

Lezione Calcolabilità e Complessità A.A. 2010/2011

Alfonso Napolitano N97/115, Vincenzo Reggina N97/88

1 Le logiche temporali lineari

In questa lezione verrà trattata la Logica Temporale Lineare. Tale logica è sostanzialmente un'estensione della logica proposizionale, necessaria per caratterizzare aspetti dinamici spesso utilizzata nella verifica formale.

1.1 Modellazione dei sistemi e strutture di Kripke

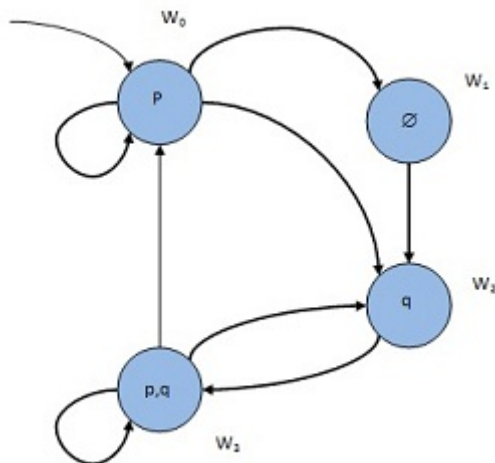
Una struttura di Kripke è una macchina a stati finiti deterministica, ideata da Saul Kripke, utilizzata spesso nel Model Checking per rappresentare un comportamento di un sistema. Si tratta essenzialmente di un grafo i cui i nodi rappresentano gli stati del sistema, e gli archi rappresentano le transizioni di stato. Una funzione di etichettatura associa ciascun nodo una serie di proprietà corrispondente allo stato.

Definiamo più formalmente di cosa si tratta:

K kripke è definita come una tupla $=\langle AP, W, R, L, W_0 \rangle$ dove:

- AP sono le atomic propositions
- W è l'insieme dei mondi
- $R \subseteq W \times W$ (relazione di transizione) è definita sul prodotto cartesiano dei mondi
- W_0 è un sottoinsieme di W
- $L: W \rightarrow 2^{AP}$ è la funzione di labellatura.

Proponiamo adesso un semplice esempio:



Nota: in particolare ci soffermeremo solo su strutture che presentano un numero di nodi finito (quindi macchine a stati finiti).

Mostriamo ora quale è il linguaggio accettato dalla Kripke Structure: $L(K) \subseteq (2^{AP})^w$ non è altro che un sottoinsieme di tutte le possibili parole infinite

Definiamo un Path

$$\text{Path}(K) \subseteq W^w$$

Dove:

$$\pi \in \text{Path}(K) \text{ sse } \forall i \in \mathbb{N} (\pi_i, \pi_{i+1}) \in R$$

Dove \mathbb{N} è l'insieme dei numeri naturali ed R è l'insieme delle transizioni.

Quindi π è un Path sull'automata K solo e soltanto se per ogni i esiste una funzione di transizione che va da π_i al suo successivo π_{i+1} (Sono in relazione con R). Sostanzialmente non si passa da uno stato ad un altro senza che vi sia un arco.

Quindi il linguaggio accettato è:

$$L(K) \stackrel{def}{=} \{a \in (2^{AP})^w \text{ tale che } \exists \pi \in \text{Path}(K) \forall i \in \mathbb{N} a_i = L(\pi_i)\}$$

Sostanzialmente una parola è una "labellatura" della struttura di kripke su un percorso infinito, quindi se esiste un percorso con tale labellatura allora questa parola appartiene al linguaggio accettato.

1.2 Trasformazione da automi di Kripke in Buchi

Mostriamo ora come è possibile trasformare una struttura di Kripke in un automa di Buchi.

Definiamo questa trasformazione :

$$K = \langle AP, W, R, L \rangle \rightarrow N = \langle 2^{AP}, W, \delta, \{W_0\}, W \rangle$$

Dove:

- N è un automa di Buchi che accetta lo stesso linguaggio di K (Kripke).
- L'insieme di possibili input (2^{AP}) è dato dalle possibili atomic proposition che possono essere vere
- Gli stati dell'automata di Buchi sono i mondi della Kripke Structure.
- Gli stati iniziali sono i mondi iniziali di K

- Come accettazione abbiamo Σ^*

La funzione di transizione è riportata di seguito:

$$\delta(q, \sigma) = \begin{cases} \{q' \in W \text{ tale che } (q, q') \in R\} & \text{se } \sigma \models L(q) \\ \emptyset & \text{altrimenti} \end{cases}$$

La funzione di transizione si comporta nel seguente modo:

- Dato uno stato q , la funzione di transizione restituisce tutti i mondi q' successori di q secondo la R solo se quello che sto leggendo è la labellatura che nella Kripke Structure era stata assegnata allo stato q .
- Altrimenti viene restituito l'insieme vuoto.

