Logica Temporale Lineare

Silvio Ghilardi

23 Maggio 2007

In questi appunti vengono studiati alcuni formalismi per il ragionamento sul flusso temporale discreto, prendendo in considerazione sia problemi di soddisfacibilità che di model-checking. Si introdurranno la logica temporale LTL e la logica del secondo ordine $S1S$; se ne sfrutteranno le relazioni con gli automi di Büchi per ricavarne importanti procedure di decisione.

La parte relativa a LTL (ossia i capitoli 1-2-3 seguenti) costituisce l'oggetto delle lezioni di una parte del modulo del corso di Logica II dedicato alla logica temporale. Questi appunti (pur sufficienti per la preparazione dell'esame) hanno carattere piuttosto sintetico; ulteriori esempi e motivazioni sono stati sviluppati durante le lezioni del corso e si possono trovare nei manuali sull'argomento (ad esempio in Clarke, Grumberg, Peled *Model Checking*, MIT Press, 2000).[1]

Per il supporto sofware sulle tematiche relative al model-checking esplicito, segnaliamo il sistema SPIN alla pagina Web

http://spinroot.com/spin/whatispin.htmlh

Una implementazione della traduzione delle formule di LTL in automi di Büchi si può trovare sul sito

http://www.liafa.jussieu.fr/ oddoux/ltl2ba/

Il model-checking simbolico è invece trattato in una dispensa a parte, necessaria per la preparazione dell'esame e disponibile anch'essa sul sito del corso di Logica II.

[1] Chi si trovasse in difficoltà a reperire il testo sopra indicato, si può rivolgere al docente.

Indice

1 La logica temporale lineare

Lo scopo di queste note è di introdurre formalismi logici per la specifica di comportamenti di sistemi soggetti ad evoluzione nel tempo; tali formalismi si sono rivelati di importanza cruciale nelle applicazioni orientate alla verifica di programmi e protocolli. Il primo formalismo logico di cui ci occupiamo è la logica del tempo lineare LTL.

1.1 Sintassi

La logica temporale lineare consente l'utilizzo, oltre che degli operatori booleani, anche di connettivi temporali. In particolare, consideremo gli operatori:

- $\Diamond\phi$: "ϕ sarà vera";
- $\Box\phi$: "ϕ sarà sempre vera";
- $\phi U\psi$: "ϕ sarà vera finchè non si verifica ψ;
- $X\phi$: "ϕ sarà vera nel prossimo istante"

(l'operatore X acquista senso perchè semanticamente ci limitiamo solo a flussi di tempo discreti). Per trattare forme normali negative, useremo anche un operatore R, detto operatore di 'release': R sarà il duale di U, nel senso che la seguente equivalenza sarà valida:

$$\phi R\psi \quad \leftrightarrow \quad \neg((\neg\phi)U(\neg\psi)).$$

Formalmente, dato un insieme Var di variabili proposizionali, l'insieme delle Var-formule (o semplicemente formule) di LTL 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 $(X\phi), (\Box\phi), (\Diamond\phi), (\phi U\psi), (\phi R\psi)$.

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 $p \vee \neg p$, $\neg\top$, $\neg\phi \vee \psi$, $(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$.

1.2 Semantica

Un *Var-modello* (o semplicemente modello) per LTL è una quintupla

$$\mathcal{M} = (\mathbb{N}, 0, s, <, V)$$

dove $(\mathbb{N}, 0, s, <)$ è la struttura standard dei numeri naturali con zero, successore e la relazione di minore e dove $V : \mathbb{N} \longrightarrow \wp(Var)$ è una funzione che assegna ad ogni istante di tempo l'insieme delle variabili proposizionali 'vere' in esso.

Se ϕ è una formula e $t \in \mathbb{N}$ un istante di tempo, la nozione di "verità di ϕ in $\mathcal{M}$ al tempo t" (scritta con $\mathcal{M} \models_t \phi$ o semplicemente con $\models_t \phi$) è così definita:

- $\models_t p$ sse $p \in V(t)$;
- $\models_t \neg\phi$ sse $\not\models_t \phi$;
- $\models_t \phi \wedge \psi$ sse ($\models_t \phi$ e $\models_t \psi$);
- $\models_t \phi \vee \psi$ sse ($\models_t \phi$ o $\models_t \psi$);
- $\models_t X\phi$ sse $\models_{s(t)} \phi$;
- $\models_t \Box\psi$ sse $(\forall t' \geq t \ \models_{t'} \psi)$;
- $\models_t \Diamond\psi$ sse $(\exists t' \geq t \ \models_{t'} \psi)$;
- $\models_t \phi U \psi$ sse $(\exists t' \geq t \ (\models_{t'} \psi \ \& \ \forall t'' \ (t \leq t'' < t' \ \Rightarrow \models_{t''} \phi))$;
- $\models_t \phi R \psi$ sse $(\forall t' \geq t \ (\models_{t'} \psi$ o $\exists t'' \ (t \leq t'' < t' \ \& \ \models_{t''} \phi))$.

1.3 Assiomatizzazione (cenni)

Riportiamo qui di seguito alcuni esempi di formule valide (cioè vere in ogni modello):

$$\Diamond\phi \leftrightarrow \top U\phi; \tag{1}$$
$$\Box\phi \leftrightarrow \neg\Diamond\neg\phi; \tag{2}$$
$$\Box(\phi \rightarrow \psi) \rightarrow (\Box\phi \rightarrow \Box\psi); \tag{3}$$
$$X(\phi \rightarrow \psi) \rightarrow (X\phi \rightarrow X\psi); \tag{4}$$
$$\neg X\phi \leftrightarrow X\neg\phi; \tag{5}$$
$$(\phi U\psi) \leftrightarrow (\psi \vee (\phi \wedge X(\phi U\psi))); \tag{6}$$
$$(\phi R\psi) \leftrightarrow \neg((\neg\phi)U(\neg\psi)); \tag{7}$$
$$\Box((\psi \vee (\phi \wedge X\theta) \rightarrow \theta)) \rightarrow ((\phi U\psi) \rightarrow \theta); \tag{8}$$
$$(\phi R\psi) \leftrightarrow \Box\psi \vee (\psi U(\phi \wedge \psi)). \tag{9}$$

Le formule (1), (2), (7), (9) chiariscono alcune relazioni di interdefinibilità. Le formule (6) e (8) rappresentano un'equazione di minimo punto fisso (in un senso che andrebbe precisato). Le formule (3)-(4) sono note in logica modale come assiomi di Aristotele e la (5) come assioma di funzionalità.

Esempio 1.1 A titolo di esempio verifichiamo la validità delle formule del tipo (9). Per l'implicazione da sinistra a destra, supponiamo che $\mathcal{M}$ sia un modello tale che per un certo $t \in \mathbb{N}$ valga $\mathcal{M} \models_t \phi R\psi$ e $\mathcal{M} \not\models_t \Box\psi$: dobbiamo provare che vale $\mathcal{M} \models_t \psi U(\phi \wedge \psi)$. Sia $i \geq t$ il minimo i per cui $\mathcal{M} \not\models_i \psi$; siccome vale $\mathcal{M} \models_t \phi R\psi$, deve essere $i > t$, poniamo $i := t+j+1$. Poichè i è stato scelto minimo, abbiamo $\mathcal{M} \models_t \psi, \mathcal{M} \models_{t+1} \psi, \ldots, \mathcal{M} \models_{t+j} \psi$. Ma allora, siccome $\mathcal{M} \models_t \phi R\psi$, deve esistere k (con $t \leq k \leq t+j$) tale che $\mathcal{M} \models_k \phi$. Si ha allora $\mathcal{M} \models_t \psi, \ldots, \mathcal{M} \models_{k-1} \psi, \mathcal{M} \models_k \phi \wedge \psi$, da cui $\mathcal{M} \models_t \psi U(\phi \wedge \psi)$.

Viceversa, se vale $\mathcal{M} \models_t \Box\psi \vee (\psi U(\phi \wedge \psi))$, proviamo che vale $\mathcal{M} \models_t \phi R\psi$. Se vale $\mathcal{M} \models_t \Box\psi$ banalmente vale anche $\mathcal{M} \models_t \phi R\psi$; altrimenti, vale $\mathcal{M} \models_t \psi U(\phi \wedge \psi)$ ed esisterà un $\tilde{t} \geq t$ tale che $\mathcal{M} \models_t \psi, \ldots, \mathcal{M} \models_{\tilde{t}-1} \psi, \mathcal{M} \models_{\tilde{t}} \phi \wedge \psi$. Consideriamo ora un generico $t' \geq t$ tale che $\mathcal{M} \not\models_{t'} \psi$: per tale t' dobbiamo far vedere che esiste t'' con $t \leq t'' < t'$ e $\mathcal{M} \models_{t''} \phi$ e, in effetti, possiamo scegliere $\tilde{t}$ come t''.

Si può provare che le formule (1)-(8) (unite alle tautologie classiche) e le regole di inferenza di modus ponens $(\phi, \phi \rightarrow \psi/\psi)$ e di necessitazione $(\phi/\Box\phi)$ costituiscono un'assiomatizzazione di LTL rispetto alla semantica sopra descritta. Tuttavia non ci occuperemo di problemi assiomatici, quanto piuttosto di problemi algoritmici motivati dalle applicazioni.

1.4 Soddisfacibilità e Model-Checking

Il problema della *soddisfacibilità* per LTL è il seguente: data ϕ, stabilire se esiste un modello $\mathcal{M}$ tale che $\mathcal{M} \models_0 \phi$. Vedremo che tale problema è decidibile, ma che la sua complessità è PSPACE (il problema è PSPACE-completo). Questo vuol dire che di fatto (a meno che emergano sconvolgimenti nelle attuali conoscenze nel campo della teoria della complessità degli algoritmi), occorre tempo esponenziale per risolvere il problema, anche se con un'occupazione di memoria polinomiale, appunto.

