Introduzione al Model-Checking Simbolico

Silvio Ghilardi

5 Marzo 2007

Queste note costituiscono una introduzione molto sintetica alle tematiche del model-checking simbolico per sistemi a stati finiti. Se le informazioni contenute in questi appunti gli risultassero insufficienti o non abbastanza chiare, lo studente è invitato a prendere contatto col docente per farsi dare del materiale aggiuntivo. Indichiamo comunque come punto di riferimento il testo:

Clarke, Grumberg, Peled *Model Checking*, MIT Press, 2000.

Per un adeguato supporto software, lo studente è invitato a prendere un po' di confidenza con il sistema NuSMV:

http://nusmv.irst.itc.it/

ATTENZIONE: *questa versione della dispensa è preliminare, incompleta e tuttora in fase di elaborazione. Si invitano pertanto tutti i lettori a segnalare correzioni e miglioramenti da apportare. Verranno immesse in rete versioni aggiornate (controllate la data in alto per sapere se la versione in vostro possesso è l'ultima disponibile).*

Indice

1 Motivazioni **3**

2 Un esempio: la mutua esclusione secondo Peterson **4**

3 Il linguaggio e la semantica di CTL **8**
3.1 Sistemi di transizione 9
3.2 Operatori derivati 10
3.3 Model-Checking 12
3.4 Condizioni di fairness 12

4 Calcolo di punti fissi **13**
4.1 Esistenza di punti fissi per operatori monotoni 14
4.2 Gli operatori temporali come punti fissi 16

5 Macchine a stati finiti **18**
5.1 Model-Checking simbolico 21
5.2 Il ruolo degli OBDD 23

6 Verifica formale tramite un model-checker **24**

1 Motivazioni

Il model-checking è una tecnica per verificare formalmente proprietà relative al comportamento di *sistemi reattivi*. I sistemi reattivi sono sistemi che mantengono una continua interazione con l'ambiente; come tali, sono altamente non deterministici e non sono necessariamente predisposti per il calcolo di un output e nemmeno per raggiungere uno stato di terminazione. Tipici esempi di sistemi reattivi sono sono i sistemi per il controllo del traffico aereo o di dispositivi meccanici; più banalmente, rientrano fra i sistemi reattivi i programmi concorrenti o gli stessi sistemi operativi. Il paradigma della *concorrenza* è strettamente collegato ai sistemi reattivi: molti sistemi reattivi vengono modellati mediante processi che si svolgono in modo concorrente e anche nel caso di un singolo programma sequenziale che interagisce con l'ambiente è utile usare un modello concorrente a due attori formato dal programma stesso e dall'ambiente.

L'estrema complessità che i moderni sistemi concorrenti raggiungono rende sempre più insufficiente in sede di verifica le tradizionali tecniche di testing e di debugging e spinge nella direzione dello sviluppo di *metodi formali*, i soli in grado di esplorare il sistema mediante un esame *esaustivo* di tutti i suoi possibili comportamenti.

All'interno dei metodi formali, il model-checking si è imposto all'attenzione sia del mondo accademico che del mondo dell'industria per alcune sue caratteristiche peculiari e particolarmente attraenti. In primo luogo, il model-checking è una tecnica *completamente automatica* che non richiede interazioni complesse con l'utente, dopo la fase preliminare di modellizzazione del sistema. In secondo luogo, le tecniche di model-checking, qualora rilevino che il sistema non soddisfi le caratteristiche specificate, sono in grado di produrre *controesempi* consistenti in tracce di esecuzione che testimoniano il comportamento indesiderato.

In queste note, ci occuperemo solo del model-checking per *sistemi ad un numero finito di stati*, poichè per tali sistemi la ricerca e la conseguente sperimentazione hanno raggiunto un grado di assestamento tale da rendere l'argomento adatto ad un corso universitario istituzionale, ancorchè di secondo livello.

In sintesi, il model-checking prevede le seguenti fasi:

(a) *Modellizzazione*. In questa fase il sistema oggetto di studio viene descritto formalmente, utilizzando un formalismo accettato da un tool

per il model-checking. Tale formalismo assomiglia per certi versi ad un linguaggio di programmazione concorrente ad alto livello. Va osservato però che la modellizzazione comporta di per sè un certo grado di astrazione e di arbitrarietà: solo le caratteristiche che si ritengono rilevanti del sistema vengono inserite nel modello. In aggiunta, problemi di limitazione di tempo e di memoria possono richiedere gradi di astrazione piuttosto elevati, rispetto alle reali dimensioni del sistema che si modella.

(b) *Compilazione*. In questa fase, il model-checker compila automaticamente il file che gli viene sottoposto utilizzando strutture dati specifiche (quali ad esempio gli OBDD 'ordered binary decision diagrams', nel caso del model-checking simbolico). Il punto di riferimento per la compilazione è il modello computazionale adottato (sistemi di transizione, macchine a stati finiti, ecc.): sarà su tale modello computazionale che si innesterà la verifica formale vera e propria.

(c) *Specifica*. Qui l'utente deve specificare la proprietà del sistema che intende testare; la specifica si avvale del linguaggio formale di una logica temporale (usualmente LTL o CTL) atta ad esprimere proprietà del modello computazionale adottato.

(d) *Verifica*. Questa fase è totalmente automatica, il tool che si sceglie esegue semplicemente gli algoritmi per il model-checking che ha implementati. Se la verifica ha esito negativo, il tool deve essere in grado (almeno per le specifiche che rientrino in certe classi sintattiche) di fornire una traccia di esecuzione che rappresenti un controesempio possibile.

2 Un esempio: la mutua esclusione secondo Peterson

Per esemplificare il discorso in modo concreto, esaminiamo in dettaglio l'algoritmo di mutua esclusione secondo Peterson riportato sui manuali di sistemi operativi.

Nell'algoritmo di Peterson sono coinvolti due processi pr_0 e pr_1 che possono richiedere entrambi l'uso di una risorsa condivisa (ad esempio, il processore). Il sistema deve garantire ad entrambi l'accesso alla risorsa (previa

richiesta esplicita), ma non deve concederne l'uso simultaneo. Usiamo implicitamente un modello discreto del flusso temporale. Ad ogni istante, solo uno dei due processi è attivato; un processo attivato che non occupi già la risorsa può o meno richiedere di accedervi (e se richiede l'accesso, la richiesta si considera valida fino al suo soddisfacimento). Per semplificare leggermente le cose, assumiamo che un processo rilasci sempre spontaneamente la risorsa nell'istante di attivazione successivo a quello in cui l'ha ottenuta. Un processo pr_i $(i = 0, 1)$, nell'atto di richiedere la risorsa, segnala la sua richiesta ponendo la variabile booleana e_i uguale a 1; al contempo, pr_i pone uguale ad i un'altra variabile booleana che chiamiamo s (per 'signature'). La variabile s, a differenza della variabile e_i, può essere modificata anche dal processo rivale.

L'accesso alla risorsa ('sezione critica') è consentito a pr_i solo in due casi:

(i) e_{1-i} ha valore 0 (in tal caso, non c'è analoga richiesta conflittuale da parte del processo rivale);

(ii) s ha valore $1 - i$ (in tal caso, potrebbe esserci richiesta conflittuale, ma pr_i si è accodato per primo).

All'uscita della sezione critica, pr_i pone la variabile e_i uguale a 0, segnalando il rilascio della risorsa.

Ciò di cui vogliamo sincerarci è che l'algoritmo sopra descritto soddisfi ad alcune proprietà cruciali, quali le seguenti:

(a) non deve mai succedere che la risorsa sia occupata sia da pr_1 che da pr_2;

(b) ciascun processo deve poter ottenere la risorsa dopo ogni richiesta che fa;

(c) non ci devono essere sorpassi, cioè chi si accoda per primo riceve la risorsa per primo.

Le proprietà (b) e (c) hanno significato solo relativamente ad esecuzioni 'fair' dell'algoritmo: un'esecuzione è fair se ognuno dei due processi viene attivato infinite volte (se questa condizione non vale, non ha senso aspettarsi che valgano (b) e (c) perchè sarebbe possibile una disattivazione perpetua di un processo subito dopo una sua richiesta di entrare in regione critica).

Ci poniamo ora il problema di come verificare (a), (b), (c) e soprattutto di come una *procedura completamente automatica* possa verificare tali proprietà.

