmo intelligente sviluppato *ad hoc*, ad ogni passo si devono ripetere tutte le semplificazioni precedenti, aumentando il numero di interazioni a $\frac{n}{2}(n+1)$ ed aumentando conseguentemente il tempo di calcolo. Si può analizzare se il tempo necessario per questa batteria di semplificazioni è poi recuperato in fase di MC grazie alle ulteriori semplificazioni ottenute. Con un algoritmo

intelligente che cerchi di ridurre al minimo queste semplificazioni si potrebbero ottenere vantaggi sia durante la semplificazione e che durante il SPMC.

Bibliografia

[1] Henrik Reif Andersen. Partial model checking (extended abstract). In *In Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 398–407. IEEE Computer Society Press, 1995.

[2] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[3] Michele Boreale and Rocco De Nicola. A symbolic semantics for the pi-calculus. *Inf. Comput.*, 126(1):34–52, 1996.

[4] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.

[5] E. Allen Emerson and Chin-Laung Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *Proceedings, Symposium on Logic in Computer Science, 16-18 June 1986, Cambridge, Massachusetts, USA*, pages 267–278, 1986.

[6] Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65, Paris, France, July 2001. Springer.

[7] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, August 1978.

[8] Dexter Kozen. Results on the propositional μ-calculus. In Mogens Nielsen and Erik Meineche Schmidt, editors, *ICALP*, volume 140 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 1982.

[9] Orna Lichtenstein and Amir Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Conference Record of the Twelfth Annual ACM Symposium on Principles of Programming Languages, New Orleans, Louisiana, USA, January 1985*, pages 97–107, 1985.

[10] Fabio Martinelli and Ilaria Matteucci. Modeling Security Automata with process algebras and related results, March 2006. Presented at the 6th International Workshop on Issues in the Theory of Security (WITS '06) - Informa.

[11] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[12] Robin Milner. The polyadic π-calculus: a tutorial. Technical report, Logic and Algebra of Specification, 1991.

[13] E. F. Moore. The shortest path through a maze. In *In Proceedings of the International Symposium on the Theory of Switching, Harvard University Press*, pages 285–192, 1959.

[14] Theo Norvell. Java compiler compliler. `http://www.engr.mun.ca/~theo/JavaCC-FAQ/javacc-faq-moz.htm#tth_sEc1.2`. [Web].

[15] Benjamin C. Pierce. Foundational calculi for programming languages. In *The Computer Science and Engineering Handbook*, pages 2190–2207. 1997.

[16] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

[17] Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000.

[18] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2007.

[19] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5(2):285–309, 1955.

Appendice A

Codice

A.1 Grammatica del π-calcolo

```
< PiParallel > : < PiSummation > (< PIPE > < PiSummation >)*
|       < PiSummation > : < PiActionRepeatedSequence >
        (< SUM > < PiActionRepeatedSequence >)*
|       < PiActionRepeatedSequence > : < PiActionSequence >
        (< RECURSION > < PiActionSequence >)*
|       < PiSequence > : (< PiRestriction >)*< PiAction >
        (< P >(< PiRestriction >)*< PiAction >)*
|   < PiAction > : < InAction > < PIPE > < OutAction >
        < PIPE > <SilentAction> < PIPE > (< PiParallel >)
|   < PiRestriction >  : < PiPrefix >
|       < PiPrefix > : < NEW > (< ID >)+
|       < OutAction > : < SEND ><ID >(< ID >)
|       < InAction > : < RECEIVE ><ID >(< ID >)
|       < SilentAction > : < T >

| < RECURSION : "rec" >
| < NEW : "nu">
| < T : "tau">
```

A.2 Grammatica del μ-calcolo

```
        <  parseOr > : < parseAnd > < OR > < parseOr > | parseAnds
|       < parseAnd > : < parseMax > < AND > < parseAnd > | parseMax
|       < parseMax > : < MAX > < ID > < P > < parseMin > | parseMin
|       < parseMin > : < ID > < ID > < P > < parseDiamond > | parseDiamond

|       < parseDia > : (< ODIA > < parseAction > < CDIA >)* < parseBox >
|       < parseBox > : (< OBOX > < parseAction > < CBOX >)* < parseAtom >
```

```
|       < OutAction > : < SEND >< ID >(< ID >)
|       < InAction > : < RECEIVE >< ID >(< ID >)
|       < SilentAction > : < T >

( |     < parseAction > : < InAction > | < OutAction > | < SilentAction >)

( |     < parseAtom > : < ID > | < OB > < parseOr > < CB > |
        < TRUE > | < FALSE >)
```

Ringraziamenti

Un gigantesco **grazie** a Gabriele per la pazienza e la disponibilità dimostrate durante la stesura di questa tesi e durante lo sviluppo del Framework. Un altrettanto gigantesco **grazie** ad Alessio e a Luca, per l'aiuto che mi hanno fornito lungo la strada. Menzione d'onore per Massimo, il quale ha aperto mondi che voi umani neanche vi potete immaginare (spin –h).