Per introdurre l'altro problema di cui ci occuperemo, occorre una semantica alternativa per LTL, basata sulla nozione di esecuzione ('run') in un sistema di transizione. Fissiamo come sempre un insieme Var di variabili proposizionali; un *sistema di transizione* è una quadrupla

$$T \;=\; (W, w_0, R, v)$$

dove W è un insieme finito (detto insieme degli stati del sistema), $w_0 \in W$ è lo stato iniziale, $R \subseteq W \times W$ è una relazione[2] su W e $v : W \longrightarrow \wp(Var)$ è una funzione che assegna ad ogni stato l'insieme delle variabili proposizionali 'vere' in esso. Un'*esecuzione* σ su T è una successione infinita

$$\sigma = \sigma(0), \sigma(1), \sigma(2), \ldots$$

di elementi di W tale che $\sigma(0) = w_0$ e $R(\sigma(i), \sigma(i+1))$ per ogni $i \geq 0$. Se σ è un'esecuzione su T, chiamiamo $\mathcal{M}_\sigma$ il modello

$$\mathcal{M}_\sigma = (\mathbb{N}, 0, s, <, V_\sigma)$$

dove $V_\sigma(t)$ è definita come $v(\sigma(t))$.

Si dice che T soddisfa una formula ϕ di LTL (scritto con $T \models \phi$) qualora si abbia $\mathcal{M}_\sigma \models_0 \phi$ *per ogni* esecuzione σ. Il problema del *model-checking* per T, ϕ è il seguente: stabilire se vale $T \models \phi$ oppure no. Come vedremo il problema risulta decidibile: la sua complessità è esponenziale nella lunghezza

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

di ϕ, ma lineare nella dimensione di T. Questo risultato è da considerarsi relativamente buono, perchè nelle applicazioni T è molto grande, mentre la lunghezza $|\phi|$ di ϕ è usualmente contenuta, trattandosi della lunghezza della specifica del comportamento che si vuole testare.

1.5 Forme Normali Negative

A conclusione del paragrafo, segnaliamo che ogni formula ϕ di LTL è equivalente ad una formula $N(\phi)$ in *forma normale negativa* (nel senso che in ogni modello $\mathcal{M}$ si ha $\mathcal{M} \models_t \phi \leftrightarrow N(\phi)$ per ogni t). Come sempre, una formula è in forma normale negativa (abbreviato fnn) se contiene negazioni solo davanti a formule atomiche. Per trovare $N(\phi)$ basta applicare le seguenti regole di riscrittura, la cui terminazione si può verificare facilmente (ad esempio con un ordinamento del tipo LPO):

$$\begin{aligned}
\neg(\phi \wedge \psi) &\rightsquigarrow \neg\phi \vee \neg\psi \\
\neg(\phi \vee \psi) &\rightsquigarrow \neg\phi \wedge \neg\psi \\
\neg\neg\phi &\rightsquigarrow \phi \\
\neg\Diamond\phi &\rightsquigarrow \Box\neg\phi \\
\neg\Box\phi &\rightsquigarrow \Diamond\neg\phi \\
\neg X\phi &\rightsquigarrow X\neg\phi \\
\neg(\phi U\psi) &\rightsquigarrow \neg\phi R\neg\psi \\
\neg(\phi R\psi) &\rightsquigarrow \neg\phi U\neg\psi.
\end{aligned}$$

Dopo aver ridotto una formula in formula normale negativa, possiamo eliminare da essa i connettivi $\Diamond$ e R, grazie alla validità delle equivalenze (1) e (9): si noti che il relativo rimpiazzamento non introduce negazioni e non ci fa uscire quindi dalla classe delle formule in fnn.[3] D'ora in poi, quindi, *supporremo che tutte le formule che incontriamo siano costruite a partire da letterali* (cioè formule atomiche o negate) *applicando solo gli operatori* $\wedge, \vee, X, \Box$ *e* U.

[3]Segnaliamo tuttavia che, per evitare che l'eliminazione suddetta del connettivo R produca esplosioni esponenziali in lunghezza, occorrebbero accorgimenti analoghi a quelli utilizzati nelle trasformazioni strutturali per evitare l'uso delle leggi distributive nella riduzione in CNF (modulo equisoddisfacibilità) delle formule proposizionali booleane. In ogni caso, la nostra scelta di usare solo formule in fnn in cui i connettivi $\Diamond$ e R siano stati eliminati è dovuta solo a semplicità di esposizione, senza alcuna rivendicazione di efficienza.

2 Automi su Parole Infinite

Un modello $\mathcal{M} = (\mathbb{N}, 0, s, <, V)$ può essere identificato con la successione di insiemi di lettere proposizionali

$$V(0), V(1), V(2), \ldots \tag{10}$$

ossia con una *parola infinita* sull'alfabeto $\Sigma := \wp(Var)$, dove Var è il nostro insieme di variabili proposizionali. Nei casi concreti, Var è finito, perchè è l'insieme delle lettere proposizionali che occorrono nella formula φ che si testa per la soddisfacibilità o per il model-checking, quindi Σ stesso è un alfabeto finito.

È chiaro allora che $\mathcal{M} \models_0 \phi$ può essere tradotto in un problema di riconoscimento delle parole (10) che corrispondono alle strutture che sono modello di ϕ. È altresì noto che il formalismo usuale per problemi di riconoscimento è costituito dagli automi.

2.1 Automi di Büchi e Linguaggi ω-regolari

La differenza rilevante fra i problemi che ci accingiamo a trattare e la teoria degli automi a stati finiti classici consiste nel fatto che le stringhe da riconoscere sono infinite, quindi l'accettazione non può far riferimento allo stato dell'automa al momento della terminazione, perchè ora non c'è terminazione (la terminazione significa impossibilità di proseguire e quindi è un caso particolare di non accettazione). Negli automi di Büchi (i più semplici fra gli automi che lavorano su parole infinite) la condizione di accettazione consiste nel fatto che uno stato finale sia *visitato infinite volte*. Formalmente, le definizioni che ci servono sono le seguenti:

Definizione 2.1 *Un automa di Büchi è una quintupla*

$$A \;=\; (\Sigma, Q, q_I, \delta, F)$$

dove

- Σ *è un insieme finito, detto alfabeto di* A*;*
- Q *è un insieme finito, detto insieme degli stati di* A*;*
- $q_I \in Q$ *è lo stato iniziale di* A*;*

- $\delta : \Sigma \times Q \longrightarrow \wp(Q)$ *è una funzione, detta transizione di* A*;*

- $F \subseteq Q$ *è l'insieme degli stati finali di* A*.*

Nel seguito, parlando di automi, intenderemo sempre automi di Büchi. Un automa A è detto deterministico quando per ogni $a \in \Sigma, q \in Q$ si ha che $\delta(a, q)$ contiene un unico elemento. Con Σ^ω indichiamo le parole infinite su Σ, ossia le sucessioni $\sigma : \mathbb{N} \longrightarrow \Sigma$; un ω-linguaggio su Σ è un sottoinsieme di Σ^ω.

Definizione 2.2 *Una esecuzione ('run') di un'automa* $A = (\Sigma, Q, q_I, \delta, F)$ *su* $\sigma \in \Sigma^\omega$ *è una successione infinita di stati*

$$\rho = \rho(0), \rho(1), \rho(2), \ldots$$

tale che $\rho(0) = q_I$ *e tale che* $\rho(i+1) \in \delta(\sigma(i), \rho(i))$*. L'esecuzone* ρ *accetta* σ *sse esiste* $q \in F$ *tale che* q *occorre infinite volte in* $\{\rho(0), \rho(1), \rho(2), \ldots\}$*. Si dice che* A *accetta* σ *qualora esista un'esecuzione accettante di* A *su* σ*.*

Siccome F è finito, la condizione di accettazione si può equivalentemente formulare dicendo che per ogni i esiste $j \geq i$ tale $\rho(j) \in F$.

Definizione 2.3 *Un* ω*-linguaggio* L *è detto* ω*-regolare sse esiste un automa* A *che accetta precisamente le* $\sigma \in L$*.*

Esempio. Sia $\Sigma = \{a, b\}$ e sia L l'insieme delle σ del tipo $\{a, b\}^* a^\omega$, cioè del tipo

$$a_1, \ldots, a_n, a, a, a, a, \ldots \qquad (a_i \in \{a, b\})$$

oppure del tipo $\{a, b\}^*(ab)^\omega$, cioè del tipo

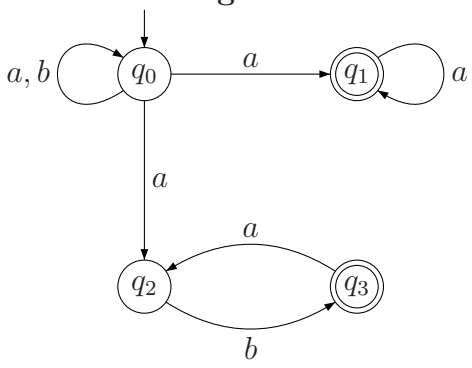
$$a_1, \ldots, a_n, a, b, a, b, a, b, \ldots \qquad (a_i \in \{a, b\}).$$

Si può provare che L è ω-regolare essendo il linguaggio accettato dall'automa della Figura 1.[4]

Per certi versi, la teoria degli automi di Büchi assomiglia alla teoria degli automi a stati finiti; tuttavia alcuni risultati cruciali sono molto più difficili da dimostrare e altri risultati non valgono più. Ad esempio, non è vero che

[4]Come sempre, nella raffigurazione di un automa, con il doppio cerchio vengono indicati gli stati finali, mentre lo stato iniziale è puntato da un arco senza nodo sorgente).

Figura 1



dato un automa di Büchi A esiste sempre un automa di Büchi deterministico A' tale che $L(A) = L(A')$ (con $L(A)$ indichiamo l'ω-linguaggio formato dalle parole infinite accettate da A): per ottenere questa proprietà, occorrono automi su parole infinite più generali degli automi di Büchi, su cui non ci soffermiamo.

2.2 Automi vuoti

Consideriamo per primo il problema di determinare, dato $A = (\Sigma, Q, q_I, \delta, F)$, se A è *vuoto*, ossia se $L(A) = \emptyset$. A tale scopo, costruiamo il grafo[5] G_A: questo grafo ha per nodi gli stati di A e per archi $q \to r$ le coppie (q, r) tali che esiste $a \in \Sigma$ con $r \in \delta(a, q)$.

Proposizione 2.4 *Un automa di Büchi $A = (\Sigma, Q, q_I, \delta, F)$ è non vuoto sse esiste $f \in F$ ed esistono due cammini del tipo*

$$q_I \to \cdots \to f \qquad e \qquad f \to \cdots \to f$$

dentro a G_A.