Oserviamo che uno stato del sistema può essere descritto completamente dalle 5 variabili booleane seguenti.

- e_0 segnala che pr_0 vuole entrare in sezione critica;
- e_1 segnala che pr_1 vuole entrare in sezione critica;
- c_0 segnala che pr_0 occupa la sezione critica;
- c_1 segnala che pr_1 occupa la sezione critica;
- s segnala il turno (cioè chi si è accodato per ultimo).

In conclusione, il sistema ha $2^5 = 32$ stati virtualmente possibili.

Costruiamo ora il grafo del sistema; tale grafo è il grafo diretto che ha per nodi gli stati e in cui lo stato σ è collegato allo stato σ' qualora σ' sia raggiungibile da σ mediante un singolo passo di esecuzione dell'algoritmo. Gli stati

$$\sigma_0 \ := \ (e_0 = e_1 = c_0 = c_1 = s = 0)$$

$$\sigma_1 \ := \ (e_0 = e_1 = c_0 = c_1 = 0, s = 1)$$

saranno marcati come stati iniziali del grafo del sistema (si noti che il valore di s in uno stato iniziale è considerato indifferente). Osserviamo che, ad esempio, da σ_0 si possono raggiungere in un passo di esecuzione i tre stati seguenti:

- σ_0 stesso (questo stato viene raggiunto quando il processo attivato, sia esso pr_0 o pr_1, non richiede la risorsa);
- $\sigma_3 := (e_1 = c_0 = c_1 = s = 0, e_0 = 1)$ (questo stato viene raggiunto quando il processo attivato è pr_0 e pr_0 richiede la risorsa);
- $\sigma_4 := (e_0 = c_0 = c_1 = 0, e_1 = s = 1)$ (questo stato viene raggiunto quando il processo attivato è pr_1 e pr_1 richiede la risorsa).

Siccome il grafo del sistema ha solo 32 stati, possiamo costruirlo tutto e verificare che la proprietà (a) vale in quanto nessuno stato con $c_0 = c_1 = 1$ è raggiungibile da σ_0 o σ_1. In realtà, gli stati raggiungibili da σ_0 e σ_1 in un numero finito di passi sono solo 10 e nessuno di essi ha la proprietà incriminata. In effetti, un model-checker non simbolico a grandi linee procederebbe

proprio così: costruirebbe ('on-the-fly', però!) il grafo del sistema e lo esplorerebbe stato per stato alla ricerca di uno stato raggiungibile dagli stati iniziali che soddisfi $c_0 = 1 \wedge c_1 = 1$.

Questo approccio va incontro tuttavia a vari problemi. Un primo problema (detto problema di *esplosione degli stati*) sta nel fatto che spesso il grafo del sistema è troppo grosso per poter essere esplorato tutto. Sono state proposte varie soluzioni al problema dell'esplosione degli stati; in queste note prenderemo la via dell'*approccio simbolico*.

Prima di analizzare (b) e (c), concentriamoci sulla specifica formale della proprietà (a). Per poterla esprimere non bastano gli operatori booleani $\neg$ (not), $\wedge$ (and), $\vee$ (or)[1] della logica classica, occorre un operatore temporale. Nel seguito per 'esecuzione' (o 'run') del sistema intenderemo sempre un cammino infinito

$$\tau_0 \to \tau_1 \cdots \to \tau_n \to \cdots \tag{1}$$

dentro al grafo del sistema. Se p è una proprietà degli stati, con $\mathbf{AG}\, p$ indichiamo l'insieme degli stati τ_0 per cui ogni run (1) che inizia con τ_0 è interamente composto da stati che godono della proprietà p. Con questo operatore a disposizione, possiamo esprimere (a) dicendo che la formula

$$\mathbf{AG}\, \neg(c_0 = 1 \wedge c_1 = 1) \tag{2}$$

deve essere vera negli stati iniziali σ_0, σ_1. La formula (2) è un esempio di formula della logica temporale CTL ("Computational Tree Logic"); CTL sarà il punto di riferimento di queste note, ma segnaliamo che esistono altre opzioni. Ad esempio, NuSMV supporta anche specifiche nella logica temporale LTL;[2] le logiche LTL e CTL sono in generale incomparabili come potere espressivo, tuttavia la formula (2) poteva essere ugualmente espressa in LTL.[3]

Passiamo ora all'analisi delle proprietà (b) e (c), che sono più complesse. Per (b) abbiamo bisogno di un ulteriore operatore temporale, che chiamiamo $\mathbf{AF}$; data una proprietà p, la formula $\mathbf{AF}\, p$ sarà vera in uno stato τ_0 qualora ogni run (1) che inizia con τ_0 contenga almeno uno stato che gode della

[1] In NuSMV tali operatori sono indicati rispettivamente con $!, \&, |$.

[2] Per inserirle, usare la parola chiave `LTLSPEC` invece di `SPEC`.

[3] La formula di LTL corrispondende è semplicemente $\mathbf{G}\, \neg(c_0 = 1 \wedge c_1 = 1)$. Per informazioni su LTL, si consultino le note relative alla logica temporale lineare (si osservi però che in tali note gli operatori $\mathbf{G}, \mathbf{F}$ sono indicati rispettivamente con $\Box, \Diamond$).

proprietà p. Per esprimere (b), chiediamo che entrambe le formule seguenti[4]

$$\mathbf{AG}\,(e_0 \to \mathbf{AF}\,c_0) \qquad \mathbf{AG}\,(e_1 \to \mathbf{AF}\,c_1) \tag{3}$$

siano vere negli stati iniziali σ_0, σ_1 (qui ovviamente $\to$ denota l'implicazione booleana 'if ... then'). Una volta scritta (b) in questo modo, possiamo darla in pasto ad un model-checker (insieme ad una descrizione formale precisa del sistema): il model-checker ci dirà che in effetti la (b) è vera (se valgono le condizioni di fairness relative all'attivazione indefinita dei due processi - tali condizioni di fairness vanno anch'esse comunicate al model-checker). In questo caso, non è così ovvio immaginare come il model-checker possa aver proceduto, perchè se è vero che gli stati del sistema sono finiti, è altrettanto vero che i run finiti non sono di certo.

Resta da formalizzare la (c); ci serve un ulteriore operatore temporale di CTL. Se p, q sono proprietà degli stati con $\mathbf{A}\,[p\,\mathbf{U}\,q]$ indichiamo la la proprietà goduta dagli stati τ_0 tali che in ogni run (1) la proprietà p vale finchè non si verifica la proprietà q. Possiamo ora esprimere la condizione (c) di rispetto del turno chiedendo che le formule seguenti[5]

$$\mathbf{AG}\,((e_0 \wedge \neg e_1) \to \mathbf{A}\,[(\neg c_1)\,\mathbf{U}\,c_0]) \qquad \mathbf{AG}\,((e_1 \wedge \neg e_0) \to \mathbf{A}\,[(\neg c_0)\,\mathbf{U}\,c_1]) \tag{4}$$

siano entrambe vere negli stati iniziali. Di nuovo la specifica è vera; nel paragrafo 6 la testeremo davvero mediante un model-checker come NuSMV.

3 Il linguaggio e la semantica di CTL

Presentiamo in questo paragrafo la sintassi e la semantica di CTL. Per semplicità, introduciamo il linguaggio usando solo i tre seguenti operatori temporali (definiremo più avanti i rimanenti in termini di questi):

- $\mathbf{EG}\,\phi$: "c'è un'esecuzione in cui ϕ sarà sempre vera";
- $\mathbf{E}\,[\phi\,\mathbf{U}\,\psi]$: "c'è un'esecuzione in cui ϕ sarà vera finchè non si verifica ψ;
- $\mathbf{EX}\,\phi$: "c'è un'esecuzione in cui ϕ sarà vera nel prossimo istante."

[4] Anche in questo caso si possono usare specifiche LTL, ad esempio $\mathbf{G}\,(e_i \to \mathbf{F}\,c_i)$, per $i = 1, 2$.

[5] Una corrispondente specifica LTL è $\mathbf{G}\,((e_i \wedge \neg e_{1-i}) \to ((\neg c_{1-i})\,\mathbf{U}\,c_i))$, per $i = 1, 2$.