Tutte le computazioni delle parole sono accettate se sono delle sequenze di informazioni che formano un run coerente con l'automa e con la relazione di transizione.

2 LTL: Logica Temporale Lineare

In questo paragrafo introduciamo la logica lineare, ossia una logica temporale in cui in ogni istante di tempo ha un unico istante successivo.

In particolare tratteremo le logiche temporali discrete, cioè quelle logiche in cui il tempo è discretizzato.

2.1 Sintassi LTL

La sintassi LTL può essere vista come un'estensione della logica proposizionale classica. In particolare è così strutturata: Insieme AP di proposizioni atomiche $AP = \{p_1, \dots, p_n\}$.

Ogni proposizione atomica $p \in AP$ è una formula LTL.

Definite $\varphi_1 \varphi_2 \varphi_3$ formule LTL sono ancora formule LTL la loro unione, negazione e disgiunzione

- $\neg\varphi_1$ Negazione
- $\varphi_1 \wedge \varphi_2$ Disgiunzione
- $\varphi_1 \vee \varphi_2$ Congiunzione

Oltre a queste descritte, si definiscono altri tre operatori logici il Next, l'Until e il Release (estensione)

- $X\varphi_1$ Next: significa che nel successivo istante è verificata φ_1
- $\varphi_1 U \varphi_2$ Until: significa che esiste un istante futuro che soddisfa φ_2 e finquando non si verifica ciò, è sempre vero (l'istante successivo potrebbe essere anche l'istante attuale stesso). φ_1 .
- $\varphi_1 R \varphi_2$ Release: (duale di Until) φ_2 deve essere vera fino al punto (compreso) in cui diventa (per la prima volta) vera φ_1 , se non essa diventa mai vera, φ_2 rimarrà vera all'infinito.

2.2 Semantica LTL

Sia $\alpha \in (2^{AP})^w$

- α soddisfa p sse $p \in \alpha_0$
- α soddisfa $\neg\varphi$ sse α non soddisfa φ
- α soddisfa $\varphi_1 \wedge \varphi_2$ sse α soddisfa φ_1 AND α soddisfa φ_2
- α soddisfa $\varphi_1 \vee \varphi_2$ sse α soddisfa φ_1 OR α soddisfa φ_2
- α soddisfa $X \varphi_1$ sse $\alpha_{\geq 1}$ soddisfa φ_1
- α soddisfa $\varphi_1 U \varphi_2$ sse $\exists j \in \mathbb{N} \alpha_{\geq j}$ soddisfa φ_2 AND $\forall i \in [0, j[\alpha_i$ soddisfa φ_1
Poniamo l'attenzione su un caso particolare di Until (Future) in cui φ_1 è "true". Tale formula indicherà semplicemente che in futuro sarà verificato φ_2 . $F \varphi \stackrel{def}{=} \text{True} U \varphi$
- α soddisfa $\varphi_1 R \varphi_2$ sse $\forall J \in \mathbb{N} \alpha_{\geq J}$ soddisfa φ_2 OR $\exists i \in [0, j[\alpha_i$ soddisfa φ_1
Ora mostriamo il duale di Future (Globaly) la quale indica che da questo istante in poi vale la formula φ . $G\varphi \stackrel{def}{=} \text{False} R \varphi$

Su tali formule è applicabile il teorema di De Morgan, con una estensione per gli operatori introdotti con LTL. Di seguito proponiamo l'estensione del teorema di De Morgan sulle formule NEXT UNTIL:

1. $\neg X \varphi \equiv X \neg\varphi$
2. $\varphi_1 U \varphi_2 \equiv \neg((\neg\varphi_1) R (\neg\varphi_2))$

2.3 Soddisfacibilità e model checking per LTL

Equivalenza di due formule

$\varphi_1 \equiv \varphi_2$ sse $\forall w \in (2^{AP})^w w \models \varphi_1$ sse $w \models \varphi_2$

Indica che φ_1 è equivalente a φ_2 se per ogni parola w appartenente a 2^{AP} , tale modello soddisfa sia φ_1 che φ_2 , in altre parole non esiste un modello che discerne tra i due.