Dim. Sia ρ un'esecuzione che accetta una $\sigma \in \Sigma^\omega$ e sia f uno stato finale che occorre infinite volte in tale esecuzione. Siccome in G_A abbiamo

$$q_I = \rho(0) \to \rho(1) \to \cdots$$

[5]In queste note, per grafo intenderemo sempre un *grafo orientato*, ossia un grafo sarà semplicemente un insieme dotato di una relazione binaria.

avremo anche i due cammini richiesti. Viceversa, siano dati tali cammini; se scriviamo $q \xrightarrow{a} r$ per $r \in \delta(a, q)$, avremo

$$q_I \xrightarrow{a_1} \cdots \xrightarrow{a_n} f \xrightarrow{b_1} \cdots \xrightarrow{b_m} f.$$

Allora A accetta la parola $a_1, \ldots, a_n(b_1, \ldots, b_m)^\omega$. $\dashv$

Dalla Proposizione precedente segue che il problema di riconoscere se A è vuoto è di classe NLOGSPACE (cioè è risolvibile usando risorse logaritmiche di memoria da una macchina non deterministica). Infatti, una macchina non deterministica prima indovina $f \in F$ e poi, dato lo stato corrente, indovina uno stato legato ad esso tramite un arco in G_A; si ferma quando, partendo da q_I, raggiunge f due volte. Per mantenere in memoria lo stato corrente, basta mantenerne in memoria l'indirizzo, per cui è sufficiente spazio logaritmico.

Siccome NLOGSPACE $\subseteq$ NC $\subseteq$ P, il problema di riconoscere se A è vuoto è risolvibile in tempo polinomiale; in realtà, con opportune considerazione su cui non ci soffermiamo, si potrebbe addirittura provare che è sufficiente un tempo lineare.

Osservazione 2.5 É possibile limitare leggermente meglio la lunghezza dei cammini richiesti dalla Proposizione 2.4, dando alla stessa Proposizione una formulazione lievemente differente. Un *palloncino* in $A = (\Sigma, Q, q_I, \delta, F)$ è un cammino in G_A

$$q_I = q_0 \to q_1 \to \cdots \to q_n$$

tale che esistono $0 \leq i \leq j \leq n$ tali che: (i) $q_n \to q_i$; (ii) $q_j \in F$.

Proviamo che $L(A) \neq \emptyset$ se e solo se esiste un palloncino in A di lunghezza minore o uguale alla cardinalità dell'insieme Q degli stati di A. Un lato di tale asserto è banalmente vero per la Proposizione 2.4; per l'altro lato supponiamo che $L(A) \neq \emptyset$ e consideriamo un cammino del tipo

$$q_1 \to q_2 \to \cdots \to q_f \to \cdots \to q_n$$

dove q_f è finale e $q_n \to q_f$ (tale cammino esiste sempre per la Proposizione 2.4). Osserviamo che possiamo accorciare tale cammino, eliminando via via le coppie di stati che si ripetono e che si trovino entrambi prima o entrambi dopo q_f. Se il cammino così accorciato non soddisfa ancora le condizioni richieste, sarà perchè c'è uno stato q_j che occorre strettamente dopo q_f che è uguale ad uno stato q_i che occorre prima di q_f. Supponiamo che q_j sia il primo in tal senso; allora possiamo creare il palloncino

$$q_1 \to q_2 \to \cdots \to q_f \to \cdots \to q_{j-1}$$

che avrà le proprietà richieste perchè in esso non vi sono più stati ripetuti.

2.3 Chiusura Booleana

I linguaggi ω-regolari sono chiusi per operazioni booleane. La dimostrazione è molto facile per l'unione, abbastanza facile per l'intersezione e molto difficile per il complemento.

Proposizione 2.6 *Se L_1 ed L_2 sono linguaggi ω-regolari, tale è $L_1 \cup L_2$.*

Dim. Siano $A_1 = (\Sigma, Q_1, q_1, \delta_1, F_1)$ un automa che accetta L_1 e $A_2 = (\Sigma, Q_2, q_2, \delta_2, F_2)$ un automa che accetta L_2. Costruiamo un automa A che sceglie al primo passo se comportarsi come A_1 o come A_2 e che poi prosegue sempre secondo la decisione presa. Formalmente, poniamo

$$A := (\Sigma, Q_1 + Q_2 + \{q_I\}, q_I, \delta, F_1 + F_2),$$

dove $+$ indica l'unione disgiunta di insiemi e δ è così definita:

$$\begin{aligned} \delta(a,q) &= \delta_1(a,q), && \text{se } q \in Q_1 \\ \delta(a,q) &= \delta_2(a,q), && \text{se } q \in Q_2 \\ \delta(a,q) &= \delta_1(a,q_1) \cup \delta_2(a,q_2), && \text{se } q = q_I. \end{aligned}$$

Quando legge una parola infinita σ, l'automa agisce secondo un'esecuzione

$$q_I, s_1, s_2, \ldots$$

tale che $q_i, s_1, s_2, \ldots$ è un'esecuzione in A_i (per $i = 1$ oppure $i = 2$). Perciò accetta $L_1 \cup L_2$. $\dashv$

Nel caso dell'intersezione, non si può usare semplicemente la costruzione del prodotto cartesiano come per gli automi a stati finiti. Ad esempio, gli automi A_1 e A_2 illustrati nelle Figure 2 e 3 accettano entrambi il linguaggio $\{a^\omega\}$, ma il loro prodotto cartesiano (definito in modo ingenuo) non accetta $\{a^\omega\} \cap \{a^\omega\} = \{a^\omega\}$, perchè $F_1 \times F_2$ non viene mai raggiunto. C'è quindi un problema di sincronizzare la visita degli stati finali dei due automi. Si può risolvere il problema introducendo gli automi di Büchi generalizzati e poi convertendoli in automi di Büchi ordinari; qui presentiamo una costruzione diretta:

Proposizione 2.7 *Se L_1 ed L_2 sono linguaggi ω-regolari, tale è $L_1 \cap L_2$.*

Figura 2

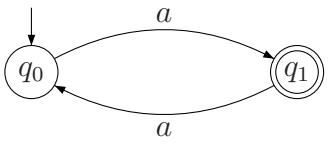
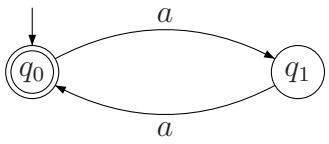


Figura 3



Dim. Siano $A_1 = (\Sigma, Q_1, q_1, \delta_1, F_1)$ un automa che accetta L_1 e $A_2 = (\Sigma, Q_2, q_2, \delta_2, F_2)$ un automa che accetta L_2. Costruiamo $A = (\Sigma, Q, q_I, \delta, F)$ che accetta $L_1 \cap L_2$ facendo due copie disgiunte del prodotto cartesiano $Q_1 \times Q_2$ (con $(s_1, s_2)^1, (s_1, s_2)^2$ indicheremo le due copie dell coppia $(s_1, s_2) \in Q_1 \times Q_2$). In dettaglio, poniamo:

$$Q := (Q_1 \times Q_2) + (Q_1 \times Q_2)$$
$$q_I := (q_1, q_2)^1$$
$$F := \{(s_1, s_2)^1 \mid s_1 \in F_1\}$$

mentre δ è definita per casi nel modo seguente:

$$\delta(a, (s_1, s_2)^1) = \{(r_1, r_2)^1 \mid r_1 \in \delta_1(a, s_1), r_2 \in \delta_2(a, s_2)\}, \quad \text{se } s_1 \notin F_1;$$
$$\delta(a, (s_1, s_2)^1) = \{(r_1, r_2)^2 \mid r_1 \in \delta_1(a, s_1), r_2 \in \delta_2(a, s_2)\}, \quad \text{se } s_1 \in F_1;$$
$$\delta(a, (s_1, s_2)^2) = \{(r_1, r_2)^2 \mid r_1 \in \delta_1(a, s_1), r_2 \in \delta_2(a, s_2)\}, \quad \text{se } s_2 \notin F_2;$$
$$\delta(a, (s_1, s_2)^2) = \{(r_1, r_2)^1 \mid r_1 \in \delta_1(a, s_1), r_2 \in \delta_2(a, s_2)\}, \quad \text{se } s_2 \in F_2.$$

In sostanza, la situazione è la seguente: l'automa, quando si trova nella componente 1, resta nella stessa componente producendo in parallelo un'esecuzione secondo A_1 e un'esecuzione secondo A_2

$$(s_1, s_2)^1 \xrightarrow{a} (r_1, r_2)^1 \xrightarrow{b} \cdots$$

finchè non raggiunge una coppia (s_1, s_2) in cui s_1 è uno stato finale di A_1. In tal caso, si posiziona sulla componente 2, finchè non raggiunge $(s_1, s_2)^2$ in cui s_2 è uno stato finale di A_2, allora torna sulla componente 1, ecc.

Un'esecuzione ρ di A su σ, si scompone quindi in due esecuzioni sincronizzate in A_1 e A_2 su σ; affinchè ρ accetti, deve visitare uno stato $(s_1, s_2)^1$ con $s_1 \in F_1$ infinite volte; per poterlo fare, deve ovviamente cambiare componente infinite volte, quindi deve passare infinite volte per una coppia $(r_1, r_2)^2$

con $r_2 \in F_2$ (questo è l'unico modo che gli è concesso per rientrare nella prima componente). Siccome F_2 è finito, dovrà visitare infinite volte la stessa coppia $(r_1, r_2)^2$ con $r_2 \in F_2$. Quindi ρ accetta sse ciascuna delle due esecuzioni parallele in cui si scompone è un'esecuzione che accetta. Questo vuol dire che il linguaggio accettato da A è proprio $L_1 \cap L_2$. $\dashv$

La chiusura dei linguaggi ω-regolari per complementazione costituisce un risultato (come si è detto) molto difficile, che ci limitiamo ad enunciare:

Teorema 2.8 *(Büchi, 1962) Se L è un linguaggio ω-regolare, tale è il suo complemento.*

Una costruzione (fra varie altre) relativamente recente ed efficiente per la complementazione è dovuta a Safra (1988): tale costruzione produce un automa con $2^{O(n \cdot log\, n)}$ stati (se n è il numero di stati dell'automa originario) ed è ottimale per il problema.[6] Per testare la soddisfacilità di formule di LTL (a differenza che per $S1S$) e per il model-checking in LTL, non avremo comunque bisogno di usare la complementazione degli automi di Büchi, esistono per fortuna metodi diretti più semplici ed efficienti.

[6]Il lettore interessato ai dettagli può consultare il testo E. Grädel, W. Thomas, T. Wilkie "Automata, Logic and Infinite Games", Springer LNCS n.2500, 2002.