Formalmente, dato un insieme Var di variabili proposizionali, l'insieme delle Var-formule (o semplicemente formule) di CTL è così definito:

- ogni $p \in Var$ è una formula:

- se ϕ, ψ sono formule, tali sono $(\neg\phi), (\phi \wedge \psi), (\psi \vee \phi)$;

- se ϕ, ψ sono formule, tali sono $(\mathbf{EG}\,\phi), (\mathbf{EX}\,\phi), \mathbf{E}\,\mathbf{U}\,(\phi, \psi)$.

La formula $\mathbf{E}\,\mathbf{U}\,(\phi, \psi)$ sarà usualmente scritta con $\mathbf{E}\,[\phi\,\mathbf{U}\,\psi]$. Le parentesi più esterne sono di regola omesse e si conviene che gli operatori unari abbiano precedenza maggiore di quelli binari. Valgono le abbreviazioni usuali per $\top, \bot, \phi \rightarrow \psi, \phi \leftrightarrow \psi$: tali formule sono definite rispettivamente come $p \vee \neg p$, $\neg\top$, $\neg\phi \vee \psi$, $(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$.

3.1 Sistemi di transizione

Per introdurre la semantica di CTL, ricordiamo la definizione di sistema di transizione e di esecuzione ('run') in un sistema di transizione. Fissiamo un insieme Var di variabili proposizionali; un *sistema di transizione* etichettato in Var è una quadrupla

$$T \;=\; (W, W_0, R, v)$$

dove W è un insieme finito (detto insieme degli stati del sistema), W_0 è un sottoinsieme non vuoto di W (detto insieme degli stati iniziali),[6] $R \subseteq W \times W$ è una relazione su W e v è una funzione che assegna ad ogni stato un assegnamento booleano[7] su Var (che dice quali sono le variabili proposizionali 'vere' in tale stato).

Per evitare che le definizioni che daremo possano avere un significato non voluto, si assume che la relazione R soddisfi la condizione (detta di *serialità*)

$$\forall w \in W\, \exists v \in W\, R(w, v).$$

La condizione di serialità significa che il sistema non può mai trovarsi in una situazione di stallo; si noti che la serialità può essere sempre banalmente

[6] La definizione che diamo qui differisce leggermente da quella data nella dispensa su LTL, dove si chiedeva che ci fosse un solo stato iniziale.

[7] Un assegnamento booleano su Var è una funzione da Var verso l'insieme $\{0, 1\}$ (quindi, per ogni s, si ha che $v(s)$ è una funzione che associa ad ogni $p \in Var$ il valore 0 o il valore 1).

forzata aggiungendo al sistema uno stato di 'errore' da cui il sistema stesso, una volta entrato, non può più uscire.

Se $s \in W$, una *s-esecuzione* σ su T è una successione infinita

$$\sigma = s_0, s_1, s_2, \ldots \tag{5}$$

di elementi di W tale che $s_0 = s$ e inoltre vale $R(s_i, s_{i+1})$ per ogni $i \geq 0$. Un'*esecuzione* tout-court è una s-esecuzione, in cui $s \in W_0$ (cioè in cui s è uno stato iniziale).

Definizione 3.1 *Se ϕ è una formula, $T = (W, w_0, R, v)$ è un sistema di transizione e $s \in W$ è uno stato di T, la nozione di* ***verità di*** *ϕ* ***in*** *T* ***nello stato*** *s (scritta con $T \models_s \phi$) è così definita:*[8]

- $T \models_s p$ *sse* $v(s)(p) = 1$;
- $T \models_s \neg\phi$ *sse* $T \not\models_s \phi$;
- $T \models_s \phi \wedge \psi$ *sse* ($T \models_s \phi$ *e* $T \models_s \psi$);
- $T \models_s \phi \vee \psi$ *sse* ($T \models_s \phi$ *o* $T \models_s \psi$);
- $T \models_s \mathbf{EX}\,\phi$ *sse esiste* t *tale che* $R(s,t)$ *e* $T \models_t \phi$;
- $T \models_s \mathbf{EG}\,\psi$ *sse esiste una s-esecuzione* (5) *tale che* $T \models_{s_i} \psi$ *vale per ogni* $i \geq 0$;
- $T \models_t \mathbf{E}\,[\phi\,\mathbf{U}\,\psi]$ *sse esiste una s-esecuzione* (5) *tale che per qualche* $i \geq 0$ *vale che* $T \models_{s_i} \psi$ *e che* $T \models_{s_j} \phi$ *per ogni* $j < i$.

Quando scriviamo $T \models \phi$ (letta con: 'ϕ è vera in T, o con 'T **soddisfa la specifica** ϕ') intendiamo dire che ϕ vale in tutti gli istanti iniziali, cioè che vale $T \models_s \phi$ per ogni $s \in W_0$.

3.2 Operatori derivati

Ci sono vari altri operatori temporali che si possono esprimere in CTL combinando quelli che abbiamo già introdotto; ne elenchiamo alcuni qui, evidenziandone anche la corrispondente condizione semantica (il lettore li tenga tutti presenti quando deve inserire le specifiche in sistemi come NuSMV).

[8] Qui e altrove 'sse' abbrevia 'se e solo se'.

- La formula $\mathbf{AX}\,\phi$ è definita come $\neg\mathbf{EX}\,\neg\phi$: abbiamo che vale $T \models_s \mathbf{AX}\,\phi$ sse per ogni t tale che $R(s,t)$ si ha $T \models_t \phi$.

- La formula $\mathbf{EF}\,\phi$ è definita come $\mathbf{E}\,[\top\,\mathbf{U}\,\phi]$: abbiamo che vale $T \models_s \mathbf{EF}\,\phi$ sse esiste una s-esecuzione (5) tale che $T \models_{s_i} \phi$ vale per un $i \geq 0$.

- La formula $\mathbf{AG}\,\phi$ è definita come $\neg\mathbf{EF}\,\neg\phi$: abbiamo che vale $T \models_s \mathbf{AG}\,\phi$ sse per ogni s-esecuzione (5) si ha che $T \models_{s_i} \psi$ vale per ogni $i \geq 0$.

- La formula $\mathbf{AF}\,\phi$ è definita come $\neg\mathbf{EG}\,\neg\phi$: abbiamo che vale $T \models_s \mathbf{AF}\,\phi$ sse per ogni s-esecuzione (5) esiste un $i \geq 0$ tale che vale $T \models_{s_i} \psi$.

- La formula $\mathbf{A}\,[\phi\,\mathbf{U}\,\psi]$ è definita come $\neg\mathbf{E}\,[\neg\psi\,\mathbf{U}\,(\neg\phi \wedge \neg\psi)] \wedge \neg\mathbf{EG}\,\neg\psi$: abbiamo che vale $T \models_s \mathbf{A}\,[\phi\,\mathbf{U}\,\psi]$ sse per ogni s-esecuzione (5) esiste un $i \geq 0$ per cui vale che $T \models_{s_i} \psi$ e che $T \models_{s_j} \phi$ per ogni $j < i$.

Il lettore, per ricordarsi le abbreviazioni, tenga presesenti che $\mathbf{A}$ e $\mathbf{E}$ (i cosiddetti 'path-quantifiers') quantificano universalmente ed esistenzialmente su s-esecuzioni; la caratteristica di CTL sta nel fatto che essi *devono essere immediatamente seguiti dagli operatori* $\mathbf{X}\,,\mathbf{F}\,,\mathbf{G}\,,\,\mathbf{U}$ che sono tipici di LTL.[9] Le combinazioni possibili sono dunque le 8 seguenti

$$\mathbf{AX}\,,\ \mathbf{EX}\,,\ \mathbf{AF}\,,\ \mathbf{EF}\,,\ \mathbf{AG}\,,\ \mathbf{EG}\,,\ \mathbf{AU}\,,\ \mathbf{EU}$$

(si noti che le abbiamo prese tutte in considerazione). Osserviamo infine che, data la condizione di serialità, le condizioni di verità per $\mathbf{AX}\,,\mathbf{EX}$ possono essere equivalentemente formulate con:

- $T \models_s \mathbf{AX}\,\phi$ sse per ogni s-esecuzione (5) si ha che $T \models_{s_1} \phi$;

- $T \models_s \mathbf{EX}\,\phi$ sse esiste una s-esecuzione (5) tale che $T \models_{s_1} \phi$.