Espansioni a punto fisso (o proprietà di equivalenza)

$\varphi_1 U \varphi_2 \equiv \varphi_2 \vee \varphi_1 \wedge X (\varphi_1 U \varphi_2)$

$\varphi_1 U \varphi_2$ è equivalente a dire: φ_2 è vera nell'istante attuale oppure è vera φ_1 e l'Until sarà vera nell'istante successivo.

$\varphi_1 R \varphi_2 \equiv \varphi_2 \wedge (\varphi_1 \vee X \varphi_1 R \varphi_2)$

$\varphi_1 R \varphi_2$ è equivalente a dire: φ_2 è vera nell'istante attuale e contemporaneamente o è vera φ_1 oppure nell'istante successivo sarà vera il Release

È evidente che, come il Release e l'Until sono uno il duale dell'altro, anche le formule a destra dell'equivalenza sono una il duale dell'altra.

2.3.1 Definizione di validità e soddisfacibilità di una formula LTL

Formula φ è SAT (una formula LTL è soddisfacibile).

φ è Sat sse $w \in (2^{AP})^w$ tale che $w \models \varphi$

Una formula è soddisfacibile solo e soltanto se esiste un modello che la soddisfa.

Formula φ è valida.

φ è Valida sse $\forall w \in (2^{AP})^w$ tale che $w \models \varphi$

Una formula è valida se non esiste un modello che rende la formula falsa (in altre parole, tutti i modelli w appartenenti a $(2^{AP})^w$ soddisfano la formula φ)

2.3.2 Model Checking

Ora definiamo cosa vuol dire che una kripke construction (modello) soddisfa una formula φ

$K \models \varphi \Leftrightarrow L(K) \subseteq L(\varphi)$.

Cioè il linguaggio accettato da K è un sotto insieme dei modelli di una formula φ .

$L(\varphi) \stackrel{def}{=} \{w \in (2^{AP})^w \text{ tale che } w \models \varphi\}$

Quindi ogni possibile parola creata sul grafo K è un modello per φ .

2.4 Sistemi per la verifica (safety, liveness)

Per la verifica formale si definiscono due categorie la safety e la liveness.

La prima è principalmente utilizzata per definire cose che non devono accadere durante l'arco di una esecuzione (ES. accesso contemporaneo ad una risorsa condivisa).

La seconda è definita per indicare che un certo evento deve verificarsi. Utilizzata principalmente per evitare attese indefinite; deadlock.

Ricordiamo cosa si indica con Future (F) e Globally (G) (già precedentemente descritte):

- F φ Future true U φ indica che prima o poi sarà vera φ
- G φ Global false R φ d'ora in poi sarà vera φ

Per meglio capire proponiamo tre esempi (1 di Safety e 2 di Liveness):

Safety : $G(Talk_i \rightarrow \neg Talk_s)$

Se un'entità parla (usa il mezzo) non esiste nessun'altra entità che occupa lo stesso mezzo. Come precedentemente accennato, tale formula evita che due o più risorse accedano ad una risorsa condivisa contemporaneamente.

Liveness : $G(req \rightarrow F (grant))$

Tale formula è letta come: (in ogni instate) se vi è una richiesta, prima o poi (Future) questa viene soddisfatta (es. se faccio una richiesta per una risorsa di in mutua esclusione, prima o poi questa risorsa mi sarà data, evita la starvation).

Nota: Tale formula non tiene in considerazione degli istanti o dell'istante temporale entro il quale tale condizione deve essere soddisfatta, infatti per tali condizioni vi è la possibilità di adoperare logiche ad intervalli (intervalli di tempo entro il cui deve verificarsi un evento).

Ora consideriamo una variante della formula di liveness precedentemente mostrata, in cui la richiesta viene fatta infinitamente spesso:

Liveness : $G(G(F(\text{req})) \rightarrow F(\text{grant}))$.

Tale formula può essere letta: se ho un punto dal quale effettuo infinitamente spesso una richiesta, questa richiesta prima o poi viene soddisfatta.