3 Soddisfacibilità e Model-Checking in LTL

Fissiamo per tutto il presente paragrafo una formula ϕ di LTL contenente le variabili proposizionali $Var = \{p_1, \ldots, p_n\}$; ricordiamo che ϕ è in forma normale negativa e che non contiene i connettivi $\Diamond, R$ (si ricordi la convenzione fatta al termine del paragrafo 1.5)

Sia $\Sigma := \wp(Var)$; ad una parola infinita σ su Σ possiamo associare il modello

$$\mathcal{M}_\sigma = (\mathbb{N}, 0, s, <, \sigma)$$

che chiameremo direttamente σ (così da poter usare direttamente notazioni come $\sigma \models_t \psi$, ecc.); la corrispondenza è ovviamente biunivoca perchè ad ogni modello $\mathcal{M}_\sigma = (\mathbb{N}, 0, s, <, V)$ si può associare la parola infinita $V \in \Sigma^\omega$. Costruiremo un automa A_ϕ tale che A_ϕ accetta σ sse $\sigma \models_0 \phi$.

3.1 Insiemi di Hintikka

Con $sub(\phi)$ indichiamo le sottoformule di ϕ e con $cl(\phi)$ indichiamo l'insieme delle formule di LTL che sono:

- sottoformule di ϕ, oppure
- formule del tipo $X(\psi U \chi)$ dove $\psi U \chi$ è una sottoformula di ϕ, oppure
- formule del tipo $X\Box\psi$ dove $\Box\psi$ è una sottoformula di ϕ.

Si noti che $cl(\phi)$ è finito e ha cardinalità $O(n)$, se n è la lunghezza di ϕ.

Per costruire l'automa A_ϕ avremo bisogno della nozione di insieme di Hintikka.

Definizione 3.1 *Un insieme di Hintikka è un sottoinsieme $H \subseteq cl(\phi)$ tale che:*

(a) *per ogni $i = 1, \ldots, n$, H non contiene sia p_i che $\neg p_i$;*

(b) *se $\psi_1 \wedge \psi_2 \in H$, allora ($\psi_1 \in H$ e $\psi_2 \in H$);*

(c) *se $\psi_1 \vee \psi_2 \in H$, allora ($\psi_1 \in H$ o $\psi_2 \in H$);*

(d) *se $\psi_1 U \psi_2 \in H$, allora ($\psi_2 \in H$ o ($\psi_1 \in H$ e $X(\psi_1 U \psi_2) \in H$));*

(e) *se $\Box\psi \in H$, allora ($\psi \in H$ e $X\Box\psi \in H$).*

Infine, con $ev(\phi)$ indichiamo l'insieme delle *eventualità* di ϕ, ossia delle sottoformule di ϕ del tipo $\psi_1 U \psi_2$.

3.2 Formule e Automi

Abbiamo ora tutti gli ingredienti per definire formalmente l'automa A_ϕ; diamo però ancora qualche spiegazione informale della costruzione. L'insieme degli stati di A_ϕ sarà costitutito da coppie (s, e) dove $s \subseteq sub(\phi)$ e $e \subseteq ev(\phi)$. Il significato di una tale coppia (s, e) è il seguente: la coppia indica che ci troviamo in un istante di tempo t in cui le formule in s sono 'vere' e e è un insieme di eventualità ('vere' a loro volta)[7] che impegnano sulla struttura degli stati futuri. In altri termini, se $\psi_1 U \psi_2 \in e$ dovremo incontrare un $t' \geq t$ in cui ψ_2 è vera (con la ψ_1 che resta vera nell'intervallo $[t, t')$). Il fatto che $\psi_1 U \psi_2$ appartenga ad e è come una *promessa* da mantenere in futuro: la promessa che l'automa fa consiste nel comportarsi in modo tale da incontrare prima o poi t' con le caratteristiche indicate. La liste delle promesse e si svuota man mano che vengono mantenute e, una volta svuotata, si riempie delle promesse 'nuove' da mantenere. Non mantenere le promesse viene penalizzato alla fine con la non accettazione: questo è il senso della definizione che ci accingiamo a completare.

L'automa

$$A_\phi \;=\; (\Sigma, Q_\phi, q_I, \delta, F_\phi)$$

è così definito (ricordiamo che Σ è l'insieme dei sottoinsiemi delle variabili proposizionali $p_1, \ldots, p_n$ che occorrono in ϕ):

- Q_ϕ è il prodotto cartesiano $\wp(sub(\phi)) \times \wp(ev(\phi))$;
- q_I è lo stato $(\{\phi\}, \emptyset)$;
- F_ϕ è l'insieme degli stati del tipo $(s, \emptyset)$;
- $(s', e') \in \delta_\phi(a, (s, e))$ sse esiste un insieme di Hintikka H adeguato alla a-transizione da (s, e) a (s', e').

Abbiamo usato la seguente Definizione:

Definizione 3.2 *Dati un insieme di Hintikka H, dato $a \in \Sigma$, dati $s, s' \subseteq sub(\phi)$ ed $e, e' \subseteq ev(\phi)$, diciamo che H è adeguato alla a-transizione da (s, e) a (s', e') sse valgono le cinque condizioni seguenti:*

(i) $s \subseteq H$;

[7] In effetti, si può provare che solo gli stati (s, e) con $e \subseteq s$ vengono realmente utilizzati in un'esecuzione.

(ii) $\{p_i \mid p_i \in H\} \subseteq a$ *e* $a \cap \{p_i \mid \neg p_i \in H\} = \emptyset$*;*

(iii) $s' = \{\psi \mid X\psi \in H\}$*;*

(iv) $e' = e \backslash \{\psi_1 U \psi_2 \in ev(\phi) \mid \psi_2 \in H\}$ *(se* $e \neq \emptyset$*);*

(v) $e' = \{\psi_1 U \psi_2 \in ev(\phi) \mid \psi_1 U \psi_2 \in s'\}$ *(se* $e = \emptyset$*).*

Come si è già preannunciato, la costruzione di A_ϕ è motivata dal seguente risultato:

Teorema 3.3 *L'automa* $A_\phi = (\Sigma, Q_\phi, q_I, \delta_\phi, F_\phi)$ *accetta* σ *sse* $\sigma \models_0 \phi$*.*

Dim. Lato $\Rightarrow$: sia

$$\rho = (s_0, e_0), (s_1, e_1), \ldots$$

un'esecuzione che accetta σ e siano $H_0, H_1, \ldots$ degli insiemi di Hintikka che garantiscono che $\rho(t+1) \in \delta_\phi(\sigma(t), \rho(t))$. Proviamo, per induzione su $|\psi|$, che $\psi \in H_t$ implica $\sigma \models_t \psi$ (in particolare, siccome $q_I = \rho(0) = (s_0, e_0)$ e $\phi \in s_0 \subseteq H_0$, seguirà che $\sigma \models_0 \phi$).
Se ψ è p_i o $\neg p_i$, si usano le Definizioni 3.2(ii) e 3.1(a).
Se ψ è $\psi_1 \wedge \psi_2$ o $\psi_1 \vee \psi_2$, si usano la Definizione 3.1(b),(c) e l'ipotesi induttiva.
Se ψ è $X\psi'$, si usano la Definizione 3.2(iii), (i) e l'ipotesi induttiva nel modo seguente: da $\psi \in H_t$, segue $\psi' \in s_{t+1} \subseteq H_{t+1}$ e quindi $\sigma \models_{t+1} \psi'$, ossia $\sigma \models_t X\psi'$.
Sia ψ del tipo $\Box\psi'$; usando ripetutamente le Definizioni 3.1(e) e 3.2(iii) abbiamo $\psi' \in H_t$, $\Box\psi' \in s_{t+1} \subseteq H_{t+1}$, $\psi' \in H_{t+1}, \ldots$; questo vuol dire che vale $\psi' \in H_{t'}$ per ogni $t' \geq t$, da cui segue $\sigma \models_t \Box\psi'$.
Sia ψ del tipo $\psi_1 U \psi_2$; usando ripetutamente le Definizioni 3.1(d) e 3.2(iii) abbiamo

$$\begin{gathered}
\psi_1 U \psi_2 \in H_t \text{ e } \psi_1 \in H_t, \\
\psi_1 U \psi_2 \in s_{t+1} \subseteq H_{t+1} \text{ e } \psi_1 \in H_{t+1}, \\
\ldots, \\
\psi_1 U \psi_2 \in s_{t+j-1} \subseteq H_{t+j-1} \text{ e } \psi_1 \in H_{t+j-1}, \\
\psi_1 U \psi_2 \in s_{t+j} \subseteq H_{t+j} \text{ e } \psi_2 \in H_{t+j}
\end{gathered}$$

(per qualche $j \geq 0$) oppure

$$\psi_1 U \psi_2 \in H_t \text{ e } \psi_1 \in H_t,$$
$$\psi_1 U \psi_2 \in s_{t+1} \subseteq H_{t+1} \text{ e } \psi_1 \in H_{t+1},$$
$$\ldots,$$
$$\psi_1 U \psi_2 \in s_{t+j} \subseteq H_{t+j} \text{ e } \psi_1 \in H_{t+j},$$
$$\ldots$$

(per ogni $j \geq 0$). Nel secondo caso, possiamo liberamente supporre che valga $\psi_2 \notin H_{t+j}$ per ogni j, altrimenti si ricadrebbe nel primo caso.
Nel primo caso, per ipotesi induttiva, si ha

$$\sigma \models_t \psi_1,\ \sigma \models_{t+1} \psi_1,\ \ldots,\ \sigma \models_{t+j-1} \psi_1,\ \sigma \models_{t+j} \psi_2$$

per cui $\sigma \models_t \psi_1 U \psi_2$. Poniamoci ora nel secondo caso per provare che è impossibile. Siccome l'esecuzione è accettante, esiste $j \geq t$ tale che $e_j = \emptyset$; allora, siccome $\psi_1 U \psi_2 \in s_{j+1}$, si ha $\psi_1 U \psi_2 \in e_{j+1}$ per la Definizione 3.2(v). Ma per la Definizione 3.2(iv) allora, siccome $\psi_2 \notin H_{j+1}$, avremmo $\psi_1 U \psi_2 \in e_{j+2}$ e continuando così anche $\psi_1 U \psi_2 \in e_{j+3}$, $\psi_1 U \psi_2 \in e_{j+4}$, ecc. Ciò però contrasta col fatto che l'esecuzione accetta (quindi col fatto che l'insieme delle 'promesse' da mantenere si svuota periodicamente).