[9] Per fare gli opportuni raffronti con la dispensa su LTL, ricordiamo che quando per il model-checking si interpretano le formule di LTL su sistemi di transizione arbitrari (e non sul flusso di tempo lineare discreto dei numeri naturali), esse vengono sempre fatte implicitamente precedere da un *singolo quantificatore universale* $\mathbf{A}$.

3.3 Model-Checking

Il problema del **model checking** è il seguente: *data una formula ϕ e dato un sistema di transizione a stati finiti T, stabilire se T soddisfa la specifica ϕ.*

Esistono due metodi per trattare il problema suddetto, il metodo *esplicito* e il metodo *simbolico*; in queste note ci concentreremo sul metodo simbolico, prima però diamo qualche informazione rilevante anche sul metodo esplicito, cominciando da una stima di complessità:

Teorema 3.2 *La relazione $T \models \phi$ è decidibile in tempo $O((|W| + |R|) \cdot |\phi|)$ (qui $|W|, |R|$ indicano la cardinalità di W, R e $|\phi|$ indica la lunghezza di ϕ).*

Una via per dimostrare questo risultato consiste nel passare attraverso opportuni automi e nello studiare poi il problema della loro vuotezza: questa è una via che abbiamo già praticato per LTL (nel caso di CTL occorre però considerare automi più complessi che lavorano su alberi anzichè su parole infinite). Un'altra via per dimostrare il Teorem 3.2 consiste nell'affrontare il problema in modo diretto, etichettando gli stati di W ricorsivamente con le sottoformule di ϕ che essi soddisfano: la procedura è relativamente semplice, ma nel caso di **EG** occorre sfruttare l'algoritmo lineare di Tarjan per la decomposizione in componenti connesse forti (dette 'scc') di un grafo.

3.4 Condizioni di fairness

Nell'analizzare l'algoritmo di mutua esclusione di Peterson, abbiamo incontrato già problemi di fairness: in generale non ha senso considerare tutte le possibili s-esecuzioni (5), ma occorre limitarsi ad alcune di esse (ad esempio, se vari processi sono coinvolti nel nostro sistema, solo le esecuzioni in cui tutti i processi vengono a turno attivati dallo scheduler - che si suppone non iniquo - avvengono realmente). Come possiamo esprimere nel quadro astratto dei nostri sistemi di transizione tali vincoli di fairness? Un vincolo di fairness dice che dobbiamo limitare le esecuzioni realistiche alle esecuzioni in cui certe condizioni (es. l'attivazione di un processo) si verificano infinite volte: sarà dunque sufficiente indicare esplicitamente tali condizioni (che saranno formule nell'approccio simbolico e semplicemente insiemi di stati nel nostro caso generale).

Un *vincolo di fairness* in un sistema di transizione $T = (W, w_0, R, v)$ è un insieme finito

$$F \;=\; \{S_1, \ldots, S_k\}$$

di insiemi di stati. Una s-esecuzione (5) è *fair* rispetto a tale F sse per ogni S_j $(j = 1, \ldots, k)$ esistono infiniti i tali che $s_i \in S_j$.

La nozione di "verità di ϕ in T nello stato s rispetto a F" (scritta con $T \models^F_s \phi$) è così definita:

- $T \models^F_s p$ sse $v(s)(p) = 1$;
- $T \models^F_s \neg\phi$ sse $T \not\models^F_s \phi$;
- $T \models^F_s \phi \wedge \psi$ sse ($T \models^F_s \phi$ e $T \models^F_s \psi$);
- $T \models^F_s \phi \vee \psi$ sse ($T \models^F_s \phi$ o $T \models^F_s \psi$);
- $T \models^F_s \mathbf{EX}\,\phi$ sse esiste una s-esecuzione fair (5) tale che $T \models^F_{s_1} \phi$;
- $T \models^F_s \mathbf{EG}\,\psi$ sse esiste una s-esecuzione fair (5) tale che $T \models^F_{s_i} \psi$ vale per ogni $i \geq 0$;
- $T \models^F_t \mathbf{E}\,[\phi\,\mathbf{U}\,\psi]$ sse esiste una s-esecuzione fair (5) tale che per qualche $i \geq 0$ vale che $T \models^F_{s_i} \psi$ e che $T \models^F_{s_j} \phi$ per ogni $j < i$;

(si noti che per **EX** abbiamo dovuto usare la condizione che fa riferimento alle esecuzioni perchè la serialità non è sufficiente a garantire che si possa prolungare un singolo arco $R(s, s_1)$ ad una intera s-esecuzione che sia fair).

La nozione $T \models^F \phi$ è ancora definita come '$T \models^F_s \phi$ per ogni $s \in W_0$' e il Teorema 3.2 si può raffinare nel seguente:

Teorema 3.3 *La relazione $T \models^F \phi$ è decidibile in tempo $O((|W| + |R|) \cdot |\phi| \cdot |F|)$ (qui $|F|$ indica il numero k di elementi di F).*

4 Calcolo di punti fissi

In questo paragrafo esponiamo alcuni elementi essenziali della teoria dei punti fissi per operatori monotoni. In seguito dimostriamo che gli operatori temporali di CTL sono tutti riconducibili a punti fissi (più avanti vedremo come utilizzare per i nostri scopi la simulazione simbolica del calcolo per approssimazione di tali punti fissi).

Se W è un insieme, con $\wp(W)$ indichiamo l'insieme di tutti i sottoinsiemi di W; per *operatore* intenderemo sempre una funzione del tipo[10]

$$O : \wp(W) \longrightarrow \wp(W);$$

un tale operatore O è detto **monotono** qualora per ogni $S_1, S_2 \in \wp(W)$ valga che

$$S_1 \subseteq S_2 \quad \Rightarrow \quad O(S_1) \subseteq O(S_2).$$

Un **punto fisso** dell'operatore $O : \wp(W) \longrightarrow \wp(W)$ è un $F \subseteq W$ tale che $F = O(F)$. Un punto fisso F di O è detto essere il **minimo punto fisso** di O se è un punto fisso di O e vale $F \subseteq F'$ per ogni altro punto fisso F' di O; il minimo punto fisso di O può non esistere, ma se esiste è unico e viene indicato con $\mu_X O(X)$. Similmente, un punto fisso F di O è detto essere il **massimo punto fisso** di O se è un punto fisso di O e vale $F' \subseteq F$ per ogni altro punto fisso F' di O; di nuovo, il massimo punto fisso di O può non esistere, ma se esiste è unico e viene indicato con $\nu_X O(X)$.

4.1 Esistenza di punti fissi per operatori monotoni

Un risultato fondamentale (noto come teorema di Knaster-Tarski) afferma l'esistenza di massimi e minimi punti fissi per gli operatori monotoni:

Teorema 4.1 *Ogni operatore monotono ammette massimo e minimo punto fisso.*

Dim. Diamo la dimostrazione per il caso del massimo punto fisso (l'altra è duale). Sia $O : \wp(W) \longrightarrow \wp(W)$ monotono; definiamo F come l'unione di tutti gli $S \subseteq W$ tali che $S \subseteq O(S)$, in simboli:

$$F \ := \ \bigcup \{S \subseteq W \mid S \subseteq O(S)\}.$$

Proviamo che F è il massimo punto fisso di O; per far questo occorre verificare le tre seguenti condizioni:

(i) $F \subseteq O(F)$;

(ii) $O(F) \subseteq F$;

[10] In realtà, in tutti i risultati che illustreremo, si può sostituire $\wp(W)$ con quello che in algebra si chiama un 'reticolo completo', ma non avremo bisogno di tale generalizzazione.

(iii) se F' è punto fisso di O, allora $F' \subseteq F$.

Dimostrazione di (i): basterà provare che per ogni $S \subseteq W$ tale che $S \subseteq O(S)$, si ha $S \subseteq O(F)$ (infatti, F è l'unione di questi S e se tutti costoro sono contenuti in $O(F)$, anche F è contenuto in $O(F)$). Ma se $S \subseteq W$ è tale che $S \subseteq O(S)$, tale S è contenuto in F (perchè fa parte della famiglia di cui F è l'unione), per cui abbiamo $O(S) \subseteq O(F)$ per la monotonia di O e infine proprio $S \subseteq O(F)$ per la transitività della relazione di sottoinsieme applicata alle inclusioni $S \subseteq O(S) \subseteq O(F)$ appena stabilite.