Grazie a Gabriele, Fabio, Giulio. Il laboratorio era un po' più triste (leggi silenzioso) senza di voi.

Grazie a Fabio ed Elisa. Davide lo ringrazio? Vabbè ok, lo ringrazio. Grazie Davide! Gente come voi (per fortuna?) è raro trovarla. Siete matti, simpatici ed allegri, al punto giusto. VVB a tutti e tre.

Grazie a Federico, Luca (sì, è sempre lo stesso: lo ringrazio due volte perchè è grosso), Andrea, Alen e Filippo. Con voi mi sento sempre a casa. Non importa quanto sia giù di morale o quanto la mia giornata sia stata brutta. Con voi c'è sempre un drink di benvenuto, una battutona, una risata, un dado fortunato oppure un aneddoto che mi rasserenano. I migliori amici che si possano desiderare. Punto.

Grazie a Luca, che ha condiviso il primo tratto di strada e grazie a Francesco, che ha condiviso con me l'ultimo. Sopravvissuti al traffico, ai sensi unici in contro mano ed ai cacciaviti. Eroi.

Un mercì ad Anastasia e Caroline. Oppure un graçias. Oppure... vabbè ho esaurito i vocaboli. Thank you. Danke. Grazie.

Un grazie ed un abbraccio ad Eros, Fabio, Roberta e Chiara. Pazienza, amicizia e risate. Siamo una bella squadra, eh?

Un meritatissimo grazie ad Alberto. Non importa quante volte si senta rispondere non posso. C'è sempre un altro invito per me. Un amico di vecchia data (ora non dico quanto vecchia che sennò ci freghiamo) a cui devo molto. Non c'è differenza che non si superi, non c'è discorso che non si possa fare. Grazie. Di cuore.

Uno speciale grazie a Francesca. Grazie per le marmotte, per i bradipi e tutti i restanti animali poco svegli del bestiario. Arrrrrrsssssssssenico! Vuoi una rosa? Scusa, mi sono distratto. Colpa degli occhi. Ma sto divagando. E' una mezza vita che ci conosciamo. Non cambierei neanche un minuto, nella storia della nostra amicizia. Doni sempre molto. E spesso non te ne rendi neppure conto. TVB bionda!

Planet House! Grazie! Claudia, Giulia e Silvia. Ornitorinco. 11 lettere, 53 punti. Ho vinto? Tipiacerebbe. 12 lettere, 234098 punti. Vuoi la rivincita? Un'altra volta, grazie. Che carine! Lo avete letto insieme!

Non vi ho dimenticati. Siete qui. E non perchè siete poco importanti, assolutamente. E' solo che sono le due e domani stampo quindi, beh... Grazie a Chiara, Nadia, Leonardo, Giulia, Serena, Alberto, Clarissa, Gianmarco, Matteo, Andrea, Daniele, Marco, Silvia, Claudia, Pamela.

Ringraziamenti random, ma doverosi. Google, Wikipedia e Stackoverflow (quando non trovi qualcosa con loro, sai che sei finito in un ginepraio). Grazie a Mr. Walker, Mr. Comfort, Mr. Mist & Mr. Daniels (avete sempre una parola di conforto ed il ginepraio si fa meno folto). Viva le magie di Latex, la documentazione del mio tool, le macchine virtuali e Dropbox. This is BibTeX, Version 0.99d (MiKTeX 2.9 64-bit)[...]A bad cross reference entry Kozen refers to entry conf/icalp/1982, which doesn't exist. F1. F1. F1. All vwork and no play maês jack a dull boy all work and nno pLay akes jack a dull boy all work and no play ma kes jack a dull boy.

Se vi sentite che mi sono dimenticato di voi, o di qualcuno che dovrebbe essere qui, ditemelo. Non è cattiveria, vi lovvo un sacco davvero, ma sono le due e venti ora tra una storia e l'altra e probabilmente sto scrivendo cose senza troppo senso a questo punto. Quindi se mi sono dimenticato qualcuno, dicevo, lo aggiungo a penna qui sotto ora, davvero, giuro:

VIP Lounge

Un grazie ed un abbraccio alle due persone alle quali devo tutto. Aldo e Gianna, vi voglio bene. La pazienza e l'incoraggiamento che mi manifestate ogni giorno mi hanno permesso di arrivare fin qui. GRAZIE!

Un pensiero a Teresa. Passa il tempo, ma il tuo sorriso gentile e la tua ironia sono con me ogni giorno. E so che i tuoi pazienti occhi azzurri vegliano sempre su di me. Ti voglio bene.