Lato $\Leftarrow$: supponiamo che valga $\sigma \models_0 \phi$ e definiamo un'esecuzione

$$\rho \ = \ (s_0, e_0), (s_1, e_1), \ldots$$

che accetta σ nel modo seguente. Sia $H_t := \{\psi \in cl(\phi) \mid \sigma \models_t \psi\}$; per induzione, poniamo

- $s_0 := \{\phi\}$ e $e_0 := \emptyset$;
- $s_{t+1} := \{\psi \mid X\psi \in H_t\}$ ed infine e_{t+1} è definito in modo da rispettare le condizioni della Definizione 3.2(iv)-(v) (cioè se $e_t \neq \emptyset$, allora $e_{t+1} := e_t \backslash \{\psi_1 U \psi_2 \in ev(\phi) \mid \psi_2 \in H_t\}$ e se $e_t = \emptyset$, allora $e_{t+1} := \{\psi_1 U \psi_2 \in ev(\phi) \mid \psi_1 U \psi_2 \in s_{t+1}\}$.

Resta da verificare che ρ così definito è davvero un'esecuzione che accetta σ. Per verificare che si tratta di un'esecuzione su σ, si deve testare che vale $(s_{t+1}, e_{t+1}) \in \delta_\phi(\sigma(t), (s_t, e_t))$ La condizione (i) della Definizione 3.2 è banale per induzione: vale per H_0 perchè $\sigma \models_0 \phi$ e vale per H_{t+1} perchè $s_{t+1} = \{\psi \mid$

$X\psi \in H_t\} \subseteq \{\psi \in cl(\phi) \mid \sigma \models_{t+1} \psi\} \subseteq H_{t+1}$. La condizione (ii) è ovvia e le (iii)-(iv)-(v) sono immediatamente riconducibili alle definizioni di cui sopra.

Per l'accettazione, è sufficiente verificare che per ogni t esiste $j \geq t$ con $e_j = \emptyset$. Osserviamo preliminarmente che $e_i \subseteq H_i$ vale per ogni i; per convincersene, basta applicare un'induzione su i: si noti che se $\psi_1 U \psi_2$ non viene cancellata passando da e_i a e_{i+1}, allora $\psi_2 \notin H_i$ per cui deve essere $X(\psi_1 U \psi_2) \in H_i$ (per la Definizione 3.1(d)) e $\psi_1 U \psi_2 \in H_{i+1}$. Fissiamo dunque t e $\psi_1 U \psi_2 \in e_t$: basta provare che esiste $k \geq t$ tale che $\psi_1 U \psi_2 \notin e_k$ (in tal modo e_i si svuota prima o poi - non si svuoterebbe solo se da un certo punto in poi restasse costante). Da $e_t \subseteq H_t$, segue $\sigma \models_t \psi_1 U \psi_2$ per definizione di H_t; quindi esiste $j \geq t$ tale che $\sigma \models_j \psi_2$ (sia j il minimo istante in cui questo succede). Segue $\psi_2 \notin H_t, \psi_2 \notin H_{t+1}, \ldots, \psi_2 \notin H_{j-1}$ e quindi $\psi_1 U \psi_2 \in e_{t+1}, \psi_1 U \psi_2 \in e_{t+2}, \ldots, \psi_1 U \psi_2 \in e_j$ per la Definizione 3.2(iv). Siccome però $\psi_2 \in H_j$, avremo $\psi_1 U \psi_2 \notin e_{j+1}$ sempre per la Definizione 3.2(iv). Quindi si ha accettazione. $\dashv$

3.3 Applicazioni

In questo paragrafo, diamo gli algoritmi per la soddisfacibilità e il model-checking in LTL.

Per la *soddisfacibilità*, grazie al Teorema 3.3 sappiamo che ϕ è soddisfacibile se A_ϕ è non vuoto, quindi possiamo applicare la Proposizione 2.4. È possibile però fare un'ulteriore ottimizzazione: si osservi che A_ϕ ha $2^{O(n)}$ stati se n è la lunghezza di ϕ, ma in realtà non c'è bisogno di costruire A_ϕ tutto insieme, basta esplorarne il pezzo che man mano interessa.

Teorema 3.4 *Il problema di decidere la soddisfacibilità di una formula ϕ di LTL è di classe PSPACE.*

Dim. Utilizziamo un risultato di complessità noto come teorema di Savitch, secondo cui NPSPACE = PSPACE: in sostanza, ci basta costruire una macchina *non deterministica* che risolve il problema usando una quantità polinomiale di risorse di memoria.

Per appurare che A_ϕ non è vuoto, basta trovare uno stato finale $f \in F_\phi$ e due cammini $q_I \to \cdots \to f$ e $f \to \cdots \to f$ dentro al grafo associato ad A_ϕ. La nostra macchina non deterministica indovina f (per far questo basta spazio $O(n \cdot log\, n)$) e poi si mette due volte in attesa di f partendo da q_I. Se sta esaminando lo stato q di A_ϕ, indovina $q' \in A_\phi$ e verifica che valga

$q \to q'$: per far questo deve indovinare H, a e verificare le cinque condizioni della Definizione 3.2. Tutto questo richiede sempre spazio $O(n \cdot log\, n)$. $\dashv$

Mettendo insieme il Teorema 3.3 e l'Osservazione 2.5 si ottiene immediatamente il seguente risultato:

Teorema 3.5 *Una formula ϕ di LTL è soddisfacibile sse esiste un palloncino*

$$q_I = q_0 \to q_1 \to \cdots \to q_n$$

dentro al grafo G_{A_ϕ}.

Dal punto di vista della stima di complessità, l'algoritmo suggerito dal Teorema 3.10 è una variante inessenziale di quello suggerito dal Teorema 3.4: di nuovo, esso può essere realizzato da una macchina non deterministica che indovini gli stati q_i e q_j del palloncino e tenti poi di raggiungerli a partire da q_I, ecc. Tuttavia l'algoritmo suggerito dal Teorema 3.10 si può considerare più praticabile (esso infatti ci fa presente che non è necessario prendere in considerazione cammini contenenti ripetizioni di stati).

Occupiamoci ora del problema del *model-checking*: dato un sistema di transizione $T = (W, w_0, R, v)$, si tratta di verificare se vale $T \models \phi$ (si veda il paragrafo 1.4 per le relative definizioni).[8] Si consideri l'automa

$$A_T \;=\; (\Sigma, W, w_0, \theta, W)$$

dove $\theta(a, u) = \{w \mid v(u) = a \;\&\; R(u, w)\}$. Siccome questo automa accetta precisamente le $\sigma \in \Sigma^\omega$ che sono esecuzioni su T, dai dati in nostro possesso otteniamo il seguente importante risultato che fornisce un algoritmo per il model-checking su LTL:

Teorema 3.6 *$T \models \phi$ sse $L(A_T) \cap L(A_{\neg\phi}) = \emptyset$.*

Si noti che l'algoritmo fornito dal teorema precedente ha complessità: (a) in spazio, logaritmica in $|T|$ e un poco più che lineare in $|\phi|$; (b) in tempo, lineare in $|T|$ ed esponenziale in $|\phi|$. Dal punto di vista dell'implementazione concreta, la realizzazione di sistemi per il model-checking e la soddisfacibilità in LTL pone problemi su cui non ci addentriamo.

[8] Nel nostro caso presente, v è una funzione il cui codominio è $\wp(Var) = \wp(\{p_1, \ldots, p_n\}) = \Sigma$.

3.4 Tableaux per LTL

Un tableau è un meccanismo che decide la soddisfacibilità di una formula ϕ analizzandone le condizioni di verità mediante un albero etichettato con insiemi di sottoformule di ϕ: la formula risulterà soddisfacibile se tale albero ha almeno un ramo aperto, cioè un ramo che non contiene contraddizioni, esplicite o implicite. La differenza fra un automa e un tableau è che, mentre il primo è un dispositivo progettato per *riconoscere* un modello (visto come una parola infinita nel caso di LTL), il secondo invece è un dispositivo progettato per *costruire* un modello (in effetti un ramo aperto di un tableau deve contenere tutte le informazioni necessarie a tale scopo).

La costruzione di tableau per LTL va incontro a problemi sottili: innanzitutto, un ramo può essere chiuso (cioè contraddittorio) non solo per un banale motivo locale dovuto alla compresenza di letterali opposti, ma anche per un motivo globale legato alle eventualità da soddisfare (le famose 'promesse da mantenere'). In aggiunta, le istruzioni necessarie per sviluppare i punti fissi rendono i rami del tableau infiniti, per cui risulta necessario (per ottenere procedure di decisione) cambiare tipo di dati ed utilizzare grafi (o alberi con 'back edges') invece degli alberi usuali. Nel seguito, preferiamo partire da un approccio *astratto* ai tableaux, onde non escludere in linea di principio anche implementazioni che ricerchino gli assegnamenti booleani richiesti dallo sviluppo del tableau mediante meccanismi di 'splitting semantico' (uniti ad euristiche opportune); la superiorità di tali meccanismi rispetto ai meccanismi di 'splitting sintattico' tradizionale è dimostrata dall'evoluzione delle tecnologie degli attuali SAT-solver basati su DPLL.

Definizione 3.7 *Data una formula ϕ, si dice* grafo di Hintikka $\mathcal{H}(\phi)$ *di ϕ il grafo avente*

- *per nodi gli insiemi di Hintikka su $cl(\phi)$;*
- *per archi le coppie (H, H') tali che $H' \supseteq \{\psi \mid X\psi \in H\}$.*

Nel seguito, scriveremo come sempre $H \rightarrow H'$ per indicare l'esistenza dell'arco che collega H con H'; con $\rightarrow^*$ indichiamo invece la chiusura riflessiva e transitiva di $\rightarrow$ (in altre parole, $H \rightarrow^* H'$ vale sse esiste un cammino che da H raggiunge H').

Chiamiamo *nodo iniziale* di $\mathcal{H}(\phi)$ un nodo H tale che $\phi \in H$. Un primo modo di usare $\mathcal{H}(\phi)$ per testare la soddisfacibilità di ϕ consiste nell'eliminazione iterata di nodi da $\mathcal{H}(\phi)$: se alla fine dell'eliminazione sopravvive

un nodo iniziale, ϕ è soddisfacibile, altrimenti no.[9] Questa procedura di eliminazione ha il difetto di essere poco efficiente perchè richiede un'analisi *globale* di $\mathcal{H}(\phi)$, la cui dimensione è esponenziale (in effetti, la procedura di eliminazione degli insiemi di Hintikka viene indicata come algoritmo di riferimento - almeno a livello teorico - per altri problemi di logica modale e temporale che sono EXPTIME-completi e non solo PSPACE-completi come la soddisfacibilità in LTL). In analogia a quanto visto nel paragrafo precedente per i metodi di soddisfacibilità basati su automi, cercheremo anche qui di utilizzare metodi *locali* di esplorazione di $\mathcal{H}(\phi)$.