Dimostrazione di (ii): siccome F è definito come l'unione di tutti gli $S \subseteq W$ tali che $S \subseteq O(S)$, basterà provare che anche $O(F)$ appartiene a tale famiglia, ossia che $O(F) \subseteq O(O(F))$: questo fatto è immediato per (i) che abbiamo già dimostrato e per la monotonia di O.

Dimostrazione di (iii): di nuovo, siccome F è definito come l'unione di tutti gli $S \subseteq W$ tali che $S \subseteq O(S)$, basterà provare che anche F' appartiene a tale famiglia, ossia che $F' \subseteq O(F')$: questa inclusione segue immediatamente dal fatto che F' è un punto fisso di O. $\dashv$

La dimostrazione del Teorema 4.1 che abbiamo riportato per completezza è molto elegante, ma non è molto utile per approssimare e calcolare i punti fissi, scopo per cui occorre un'altra idea. Consideriamo i sottoinsiemi:

$$W,\ O(W),\ O(O(W)),\ \ldots,$$

Questi sottoinsiemi (che chiamiamo $N_0, N_1, \ldots$) sono definiti ricorsivamente nel modo seguente:

$$N_0 \ := \ W, \quad N_{i+1} \ := \ O(N_i). \tag{6}$$

Per induzione, è immediato verificare che essi formano una successione decrescente $N_0 \supseteq N_1 \supseteq N_2 \supseteq \cdots$: in effetti, $N_0 \supseteq N_1$ vale perchè $N_0 = W$ è il sottoinsieme più grande di tutti e $N_{i+1} \supseteq N_{i+2}$ segue dall'ipotesi induttiva $N_i \supseteq N_{i+1}$ per monotonia. Se ora W è **finito** (cosa che a noi accadrà sempre) è chiaro che la successione (6) diventerà per forza costante da un certo punto in poi, cioè avremo $N_j = N_{j+1}$ per qualche j.

Verifichiamo che **tale N_j è il massimo punto fisso di O.**[11] Siccome $N_j = N_{j+1} = O(N_j)$, abbiamo che N_j è davvero un punto fisso. Sia F' un

[11] Si può ottenere lo stesso risultato anche nel caso non finito, ma occorre iterare la costruzione sugli ordinali transfiniti.

altro punto fisso: per induzione è immediato verificare che $F' \subseteq N_i$ per ogni i (infatti per $i = 0$, N_0 è il più grande sottoinsieme di W e per $i > 0$ basta applicare la monotonia e la definizione di punto fisso all'ipotesi induttiva $F' \subseteq N_{i-1}$).

Per il **calcolo del minimo punto fisso**, si usa un procedimento analogo: basta proseguire finchè non si stabilizza la successione crescente

$$\emptyset,\ O(\emptyset),\ O(O(\emptyset)),\ \ldots,$$

definita ricorsivamente nel modo seguente:

$$M_0\ :=\ \emptyset, \quad M_{i+1}\ :=\ O(M_i). \tag{7}$$

4.2 Gli operatori temporali come punti fissi

Riprendiamo ora il discorso sulla semantica di CTL. Dati un sistema di transizione $T = (W, w_0, R, v)$ e una formula ϕ, abbiamo definito nel paragrafo 3.1 la relazione $T \models_s \phi$ per ogni stato s; per molte questioni è utile identificare una formula con l'insieme degli stati in cui è vera. Formalmente, definiamo

$$[\![\phi]\!]_T\ :=\ \{s \in W \mid T \models_s \phi\}.$$

In questo modo possiamo associare ai connettivi delle operazioni insiemistiche: ad esempio, la congiunzione, la disgiunzione e la negazione logiche corrisponderanno a intersezione, unione e complemento, nel senso che abbiamo:

$$[\![\psi_1 \wedge \psi_2]\!]_T = [\![\psi_1]\!]_T \cap [\![\psi_2]\!]_T, \quad [\![\psi_1 \vee \psi_2]\!]_T = [\![\psi_1]\!]_T \cup [\![\psi_2]\!]_T, \tag{8}$$

$$[\![\neg\psi]\!]_T = W \setminus [\![\psi]\!]_T. \tag{9}$$

Cosa possiamo dire dei connettivi temporali? Cominciamo con $\mathbf{EX}$: questo connettivo corrisponde ad un'operazione che possiamo chiamare di *immagine inversa lungo la relazione* R e che è definita ponendo per $S \subseteq W$

$$R^-(S)\ :=\ \{s \in W \mid \exists s'\ (R(s, s')\ \&\ s' \in S)\}.$$

Con tale definizione a disposizione si verifica subito che vale

$$[\![\mathbf{EX}\,\psi]\!]_T = R^-([\![\psi]\!]_T). \tag{10}$$

Il fatto interessante che vedremo è che i connettivi $\mathbf{EG}, \mathbf{EU}$ corrispondono a punti fissi di operatori che si ottengono combinando operatori booleani e R^-.

Proposizione 4.2 *Sia dato un sistema di transizione $T = (W, w_0, R, v)$. Per ogni formula ψ si ha che*

$$[\![\mathbf{EG}\,\psi]\!]_T = \nu_X([\![\psi]\!]_T \cap R^-(X)).$$

Dim. Dal paragrafo 3.1 sappiamo che

$$[\![\mathbf{EG}\,\psi]\!]_T = \{s \in W \mid \exists\sigma\ \forall i \geq 0\ (s_i \in [\![\psi]\!]_T)\} \tag{11}$$

dove σ indica una s-esecuzione del tipo (5). Fissato $[\![\psi]\!]_T$, l'operatore che associa ad $X \subseteq W$ il sottoinsieme $[\![\psi]\!]_T \cap R^-(X)$ è chiaramente monotono, pertanto possiamo determinarne il massimo punto fisso per approssimazioni successive dal di sopra come in (6). Ricordando che vale $R^-(W) = W$ per la serialità della R, abbiamo:

$$N_0 = [\![\psi]\!]_T \cap R^-(W) = [\![\psi]\!]_T \tag{12}$$
$$N_1 = [\![\psi]\!]_T \cap R^-([\![\psi]\!]_T) \tag{13}$$
$$N_2 = [\![\psi]\!]_T \cap R^-([\![\psi]\!]_T \cap R^-([\![\psi]\!]_T)) \tag{14}$$
$$\cdots \tag{15}$$

Ora è chiaro che N_0 contiene gli stati che stanno in $[\![\psi]\!]_T$, N_1 gli stati che stanno in $[\![\psi]\!]_T$ e da cui si diparte un arco alla cui estremità c'è ancora uno stato che sta in $[\![\psi]\!]_T$, ecc. ecc. Uno stato s appartiene al punto fisso $\nu_X([\![\psi]\!]_T \cap R^-(X))$ se e solo se appartiene a tutti i N_i:[12] tale stato avrà quindi la proprietà che per ogni $i \geq 0$ esiste un cammino $s = s_0 R s_1 \cdots R s_i$ tutto incluso in $[\![\psi]\!]_T$. Tale cammino non è proprio una s-esecuzione tutta inclusa in $[\![\psi]\!]_T$ (come sarebbe richiesto da (11)), ma per i maggiore o uguale alla cardinalità di W in pratica la diventa: per un tale i gli stati $s_0, \ldots, s_i$ non possono essere tutti distinti, perciò si può ripetere il ciclo presente nel cammino infinite volte e ottenere proprio una s-esecuzione inclusa in $[\![\psi]\!]_T$. $\dashv$

Proposizione 4.3 *Sia dato un sistema di transizione $T = (W, w_0, R, v)$. Per ogni coppia di formule ϕ, ψ si ha che*

$$[\![\mathbf{E}\,[\phi\,\mathbf{U}\,\psi]]\!]_T = \mu_X([\![\psi]\!]_T \cup ([\![\phi]\!]_T \cap R^-(X))).$$

[12] Si ricordi che gli N_i formano una successione decrescente che si stabilizza, perciò l'intersezione di tale successione coincide con la sua stabilizzazione.