Una *componente fortemente connessa* ('strongly connected component', abbreviato con scc) di $\mathcal{H}(\phi)$ è una classe di equivalenza rispetto alla relazione di equivalenza

$$H \approx H' \quad : \text{sse} \quad H \rightarrow^* H' \rightarrow^* H.$$

Una scc C è *isolata* se e solo se $C = \{H\}$ e $H \not\rightarrow H$, ossia se e solo se C è ridotta ad un punto solo che per di più non ha un arco rivolto verso se stesso.

Definizione 3.8 *Una scc C è* buona *sse non è isolata e per ogni $H \in C$, per ogni $\psi_1 U \psi_2 \in H$ esiste $H' \in C$ tale che $\psi_2 \in H'$.*

La definizione precedente sarà utile per trattare il problema della verità delle eventualità nei modelli di Kripke. Un fatto semplice ma essenziale di tale definizione è sintetizzato dal Lemma seguente:

Lemma 3.9 *Una scc C è buona sse non è isolata ed* esiste *$H \in C$ tale che per ogni $\psi_1 U \psi_2 \in H$ esiste $H' \in C$ tale che $\psi_2 \in H'$.*

Dim. In altre parole, per testare che una scc è buona basta considerare le eventualità di *uno* qualsiasi dei suoi elementi (e non di *tutti* i suoi elementi). Per verificarlo, sia C una scc non isolata e sia $H \in C$ tale che per ogni $\psi_1 U \psi_2 \in H$ esiste $H' \in C$ tale che $\psi_2 \in H'$. Verichiamo che la stessa proprietà vale per un altro qualsiasi $\tilde{H} \in C$: a tale scopo, fissiamo $\psi_1 U \psi_2 \in \tilde{H}$. Per definizione di scc, esiste un cammino (che per la definizione di scc è tutto interno a C)

$$\tilde{H} = H_0 \rightarrow H_1 \rightarrow \cdots\cdots \rightarrow H_k = H.$$

[9] In dettaglio, in ogni ciclo di eliminazione vengono cancellati: (a) i nodi H per cui non esiste H' tale che $H \rightarrow H'$; (b) i nodi H per cui esiste $\psi_1 U \psi_2 \in H$ con $\psi_2 \notin H'$ per nessun H' tale che $H \rightarrow^* H'$.

Dobbiamo trovare $H' \in C$ con $\psi_2 \in H'$; l'esistenza di tale H' segue dal fatto che o $\psi_2 \in H_i$ per qualche $i \leq k$ oppure $\psi_1 U \psi_2 \in H$ (si ricordi la Definizione 3.1(d)). $\dashv$

Il seguente Teorema costituisce il fondamento teorico (di validità e completezza) per procedure di decisione basate su tableaux:

Teorema 3.10 *Una formula ϕ è soddisfacibile sse esiste un cammino*

$$H_0 \rightarrow H_1 \rightarrow \cdots\cdots \rightarrow H_k$$

interno ad $\mathcal{H}(\phi)$ tale che: (i) H_0 è un nodo iniziale e (ii) la scc cui appartiene H_k è buona.

Dim. Sia ϕ soddisfacibile nel modello di Kripke $\mathcal{M} = (\mathbb{N}, 0, s, <, V)$; consideriamo la successione degli insiemi di Hintikka $H_0, H_1, H_2, \ldots$, dove

$$H_i := \{\psi \in cl(\phi) \mid \mathcal{M} \models_i \psi\}.$$

Abbiamo ovviamente un cammino infinito $H_0 \rightarrow H_1 \rightarrow H_2 \rightarrow \cdots$ dentro a $\mathcal{H}(\phi)$. Sia H_k il primo insieme di Hintikka che si ripete infinite volte in tale cammino e dimostriamo (usando il Lemma 3.9) che la scc cui appartiene H_k è buona. In effetti, se H_k si ripete infinite volte, ciò significa che $\{H_k, H_{k+1}, H_{k+2}, \ldots\}$ appartengono tutti alla stessa scc; inoltre se $\psi_1 U \psi_2 \in H_k$, allora $\mathcal{M} \models_k \psi_1 U \psi_2$ ed esiste $j \geq k$ tale che $\mathcal{M} \models_j \psi_2$, ossia tale che $\psi_2 \in H_j$. Questo vuol dire che la scc cui appartiene H_k è buona.

Viceversa, si supponga che esista un cammino $H_0 \rightarrow H_1 \rightarrow \cdots \rightarrow H_k$ tale che $\phi \in H_0$ e tale che la scc di H_k sia buona. Possiamo costruire un altro cammino

$$H_k \rightarrow \cdots\cdots \rightarrow H_{k+s} \rightarrow H_k$$

in modo tale che la lista $(H_k, \ldots, H_{k+s})$ contenga (con eventuali ripetizioni) tutti e soli i nodi della scc di H_k. Estendiamo il cammino composto

$$H_0 \rightarrow H_1 \rightarrow \cdots \rightarrow H_k \rightarrow \cdots \rightarrow H_{k+s}$$

ad un cammino infinito dentro a $\mathcal{H}(\phi)$ ripetendo ciclicamente gli ultimi stati $H_k, \ldots, H_{k+s}$.[10]

[10] Formalmente, si può fare così. Ricordiamo che ogni intero i o è minore di k, oppure si rappresenta in modo unico nella forma $i = k + n(s+1) + j$ con $n \geq 0$ e $j < s+1$ (basta usare quoziente e resto della divisione di $i - k$ per $s+1$). Dunque si può porre $H_{k+n(s+1)+j} := H_{k+j}$.

Costruiamo il modello di Kripke $\mathcal{M} = (\mathbb{N}, 0, s, <, V)$ ponendo $V(i) := \{p \in Var \mid p \in H_i\}$. Si tratta ora di provare per induzione sulla lunghezza di $\psi \in cl(\phi)$ che per ogni i vale

$$\psi \in H_i \quad \Rightarrow \quad \mathcal{M} \models_i \psi \tag{11}$$

(siccome H_0 è iniziale, seguirà $\mathcal{M} \models_0 \phi$ come si voleva).

La condizione (11) è ovvia se ψ è un letterale o è del tipo $\psi_1 \wedge \psi_2, \psi_1 \vee \psi_2, X\chi$. Se ψ è del tipo $\Box\chi$, allora da $\Box\chi \in H_i$ segue $\chi \in H_j$ per ogni $j \geq i$ (per la Definizione 3.1(e)); ma allora per ipotesi induttiva abbiamo $\mathcal{M} \models_j \chi$ per ogni $j \geq i$, ossia $\mathcal{M} \models_i \Box\chi$.

Resta dunque solo il caso in cui ψ è del tipo $\psi_1 U \psi_2$. Distinguiamo il Caso (i) in cui si ha $i < k$ dal Caso (ii) in cui vale $i \geq k$; facciamo vedere che il Caso (i) si riduce al Caso (ii). Infatti se $i < k$, allora (per la Definizione 3.1(d)) abbiamo due sottocasi: (i.1) $\psi_1 U \psi_2 \in H_k$ e $\psi_1 \in H_j$ per $i \leq j < k$; (i.2) esiste $j < k$ con $\psi_2 \in H_j$ e con ψ_1 che appartiene a tutti gli H_r compresi fra i e $j-1$ inclusi. Nel sottocaso (i.2) abbiamo $\mathcal{M} \models_i \psi_1 U \psi_2$ per ipotesi induttiva; anche nel sottocaso (i.1) possiamo concludere $\mathcal{M} \models_i \psi_1 U \psi_2$ dall'ipotesi induttiva e dal fatto che $\mathcal{M}_k \models_k \psi_1 U \psi_2$ vale per il Caso (ii).

Resta quindi da provare il Caso (ii) in cui $k \geq i$. Siccome la scc C cui H_k appartiene è buona, esiste un $H \in C$ tale che $\psi_2 \in H$; tale H occorre nella lista $(H_k, \ldots, H_{k+s})$ e quindi (essendo tale lista ripetuta ciclicamente dal posto k in poi), esisterà un $j \geq i$ con $\psi_2 \in H_j$. Possiamo scegliere j minimo con tale proprietà; ma allora (per la Definizione 3.1(d)) abbiamo che ψ_1 appartiene a tutti gli H_r compresi fra i e $j-1$ inclusi. Da questo si ricava per ipotesi induttiva che vale $\mathcal{M} \models_i \psi_1 U \psi_2$. $\dashv$

Osserviamo che anche l'algoritmo del Teorema 3.10 ha la stessa complessità PSPACE degli algoritmi delle precedenti procedure di decisione per la soddisfacibilità basate su automi.

Osservazione 3.11 Tableaux e procedure di decisione similari utilizzano in realtà non proprio $\mathcal{H}(\phi)$ ma un suo sottografo:[11] questo perchè alcuni insiemi di Hintikka possono essere ottenuti combinando in modo molto disparato formule di $cl(\phi)$, al punto tale da risultare completamente ininfluenti. Il sottografo di $\mathcal{H}(\phi)$ cui si sceglie di limitarsi deve

[11] In queste note, per sottografo di un grafo $G = (N, E)$ intendiamo un grafo $G' = (N', E')$ ottenuto da G rimuovendo alcuni dei nodi in N come pure eventualmente anche alcuni degli archi in E. In altre parole, abbiamo $N' \subseteq N$ e $E' \subseteq E$; nel seguito, diremo che un arco o un nodo di G appartiene a G' per indicare che tale arco o nodo sta in N' o in E', rispettivamente.

però soddisfare alcune buone proprietà; a titolo di esempio, segnaliamo che ai sottografi G di $\mathcal{H}(\phi)$ che soddisfano le due proprietà seguenti si estende banalmente il Teorema 3.10 (come si può vedere ripercorrendo la relativa dimostrazione):[12]

(i) per ogni nodo iniziale H' di $\mathcal{H}(\phi)$, esiste un altro nodo iniziale $\tilde{H} \in G$ tale che $\tilde{H} \subseteq H'$;

(ii) se $H \in G$ e $H \to H'$ è un arco di $\mathcal{H}(\phi)$, esistono un nodo $\tilde{H}$ ed un arco $H \to \tilde{H}$, entrambi in G, con le seguenti proprietà: (a) $\tilde{H} \subseteq H'$; (b) se $X(\psi_1 U \psi_2) \in H$ e $\psi_2 \in H'$, allora $\psi_2 \in \tilde{H}$.