Dim. Seguiamo anche qui lo schema precedente e osserviamo dapprima che le condizioni semantiche di verità del paragrafo 3.1 ci dicono che

$$[\![\mathbf{E}\,[\phi\,\mathbf{U}\,\psi]]\!]_T = \{s \in W \mid \exists\sigma\ \exists i \geq 0\ (s_i \in [\![\psi]\!]_T\ \&\ \forall j < i\ (s_j \in [\![\phi]\!]_T))\} \quad (16)$$

dove σ indica una s-esecuzione del tipo (5). Fissati $[\![\psi]\!]_T$ e $[\![\psi]\!]_T$, l'operatore che associa ad $X \subseteq W$ il sottoinsieme $[\![\psi]\!]_T \cup ([\![\phi]\!]_T \cap R^-(X))$ è monotono, pertanto possiamo determinarne il minimo punto fisso per approssimazioni successive dal di sotto come in (7). Osservando preliminarmente che $R^-(\emptyset) = \emptyset$, abbiamo:

$$M_0 = [\![\psi]\!]_T \cup ([\![\phi]\!]_T \cap R^-(\emptyset)) = [\![\psi]\!]_T \quad (17)$$

$$M_1 = [\![\psi]\!]_T \cup ([\![\phi]\!]_T \cap R^-([\![\psi]\!]_T)) \quad (18)$$

$$\cdots \quad (19)$$

Ora è chiaro che M_0 contiene gli stati che stanno in $[\![\psi]\!]_T$, M_1 gli stati che o stanno in $[\![\psi]\!]_T$ oppure che stanno in $[\![\phi]\!]_T$ e da cui si diparte un arco alla cui estremità c'è uno stato che sta in $[\![\psi]\!]_T$, ecc. ecc. Ma allora uno stato s appartiene al punto fisso $\mu_X([\![\psi]\!]_T \cup ([\![\phi]\!]_T \cap R^-(X)))$ se e solo se esiste un cammino $s = s_0 R s_1 \cdots R s_i$ tale che $s_i \in [\![\psi]\!]_T$ e tale che per ogni $j < i$, $s_j \in [\![\phi]\!]_T$: questo è proprio quanto afferma il secondo membro della (16) (si ricordi che per la serialità, è sempre possibile estendere un cammino finito che parte da s ad una s-esecuzione). $\dashv$

Risultati di caratterizzazione tramite punti fissi *possono essere ottenuti anche in presenza di condizioni di fairness* (omettiamo i dettagli per cui rimandiamo al citato testo di Clarke et. al.).

5 Macchine a stati finiti

Informalmente, una macchina a stati finiti è un sistema (come quello studiato nel paragrafo 2) che si evolve secondo una data regola non deterministica e il cui stato corrente è completamente specificato dal valore di un numero finito di variabili, che a loro volta variano su domini finiti. Per semplicità (e senza reale perdita di generalità) assumiamo che tali domini coincidano sempre con l'insieme $\{0, 1\}$ e che quindi le variabili dei sistemi che studiamo siano tutte variabili booleane.

Prima di procedere, richiamiamo alcune definizioni sulla logica booleana. Dato un insieme Var di variabili proposizionali, l'insieme delle Var-formule booleane (o semplicemente formule booleane) è così definito:

- $0, 1$ e ogni $p \in Var$ sono formule booleane;

- se ϕ, ψ sono formule booleane, tali sono $(\neg\phi), (\phi \wedge \psi), (\psi \vee \phi)$.

In altre parole, sono formule booleane le formule di CTL che non contengono operatori temporali; si noti che abbiamo anche aggiunto le costanti 0 e 1 per indicare sintatticamente i valori di verità (cioè 1 denoterà la proposizione sempre vera e 0 la proposizione sempre falsa).

Con $\underline{p}, \underline{q}., \ldots$ indichiamo tuple di variabili proposizionali di cui non sentiamo il bisogno di specificare la lunghezza (es. $\underline{p}$ può stare per $p_1, \ldots, p_n$ o per $p_1, \ldots, p_m$, ecc.). Potremo anche usare $\underline{p}, q$ per indicare la tupla ottenuta giustapponendo q a $\underline{p}$; similmente potremo usare $\underline{p}, \underline{q}$ per la giustapposizione delle tuple $\underline{p}$ e $\underline{q}$.

Se $\underline{p}$ è una tupla di variabili proposizionali, con $\underline{p}'$ indichiamo una tupla rinominata di $\underline{p}$: se $\underline{p}$ è $p_1, \ldots, p_n$, allora $\underline{p}'$ sarà ad esempio $p'_1, \ldots, p'_n$.

La notazione $\alpha(\underline{p})$ indica che la formula booleana α è costruita utilizzando al più le variabili proposizionali della tupla $\underline{p}$.

I quantificatori proposizionali booleani avranno un ruolo cruciale in quanto segue; siccome essi sono eliminabili, li introduciamo come abbreviazioni (operiamo questa scelta per semplicità di esposizione e rimandiamo al paragrafo 5.2 seguente una breve discussione sul problema fondamentale della rappresentazione efficiente delle formule booleane e di tutte le operazioni su di esse). Se $\alpha(\underline{p}, q)$ è una formula booleana, le formule booleane $\exists q\, \alpha(\underline{p}, q)$ e $\forall q\, \alpha(\underline{p}, q)$ sono definite come segue:

$$\exists q\, \alpha(\underline{p}, q) \;:\equiv\; \alpha(\underline{p}, 1) \vee \alpha(\underline{p}, 0) \tag{20}$$

$$\forall q\, \alpha(\underline{p}, q) \;:\equiv\; \alpha(\underline{p}, 1) \wedge \alpha(\underline{p}, 0) \tag{21}$$

dove, ad esempio, $\alpha(\underline{p}, 1)$ indica la formula ottenuta sostituendo in $\alpha(\underline{p}, q)$ la variabile proposizionale q con 1. Le quantificazioni $\exists \underline{q}\, \alpha(\underline{p}, \underline{q})$ e $\forall \underline{q}\, \alpha(\underline{p}, \underline{q})$ sono definite in modo simile per iterazione. Giungiamo ora alle seguente fondamentale

Definizione 5.1 *Una* macchina a stati finiti *è una terna*

$$\mathcal{M} = \langle \underline{p}, \iota, \tau \rangle$$

dove $\underline{p}$ è una tupla di variabili proposizionali e $\iota(\underline{p}), \tau(\underline{p}, \underline{p}')$ sono due formule proposizionali booleane.

La tupla $\underline{p}$ rappresenta le variabili che, una volta istanziate su 0 e 1, descrivono lo stato corrente della macchina; la *condizione iniziale* $\iota(\underline{p})$ descrive l'insieme degli stati iniziali e la *regola di transizione* $\tau(\underline{p}, \underline{p}')$ descrive l'evoluzione non deterministica della macchina.

Per chiarire la definizione 5.2, possiamo ritornare sull'esempio del paragrafo 2. Una macchina a stati finiti per il sistema illustrato in tale esempio si può costruire così. Come tupla di variabili booleane prendiamo le 5 variabili

$$e_0,\ e_1,\ c_0,\ c_1,\ s$$

che significano rispettivamente che e_0 (e_1) vuole accedere alla sezione critica, che e_0 (e_1) è in sezione critica e che l'ultimo ad accodarsi è stato s ($s = 0, 1$). La formula ι che descrive i due stati iniziali è

$$\neg e_0 \wedge \neg e_1 \wedge \neg c_0 \wedge \neg c_1$$

(questo perchè all'inizio nessuno dei due processi è in sezione critica, nè ha ancora manifestato l'intenzione di entrarvi). La costruzione della formula di transizione τ è molto più complessa (paghiamo qui il fatto che un formalismo booleano puro è un formalismo di basso livello, la sintassi di un model-checker è usualmente di livello più alto e quindi è più maneggevole): per costruire la τ bsognerà usare una disgiunzione di casi. Svolgiamo un caso solo e lasciamo gli altri disgiunti al lettore: il caso di cui ci occupiamo è quello in cui e_0 entra in sezione critica. Ciò può avvenire, come sappiamo, solo se (i) c'è stata preventiva richiesta da parte di e_0 (per cui la variabile e_0 deve avere valore 1); (ii) e_0 non deve già essere in sezione critica (se no deve subito rilasciare la risorsa per quanto convenuto nel paragrafo 2); (iii) e_1 non deve aver fatto richiesta conflittuale o deve essersi accodato per ultimo. Tutto ciò è espresso dalla seguente formula booleana che comparirà come disgiunto della τ:

$$e_0 \wedge e_0' \wedge \neg c_0 \wedge c_0' \wedge (\neg e_1 \vee s) \wedge (s \leftrightarrow s').$$

Si noti che le variabili con l'apice servono a denotare il valore della corrispondente variabile booleana *dopo* la transizione al nuovo stato: ad esempio $\neg c_0 \wedge c_0'$ dice che e_0 non era in sezione critica e vi entrerà, così $s \leftrightarrow s'$ dice che il valore della variabile s non cambierà, ecc.

Associamo ad una macchina a stati finiti $\mathcal{M}$ un sistema di transizione $T^{\mathcal{M}}$: tale sistema di transizione sarà il grafo che rappresenta gli stati di $\mathcal{M}$ e la loro evoluzione. Per far questo, ci servono alcune definizioni preliminari.

Fissata una tupla finita di variabili $\underline{p}$, un *assegnamento* su $\underline{p}$ è una mappa s che associa un valore di verità ad ogni elemento di $\underline{p}$: tale s può essere esteso a tutte le formule booleane del tipo $\alpha(\underline{p})$ usando le tavole di verità. Se s è un assegnamento su $\underline{p}$, con s' indichiamo la copia rinominata di s su $\underline{p}'$: ad esempio, se $\underline{p}$ è $p_1, \ldots, p_n$, avremo che $\underline{p}'$ è $p'_1, \ldots, p'_n$ e che vale $s'(p') = s(p_i)$ per ogni $i = 1, \ldots, n$. Se s, t sono assegnamenti su $\underline{p}$, indicheremo con $\langle s, t' \rangle$ l'assegnamento $s \cup t'$ (che sarà un assegnamento su $\underline{p}, \underline{p}'$).

Definizione 5.2 *Sia data una macchina a stati finiti $\mathcal{M} = \langle \underline{p}, \iota, \tau \rangle$; il sistema di transizione (etichettato su $\underline{p}$)*

$$T^{\mathcal{M}} \;=\; (W^{\mathcal{M}}, W_0^{\mathcal{M}}, R^{\mathcal{M}}, v^{\mathcal{M}})$$

è così definito:

- $W^{\mathcal{M}}$ *è l'insieme di tutti gli assegnamenti su $\underline{p}$;*
- $W_0^{\mathcal{M}}$ *è l'insieme di tutti gli assegnamenti s tali che $s(\iota) = 1$;*
- $R^{\mathcal{M}}$ *è l'insieme di tutte le coppie di assegnamenti (s, t) tali che vale $\langle s, t' \rangle(\tau) = 1$;*
- $v^{\mathcal{M}}$ *associa ad ogni assegnamento se stesso.*

Il senso della definizione che abbiamo appena dato è semplicemente che: (i) $T^{\mathcal{M}}$ ha per stati gli assegnamenti booleani alle variabili proposizionali di $\mathcal{M}$; (ii) gli archi $R^{\mathcal{M}}(s, t)$ corrispondono a passaggi di stato (da s a t) che sono in accordo con la regola di evoluzione τ di $\mathcal{M}$ (siccome τ è non deterministica, da s sarà in generale possibile passare a più di un t).

5.1 Model-Checking simbolico

Il problema del **model checking** per una macchina a stati finiti $\mathcal{M}$ è il seguente: *data una formula ϕ di CTL, stabilire se $T^{\mathcal{M}}$ soddisfa la specifica ϕ.*

Questo problema potrebbe certamente essere risolto con gli algoritmi derivanti dai risultati del paragrafo 3.3, tuttavia l'uso di tali algoritmi presupporrebbe una esplorazione di tutto il grafo degli stati $T^{\mathcal{M}}$, che ha dimensione esponenziale rispetto a $\mathcal{M}$. L'idea alternativa che illustreremo sarà differente: trasformeremo la formula ϕ in una formula booleana ϕ^B, in modo che

il problema si riduca a testare la tautologicità di $\iota \rightarrow \phi^B$. Questa trasformazione di ϕ in ϕ^B sarà puramente *simbolica* e il grafo degli stati $T^{\mathcal{M}}$ non comparirà mai in modo esplicito. La pratica sperimentale ha dimostrato che l'approccio simbolico può essere una soluzione interessante al problema dell'esplosione degli stati se accompagnato da una rappresentazione efficiente delle formule booleane.

Tutto quello che dobbiamo fare è semplicemente di sfruttare le informazioni in nostro possesso sui punti fissi e riformularle a livello di macchine a stati finiti. Per apprezzare l'enunciato del prossimo teorema-chiave, si ricordi che se ψ è una formula di CTL, $[\![\psi]\!]_{T^{\mathcal{M}}}$ indica l'insieme degli stati s tali che $T^{\mathcal{M}} \models_s \psi$ (si ricordi la definizione di verità 3.1 per la logica temporale CTL). Se ψ è booleana (cioè se non ha operatori temporali), è immediato vedere che $[\![\psi]\!]_{T^{\mathcal{M}}}$ contiene gli stati s tali che $s(\psi) = 1$, ossia $[\![\psi]\!]_{T^{\mathcal{M}}}$ contiene gli assegnamenti s che soddisfano ψ nel senso delle tavole di verità: in effetti, fissato uno stato-alias-assegnamento s di $T^{\mathcal{M}}$, la definizione di verità 3.1 coincide con quella delle ordinarie tavole di verità per quanto riguarda le formule booleane.

Teorema 5.3 *Siano $\mathcal{M} = \langle \underline{p}, \iota, \tau \rangle$ una macchina a stati finiti e ϕ una formula di CTL contenente al più le variabili proposizionali $\underline{p}$. È possibile calcolare una formula booleana $\phi^B(\underline{p})$ tale che*

$$[\![\phi]\!]_{T^{\mathcal{M}}} = [\![\phi^B]\!]_{T^{\mathcal{M}}}; \tag{22}$$

di conseguenza, abbiamo che $T^{\mathcal{M}}$ soddisfa la specifica ϕ se e solo se $[\![\iota]\!]_{T^{\mathcal{M}}} \subseteq [\![\phi^B]\!]_{T^{\mathcal{M}}}$ (ossia se e solo se $\iota \rightarrow \phi^B$ è una tautologia).[13]

Dim. Per induzione sul numero di operatori logici di ϕ. Se ϕ è atomica, ϕ^B è uguale a ϕ e, se ϕ è del tipo $\psi_1 \wedge \psi_2, \psi_1 \vee \psi_2, \neg\psi$, allora ϕ^B è rispettivamente uguale a $\psi_1^B \wedge \psi_2^B, \psi_1^B \vee \psi_2^B, \neg\psi^B$ (la proprietà (22) è ovvia per ipotesi induttiva e per le (8)-(9)). Se ϕ è del tipo $\mathbf{EX}\,\psi$, allora ϕ^B sarà uguale a $\exists \underline{p}'(\tau(\underline{p}, \underline{p}') \wedge \psi^B(\underline{p}'))$ (la proprietà (22) è ora garantita dalla (10)). Negli ultimi due casi che restano, usiamo la teoria dell'approssimazione dei massimi e minimi punti fissi.

[13] Anche lo studente che abbia concordato un programma non comprendente lo studio delle dimostrazioni delle presenti note deve comunque avere in mente la procedura di calcolo della ϕ^B illustrata qui sotto.

Se ϕ è $\mathbf{EG}\,\psi$, ricordiamo l'enunciato della proposizione 4.2 e consideriamo le formule:

$$\begin{aligned} N_0 &:= \psi^B; \\ N_1 &:= \psi^B \wedge \exists \underline{p}'(\tau(\underline{p}, \underline{p}') \wedge \psi^B(\underline{p}')); \\ N_2 &:= \psi^B \wedge \exists \underline{p}'(\tau(\underline{p}, \underline{p}') \wedge \psi^B(\underline{p}') \wedge \exists \underline{p}''(\tau(\underline{p}', \underline{p}'') \wedge \psi^B(\underline{p}''))) \\ &\dots \end{aligned}$$

Queste formule booleane (sempre per le (8)-(9), (10)) rappresentano proprio le approssimazioni (12)-(15) del massimo punto fisso $[\![\mathbf{EG}\,\psi]\!]_{T^{\mathcal{M}}} = \nu_X([\![\psi]\!]_{T^{\mathcal{M}}} \cap R^-(X))$. Per questo motivo avremo che per un certo $i \geq 0$ esse si stabilizzano, nel senso che $N_i \leftrightarrow N_{i+1}$ sarà una tautologia per qualche i: porremo allora ϕ^B uguale a N_i per tale i.