Osservazione 3.12 Esponiamo ora un calcolo a tableaux per LTL. Un *tableau per* ϕ è un albero i cui nodi sono etichettati con insiemi finiti di formule di $cl(\phi)$; la radice è etichettata con la sola formula ϕ. Le formule possono essere o meno asteriscate, a seconda che siano già state analizzate o no; per analizzare una formula, la si asterisca e si aggiungono ad ogni foglia sotto il nodo in cui la formula occorre,[13] uno o più nodi come indicato dalle regole α e β che seguono (avremo un solo nodo etichettato con due formule per le regole α e due nodi etichettati con una o due formule per le regole β).

(i) *regole* α:

$$\frac{\psi_1 \wedge \psi_2}{\psi_1, \psi_2}$$

$$\frac{\Box\psi}{X\Box\psi, \psi}$$

(ii) *regole* β:

$$\frac{\psi_1 \vee \psi_2}{\psi_1 \quad \psi_2}$$

$$\frac{\psi_1 U \psi_2}{\psi_2 \quad \psi_1,\ X(\psi_1 U \psi_2)}$$

Quando non è più possibile applicare nè regole α nè regole β, si applica la *regola dello stato successivo*. Tale regola opera nel modo seguente. Siano

$$L_1, \ldots, L_k, X\psi_1, \ldots, X\psi_n$$

le formule non asteriscate occorrenti in un ramo (cui ripetiamo non devono essere applicabili regole α-β) sotto la precedente applicazione di una regola dello stato successivo (se ne è già stata applicata una). Se fra $L_1, \ldots, L_k$ ci sono letterali contraddittori (cioè del

[12] Osserviamo che sono necessarie solo piccole variazioni nel lato 'validità' di tale dimostrazione.

[13] Questa è la procedura comunemente indicata nella letteratura, ma non è la più efficiente (rappresenta in effetti una dispendiosa esplorazione in ampiezza.)

tipo p e $\neg p$), il ramo è dichiarato *chiuso* e viene di fatto ignorato; altrimenti la regola delle stato successivo produce un nuovo nodo etichettato con

$$\psi_1, \ldots, \psi_n.$$

Siccome l'albero che viene costruito in questo modo è infinito, cambiamo tipo di dati, usiamo un grafo e ammettiamo la possibilità che la regola di stato successivo crei un arco verso un nodo identicamente etichettato prodotto in precedenza.

Osserviamo che l'insieme delle formule che si trovano su un ramo compreso fra due applicazioni di regole dello stato successivo (o fra la radice e la prima applicazione di regola dello stato successivo) costituisce un insieme di Hintikka. Se colleghiamo con un arco gli insiemi di Hintikka corrispondenti a due applicazioni consecutive della regola dello stato successivo, è facile vedere che il grafo costruito in questo modo soddisfa le condizioni della Osservazione 3.12, per cui ad esso si applica il Teorema 3.10: ϕ è soddisfacibile sse si raggiunge in tale grafo una scc buona partendo da un nodo iniziale.

Osservazione 3.13 È possibile utilizzare il formalismo dei tableaux anche per il model-checking. Data una formula ϕ e dato un sistema di transizione $T = (W, w_0, R, v)$, definiamo il grafo $\mathcal{H}(\phi) \otimes T$ nel modo seguente: (i) per nodi consideriamo le coppie $(H, w) \in \mathcal{H}(\phi) \times W$ tali che $\{p \in Var \mid p \in H\} \subseteq \{p \in Var \mid p \in v(w)\}$ e $\{p \in Var \mid \neg p \in H\} \cap \{p \in Var \mid p \in v(w)\} = \emptyset$; (ii) c'è un arco $(W, w) \to (H', w')$ in $\mathcal{H}(\phi) \otimes T$ se e solo se vale $R(w, w')$ e c'è un arco $H \to H'$ in $\mathcal{H}(\phi)$. Definendo ora nel modo ovvio la nozione di scc buona e di nodo iniziale,[14] si può provare (ripetendo gli argomenti della dimostrazione del Teorema 3.10) che vale $T \models \phi$ se e solo se non esiste un cammino in $\mathcal{H}(\neg\phi) \otimes T$ che parte da un nodo iniziale e raggiunge una scc buona.

[14] La definizione di scc e di scc isolata sono generali (nel senso che si applicano a grafi diretti qualunque). Per definire invece 'nodo iniziale' e 'scc buona', si dovrà far riferimento alla prima componente, riutilizzando le definizioni date per $\mathcal{H}(\phi)$.

4 La Logica Monadica del Secondo Ordine

Esistono vari formalismi per la specifica di proprietà di sistemi di transizione (CTL, CTL^*, μ-calcolo). Restando nel caso del flusso di tempo lineare e discreto, studiamo un formalismo molto espressivo ma pur sempre decidibile: la logica monadica del secondo ordine su $\mathbb{N}$, chiamata $S1S$.

4.1 Sintassi di S1S

In $S1S$ sono presenti due tipi di variabili: le variabili per istanti di tempo (che indichiamo con $x, y, \dots$) e le variabili per sottoinsiemi (che indichiamo con $P, Q, \dots$). Utilizziamo in aggiunta due relazioni binarie fra istanti di tempo, cioè l'uguaglianza e la relazione 'essere l'istante successivo di'. Le formule di $S1S$ saranno quindi definite ricorsivamente così:

- se x, y sono variabili individuali, $x = y$ e $s(x, y)$ sono formule;
- se x è una variabile individuale e P una variabile per sottoinsiemi, $P(x)$ è una formula;
- se ϕ, ψ sono formule, tali sono $(\neg\phi), (\phi \wedge \psi), (\phi \vee \psi)$;
- se ϕ è una formula e x una variabile individuale, $(\exists x\phi)$ è una formula;
- se ϕ è una formula e P una variabile per sottoinsiemi, $(\exists P\phi)$ è una formula.

$S1S$ è molto espressiva, come si può vedere dalla seguente lista di abbreviazioni che useremo:[15]

[15] Valgono sempre le solite abbreviazioni proposizionali per $\top, \bot, \phi \rightarrow \psi, \phi \leftrightarrow \psi$, ecc. In aggiunta, $\forall$ è abbreviazione di $\neg\exists\neg$.

Abbreviazione	Definizione
$P \subseteq Q$	$\forall x\ (P(x) \rightarrow Q(x))$
$P = Q$	$P \subseteq Q \wedge Q \subseteq P$
$Empty(P)$	$\forall Q\ (P \subseteq Q)$
$Sing(P)$	$\forall Q\ (Q \subseteq P \rightarrow Q = P \vee Empty(Q))$
$S(P,Q)$	$\forall x \forall y\ (P(x) \wedge s(x,y) \rightarrow Q(y))$
$Up(P)$	$\forall x \forall y\ (P(x) \wedge s(x,y) \rightarrow P(y))$
$x \leq y$	$\forall P\ (Up(P) \wedge P(x) \rightarrow P(y))$
$x < y$	$x \leq y \wedge \neg x = y$
$zero(x)$	$\neg \exists y\, s(y,x)$
$Infinite(P)$	$\forall x\ (P(x) \rightarrow \exists y\, (x < y \wedge P(y)))$

Proveremo che $S1S$ è decidibile per la soddisfacibilità (di fatto, $S1S$ è la esatta controparte logica degli automi di Büchi). Per semplificarci il compito, eliminiamo le variabili individuali da $S1S$ dandone una riformulazione.

Nella nuova formulazione, usiamo le relazioni binarie fra sottoinsiemi $\subseteq$ e S come primitive, per cui le formule saranno definite come segue:

- se P, Q sono variabili per sottoinsiemi, $P \subseteq Q$ e $S(P,Q)$ sono formule;

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

- se ϕ è una formula e P una variabile per sottoinsiemi, $(\exists P \phi)$ è una formula.

Le vecchie formule si ottengono come indicato nella seguente tabella (si osservi che per definire $Sing(P)$ basta definire prima $Empty(Q)$ e $P = Q$, le quali si definiscono a loro volta a partire da $P \subseteq Q$ come indicato nella tabella precedente):

Vecchia Formula	Ridefinizione
$x = y$	$Sing(x) \wedge Sing(y) \wedge x \subseteq y$
$s(x, y)$	$Sing(x) \wedge Sing(y) \wedge S(x, y)$
$P(x)$	$Sing(x) \wedge x \subseteq P$
$\exists x \phi$	$\exists x(Sing(x) \wedge \phi)$

4.2 Immagini Dirette ed Inverse

Prima di poter continuare, ci servono alcune informazioni complementari sugli automi di Büchi.

Fissiamo una funzione $f : \Sigma_0 \longrightarrow \Sigma$ e un automa di Büchi

$$A \ = \ (\Sigma, Q, q_I, \delta, F)$$

che lavora sull'alfabeto Σ; definiamo un altro automa che lavora sull'alfabeto Σ_0 mediante

$$f^*(A) \ = \ (\Sigma_0, Q, q_I, \delta^*, F)$$

dove si è posto

$$\delta^*(a, q) := \delta(f(a), q).$$

La relazione fra A e $f^*(A)$ è spiegata nel seguente

Lemma 4.1 *$f^*(A)$ accetta la parola*

$$\sigma = a_0, a_1, \ldots$$

di Σ_0^ω sse A accetta la parola

$$f(\sigma) := f(a_0), f(a_1), \ldots$$

di Σ^ω.