Il discorso è del tutto analogo nel caso in cui ϕ è $\mathbf{E}\,[\psi_1\,\mathbf{U}\,\psi_2]$: tenendo in mente l'enunciato della proposizione 4.3, consideriamo dapprima le formule

$$\begin{aligned} M_0 &:= \psi^B; \\ M_1 &:= \psi^B \vee (\phi^B \wedge \exists \underline{p}'(\tau(\underline{p}, \underline{p}') \wedge \psi^B(\underline{p}'))); \\ &\dots \end{aligned}$$

(suggerite dalle (17)-(19)) e poniamo ϕ^B uguale a M_i, dove i è tale che $M_i \leftrightarrow M_{i+1}$ è una tautologia. $\dashv$

5.2 Il ruolo degli OBDD

La funzionalità dell'algoritmo simbolico suggerito dal teorema 5.3 è tutta legata all'efficienza della rappresentazione delle formule booleane e delle operazioni su di esse. Le due principali fonti di preoccupazione sono il calcolo iterato di quantificatori booleani tramite le (20)-(21) (che può produrre un'esplosione in memoria) e i test di equivalenza logica necessari a stabilizzare le approssimazioni dei punti fissi.

Gli OBDD ("ordered binary decision diagrams") forniscono una risposta adeguata, in termini di strutture-dati, alle nostre esigenze e rappresentano il cuore dei moderni model-checkers simbolici. Fissato un ordinamento sulle variabili proposizionali tramite opportune euristiche, un OBDD rappresenta una formula booleana tramite un grafo aciclico diretto i cui nodi interni sono etichettati (in modo decresecente secondo l'ordinamento scelto) con

variabili proposizionali e i cui nodi esterni sono etichettati da valori di verità. Un OBDD si può normalizzare in tempo lineare e, una volta normalizzato, esso rappresenta in modo canonico (pertanto *unico*) una formula booleana, fatto che rende immediati i test di equivalenza logica che ci servono. Non ci soffermeremo oltre sugli OBDD: contiamo di fornire più informazioni durante le lezioni del corso e in una versione ampliata delle presenti note (chi volesse saperne di più, può consultare il paragrafo 5.1 del citato testo Clarke et al., che fornisce un'introduzione sintetica e molto chiara all'argomento).

6 Verifica formale tramite un model-checker

Lo studente è incoraggiato a scaricare ed installare il sistema NuSMV che implementa le nozioni spiegate a grandi linee in queste note. La distribuzione di NuSMV include un manuale, un tutorial e vari file eseguibili; tutto questo materiale è da considerarsi *materiale di consultazione* a tutti gli effetti per la preparazione dell'esame. In questo paragrafo finale riprendiamo semplicemente l'esempio dell'algoritmo di mutua esclusione di Peterson illustrato nel paragrafo 2.

Nelle Figure 1,2, 3 seguenti sono riprodotti nella sintassi di NuSMV un modulo principale e due moduli secondari che eseguono l'algoritmo di Peterson (i tre moduli sono stati separati per ragioni grafiche, vanno in realtà messi insieme all'interno di un unico file).

Come si vede, il modulo principale utilizza tre variabili booleane globali:[14] `e0` ed `e1` indicano la volontà da parte dei processi pr_0 e pr_1 di entrare in sezione critica (mentre `s` indica il turno). I processi pr_0 e pr_1 sono specificati mediante le variabili `p0` e `p1` che realizzano rispettivamente un'istanza del modulo `pr0` e un'istanza del modulo `pr1`. La presenza della parola riservata `process` nelle relative dichiarazioni indica che `p0` e `p1` saranno eseguiti in modo *asincrono* (ad ogni passo, uno dei due sarà casualmente selezionato ed attivato). Le tre variabili booleane `e0, e1, s` vengono inizializzate a 0 all'interno del modulo principale (esse verranno poi modificate da `p0` e `p1` durante la loro attivazione). A completamento del modulo principale sono state inserite le specifiche CTL discusse nel paragrafo 2.

Ciascuno dei due moduli secondari (Figg. 2 e 3) dipende da tre parametri formali e manipola anche una variabile locale `critical` che indica lo stato

[14] È possibile scrivere una versione dello stesso programma con la sola variabile globale `s`.

```
MODULE main

VAR
s:   boolean;
e0:  boolean;
e1:  boolean;
p0:  process pr0(s,e0,e1);
p1:  process pr1(s,e0,e1);

ASSIGN
init(s)  := 0;
init(e0) := 0;
init(e1) := 0;

SPEC AG !(p0.critical & p1.critical);
SPEC AG (e0 -> AF p0.critical);
SPEC AG (e1 -> AF p1.critical);
SPEC AG (e0 & !e1 -> A [(!p1.critical) U (p0.critical)])
SPEC AG (e1 & !e0 -> A [(!p0.critical) U (p1.critical)])
```

Figura 1: Modulo main

di accesso alla risorsa. Per referenziare tale variabile all'esterno dei moduli, si deve usare la notazione con 'dot' (ossia ad esempio, `p0.critical` denota il valore di `critical` dentro a `p0` - si vedano le specifiche CTL all'interno del modulo principale in Fig.1). Ogni istanza di un modulo ha anche associata una variabile locale implicita `running` che indica l'attivazione del processo stesso (tale variabile può essere referenziata con le stesse modalità delle altre variabili locali).

Il comportamento di `pr0` descritto nella Fig.2 è quello prescritto dall'algoritmo per il processo pr_0. La variabile `critical` è inizializzata a 0 e viene posta uguale ad 1 se `e0` vale 1 e se (o `e1` vale 0 oppure `s` $\neq 0$); `critical` è rimessa a 0 se valeva 1 nell'istante precedente. Queste operazioni sono descritte nella relativa definizione per casi di `next(critical)` della Figura 2.

Le definizioni per casi di `next(s)` e `next(e0)` sono quelle attese; si osservi che nell'ultimo caso della definizione di `e0` il valore non è completamente

determinato, ma viene scelto nell'insieme $\{0, 1\}$ (questo perchè non è determinato in partenza se pr_0 voglia o meno entrare in sezione critica). Si osservi anche che nella definizione di `next(s)` si utilizzano sia il valore corrente che il valore precedente di `e0`; queste riferimenti innestati sono possibili nelle definizioni, purchè non si verifichino circoli viziosi (che l'interprete peraltro rileva). È anche possibile utilizzare definizioni esplicite delle variabili e non solo assegnare loro un valore tramite `init` e `next`.

Per concludere la descrizione dei moduli della Figure 2 e 3, è stata aggiunta la condizione di fairness descritta nel paragrafo 2. L'intero file (consistente dei tre moduli) può ora essere eseguito da NuSMV. Sono possibili sia un'esecuzione in modalità batch che dà subito il risultato, sia un'esecuzione in modalità interattiva che permette all'utente di costruirsi parzialmente o totalmente una traccia di evoluzione del sistema.

Per maggiori dettagli sulla sintassi e sull'uso di NuSMV, rimandiamo alla documetazione della distribuzione.

```
MODULE pr0(s,e0,e1)

VAR
critical :  boolean;

ASSIGN

init(critical) := 0;

next(critical) :=
case
!critical & e0 & !e1          :1;
!critical & e0 & !s=0         :1;
1                             :0;
esac;

next(e0) :=
case
critical            :0;
!critical & e0      :1;
1                   :{0,1};
esac;

next(s) :=
case
!e0 & next(e0)   :0;
1                :s;
esac;

FAIRNESS
running
```

Figura 2: Modulo pr0

```
MODULE pr1(s,e0,e1)

VAR
critical :  boolean;

ASSIGN

init(critical) := 0;

next(critical) :=
case
!critical & e1 & !e0          :1;
!critical & e1 & !s=1         :1;
1                             :0;
esac;

next(e1) :=
case
critical            :0;
!critical & e1      :1;
1                   :{0,1};
esac;

next(s) :=
case
!e1 & next(e1)   :1;
1                :s;
esac;

FAIRNESS
running
```

Figura 3: Modulo pr1