Dim. Basta osservare che

$$\rho = \rho(0), \rho(1), \rho(2), \ldots$$

è un'esecuzione di A relativamente a $f(\sigma)$ sse lo è di $f^*(A)$ relativamente a σ. Infatti ρ è un'esecuzione di A relativamente a $f(\sigma)$ sse per ogni i

$$\rho(i+1) \in \delta(f(a_i), \rho(i))$$

e ρ è un'esecuzione di $f^*(A)$ relativamente a σ sse per ogni i

$$\rho(i+1) \in \delta^*(a_i, \rho(i)).$$

Le due condizioni coincidono, stante la definizione di δ^*. $\dashv$

Data ancora una funzione $f : \Sigma_0 \longrightarrow \Sigma$, descriviamo ora una costruzione che opera in senso opposto; sia dato un automa

$$B \;=\; (\Sigma_0, Q, q_I, \delta, F)$$

che lavora sull'alfabeto Σ_0; definiamo un altro automa che lavora sull'alfabeto Σ mediante

$$f(B) \;=\; (\Sigma, Q, q_I, \delta_f, F)$$

dove si è posto

$$r \in \delta_f(a, q) \quad \text{sse} \quad \exists\, a' \; (f(a') = a \;\&\; r \in \delta(a', q)).$$

La relazione fra B e $f(B)$ è spiegata nel seguente

Lemma 4.2 *$f(B)$ accetta la parola $\sigma \in \Sigma^\omega$ sse esiste $\sigma_0 \in \Sigma_0^\omega$ tale che B accetta σ_0 e $f(\sigma_0) = \sigma$.*

Dim. Si supponga che ρ sia un'esecuzione di $f(B)$ che accetta

$$\sigma = a_0, a_1, a_2 \cdots .$$

Allora per ogni i, $\rho(i+1) \in \delta_f(a_i, \rho(i))$, cioè esiste $a_i' \in \Sigma_0$ con $f(a_i') = a_i$ e $\rho(i+1) \in \delta(a_i', \rho(i))$. Ponendo σ_0 uguale alla parola infinita formata dalle a_i' così raccolte, si ha che ρ è un'esecuzione di B che accetta σ_0.

Viceversa, sia

$$\sigma_0 = a_0', a_1', a_2' \cdots$$

tale che $f(\sigma_0) = \sigma$ e sia ρ un'esecuzione di B che accetta σ_0. Allora, per ogni i, $\rho(i+1) \in \delta_f(a_i, \rho(i))$, da cui si ricava che ρ è un'esecuzione di $f(B)$ che accetta σ. $\dashv$

4.3 Decidibilità tramite Automi

Torniamo ora a $S1S$ (che abbiamo riformulato usando solo le variabili del secondo ordine). Siano $P_1, \ldots, P_n$ variabili per sottoinsiemi; un *assegnamento* è una funzione σ che fa loro corrispondere dei veri sottoinsiemi $\sigma(P_k) \subseteq \mathbb{N}$ $(k = 1, \ldots, n)$. Equivalentemente, posto

$$\Sigma := \wp(\{P_1, \ldots, P_n\})$$

un assegnamento può essere visto come una parola infinita $a_0, a_1, \ldots$ sull'alfabeto Σ: in tale corispondenza, a_i è definito come l'insieme dei $P_k \in \{P_1, \ldots, P_n\}$ tali che $i \in \sigma(P_k)$. Questo ci autorizza a confondere assegnamenti con parole su Σ^ω, esattamente come nel caso di LTL avevamo confuso parole su Σ^ω e modelli. Così ci prenderemo la libertà di dire che una formula ϕ di $S1S$ contenente al più le variabili libere $P_1, \ldots, P_n$ è *vera* nell'assegnamento/parola $\sigma \in \Sigma^\omega$ (dove $\Sigma := \wp(\{P_1, \ldots, P_n\})$) e riserveremo la notazione $\sigma \models \phi$ per indicare tale concetto. In questo modo riotteniamo un teorema di trasformazione di formule in automi come per LTL:

Teorema 4.3 *Data una formula ϕ di $S1S$, è possibile associarle un automa di Büchi A_ϕ tale che ϕ è soddisfacibile in $\mathbb{N}$ sse $L(A_\phi) \neq \emptyset$.*

Dim. La costruzione è per induzione su ϕ; l'alfabeto di A_ϕ sarà sempre $\Sigma := \wp(\{P_1, \ldots, P_n\})$, dove $P_1, \ldots, P_n$ sono le variabili che occorrono libere in ϕ. Costruiamo A_ϕ in modo tale che per ogni $\sigma \in \Sigma^\omega$

$$A_\phi \textit{ accetta } \sigma \qquad \text{sse} \qquad \sigma \models \phi$$

(si ricordi quanto sopra detto circa assegnamenti e parole di Σ^ω).

Se ϕ è atomica, ϕ è del tipo $P \subseteq Q$ o $S(P, Q)$: i relativi automi sono indicati nelle Figure 4 e 5.[16]

Supponiamo che ϕ sia del tipo $\exists P\psi$. Per induzione, è disponibile un automa A_ψ (che lavora sull'alfabeto $\Sigma_0 = \wp(\{P, P_1, \ldots, P_n\})$) con le caratteristiche richieste rispetto a ψ. Sia $\Sigma = \wp(\{P_1, \ldots, P_n\})$ e sia $f : \Sigma_0 \longrightarrow \Sigma$ la funzione che ad $a \subseteq \{P, P_1, \ldots, P_n\}$ associa $a \cap \{P_1, \ldots, P_n\}$ (cioè f cancella P se c'era). Costruiamo l'automa $f(A_\psi)$: per il Lemma 4.2 tale automa

[16] Si noti che quando l'automa della Figura 4 legge un $a \in \Sigma$ in cui P è vero ma Q no (cioè in cui $P \in a$ e $Q \notin a$) non riesce più a continuare - quindi non accetta. Analogamente succede all'automa della Figura 5 se Q non diventa vera immediatamente dopo P.

Figura 4

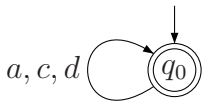
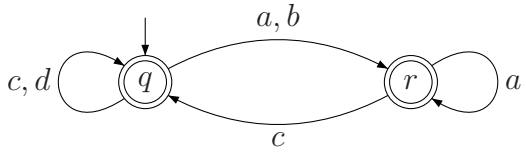


Figura 5



Legenda: $a = \{P, Q\}$, $b = \{P\}$, $c = \{Q\}$, $d = \emptyset$.

accetta σ sse esiste σ_0 accettata da A_ψ con $f(\sigma_0(i)) = \sigma(i)$ per ogni i. La σ_0 corrisponde ad un assegnamento che estende σ perchè assegna un valore anche a P, segnatamente a P viene assegnato il sottoinsieme di $\mathbb{N}$ costituito dagli i per cui $P \in \sigma_0(i)$. Quindi $f(A_\psi)$ accetta σ sse $\sigma \models \psi$.

Se ϕ è del tipo $\neg\psi$, si prende come A_ϕ un automa che accetta il linguaggio complementare di $L(A_\psi)$. Lo stesso si fa nel caso della congiunzione e della disgiunzione (relativamente ad intersezione e complemento, cfr. il paragrafo 2.3). C'è però un piccolo dettaglio di cui tener conto: se ϕ è ad esempio $\psi_1 \wedge \psi_2$, non è detto che le variabili che occorrono libere in ϕ occorrano libere in entrambe le ψ_1, ψ_2: qui serve la costruzione del Lemma 4.1. Sia, ad esempio, $f : \Sigma_0 \longrightarrow \Sigma$ la funzione che 'cancella P' che abbiamo visto sopra (qui $\Sigma = \wp(\{P_1, \ldots, P_n\})$ e $\Sigma_0 = \wp(\{P, P_1, \ldots, P_n\})$); sia anche A_{ψ_i} un automa che accetta esattamente gli assegnamenti che soddisfano ψ_i $(i = 1, 2)$. Dal Lemma 4.1 risulta immediatamente che $f^*(A_{\psi_i})$ accetta i $\{P, P_1, \ldots, P_n\}$-assegnamenti il cui $\{P_1, \ldots, P_n\}$-ridotto soddisfa ψ_i. Tenendo presente tutto questo, è possibile definire $A_{\psi_1 \wedge \psi_2}$ e $A_{\psi_1 \vee \psi_2}$ in termini di intersezione e unione dei linguaggi accettati da A_{ψ_1} e A_{ψ_2}: basta semplicemente applicare ripetutamente delle f^* prima di procedere con le operazioni booleane. $\dashv$

Il Teorema 4.3 dà una procedura di decisione per la soddisfacibilità in $S1S$: per sapere se ϕ è soddisfacibile, si testa se il linguaggio accettato da A_ϕ è vuoto. Questa procedura è però difficilmente praticabile nei casi concreti perchè ogni volta che si incontra una negazione nel costruire A_ϕ, il numero degli stati passa da n a $2^{O(n \cdot log\, n)}$ (tecnicamente, questo vuol dire che la procedura *non è elementare*). Per questo motivo, è utile per classi di formule più ristrette, esplorare dei metodi alternativi più diretti, come abbiamo fatto per LTL.

Gli automi di Büchi e $S1S$ sono formalismi equivalenti: possiamo infatti 'invertire' il Teorema 4.3 *facendo corrispondere ad ogni automa di Büchi* $A = (\Sigma, Q, q_I, \delta, F)$ *una formula* ϕ_A *di* $S1S$ *che è soddisfatta esattamente dagli assegnamenti accettati da* A.

Per definire ϕ_A usiamo variabili libere del secondo ordine P_a ($a \in \Sigma$) e variabili vincolate sempre del secondo ordine R_q ($q \in Q$). Supponendo che sia $Q = \{q_1, \ldots, q_n\}$, la formula ϕ_A è data da

$$\phi_A \ =: \ \exists R_{q_1} \cdots \exists R_{q_n} \, (Part \wedge Init \wedge Trans \wedge Accept)$$

dove abbiamo usato le seguenti abbreviazioni:

- $Part$ esprime il fatto che $R_{q_1}, \ldots, R_{q_n}$ formano una partizione e quindi sta per $\forall x \, (\bigvee_i R_{q_i}(x) \wedge \bigwedge_{i \neq j} \neg (R_{q_i}(x) \wedge R_{q_j}(x)))$;
- $Init$ esprime la condizione di inizializzazione e quindi, supponendo $q_I = q_1$, sta per $\exists x (zero(x) \wedge R_{q_1}(x))$;
- $Trans$ esprime la condizione di transizione e quindi sta per la congiunzione delle seguenti formule (al variare di $i = 1, \ldots, n$ e di $a \in \Sigma$) $\forall x \forall y \, (s(x,y) \wedge R_{q_i}(x) \wedge P_a(x) \rightarrow \bigvee_{q_j \in \delta(a,q_i)} R_{q_j}(y))$;
- *Accept* esprime la condizione di accettazione negli automi di Büchi, ossia sta per $\bigvee_{q_i \in F} Infinite(R_{q_i})$.

È immediato verificare che per ogni σ, si ha $\sigma \models \phi_A$ se e solo se esiste un'esecuzione di A che accetta σ.