3 Struttura dei modelli delle formule LTL

3.1 Chiusura di una formula LTL

La chiusura è un insieme di sottoformule della formula φ , quest'insieme s'indicherà con $\text{cl}(\varphi)$, ed è strutturata nel seguente modo:

1. $\varphi \in \text{cl}(\varphi)$
2. $\text{OP } \varphi' \in \text{cl}(\varphi) \Rightarrow \varphi' \in \text{cl}(\varphi)$ con $\text{OP} \in \{\neg, X\}$
Tale proprietà va letta in questo modo: se $\neg\varphi'$ o $X\varphi'$ appartiene alla chiusura, appartiene alla chiusura anche la formula φ' .
3. $\varphi_1 \text{ OP } \varphi_2 \in \text{cl}(\varphi) \Rightarrow \varphi_1, \varphi_2 \in \text{cl}(\varphi)$ con $\text{OP} \in \{\vee, \wedge\}$
4. $\varphi_1 \text{ U } \varphi_2 \in \text{cl}(\varphi) \Rightarrow (\varphi_2 \vee \varphi_1 \wedge X\varphi_1 \varphi_2) \in \text{cl}(\varphi)$
5. $\varphi_1 \text{ R } \varphi_2 \in \text{cl}(\varphi) \Rightarrow \varphi_2 \wedge (\varphi_1 \vee X\varphi_1 \text{ R } \varphi_2) \in \text{cl}(\varphi)$
Nei casi 4 e 5 si dice che si ottiene il one step, one folding poiché per esplicitare la formula si usa il suo primo passo successivo, quindi nell'istante di tempo successivo.

Inutile utilizzare until e release nella definizione 3, poiché per questi due operatori con la definizione 4 e 5 assieme al one step one folding possiamo concludere comunque che se abbiamo $\varphi_1 \text{ R } \varphi_2$ oppure $\varphi_1 \text{ U } \varphi_2$ allora $\varphi_1, \varphi_2 \in \text{cl}(\varphi)$.

3.2 Definizione di Atomo per formule LTL

Un atomo a è un insieme massimale e consistente, esso è definito come $a \subseteq \text{cl}(\varphi)$. Per consistente s'intende che non deve avere al suo interno una formula e contemporaneamente la sua negata.

Nota: Per massimale e consistente non vuol dire che è per forza soddisfacibile, perché localmente potrebbe esserlo (se viene vista come una formula booleana) ma poi in LTL si potrebbe richiedere che nel futuro essa non lo sia (sfruttando opportunamente l'until e il release) così l'insoddisfacibilità della formula all'infinizio dovrà essere possibile e sarà compito della Sequenza di Hintikka.

Indichiamo l'insieme $a \subseteq \text{cl}(\varphi)$ con $\text{atm}(\varphi)$.

L'atomo deve essere massimale, cioè ogni formula presente in $\text{cl}(\varphi)$ o è presente in "a" oppure è presente la sua negata.

Inoltre qualsiasi proposizione atomica anche se non sta in $\text{cl}(\varphi)$ o è presente in a oppure è presente la sua negata.

Formalmente:

1. $\forall \varphi \in \text{cl}(\varphi) \cup \text{AP} \Rightarrow \varphi \in a$ sse $\text{AP} \neg \varphi \notin a$
 Questa regola automaticamente ci permette di verificare la correttezza e completezza, poiché non ci possono essere entrambe le formule.
2. $\varphi_1 \wedge \varphi_2 \in a \Rightarrow \varphi_1, \varphi_2 \in a$
3. $\varphi_1 \vee \varphi_2 \in a \Rightarrow \varphi_1 \in a$ o $\varphi_2 \in a$
 La 4 non è un OR esclusivo, ma possono esser presenti sia φ_1 che φ_2 in a .
4. $\varphi_1 \cup \varphi_2 \in a \Rightarrow \varphi_2 \vee \varphi_1 \wedge X\varphi_1 \cup \varphi_2 \in a$
5. $\varphi_1 \text{ R } \varphi_2 \in a \Rightarrow \varphi_2 \wedge (\varphi_1 \vee X\varphi_1 \text{ R } \varphi_2) \in a$

Dobbiamo notare che non si hanno regole per il next, proprio perché gli atomi sono elementi locali. Il next invece prende in considerazione gli elementi negli istanti successivi, quindi questo passo sarà preso in considerazione nella sequenza d'Hintikka.

3.3 Sequenza di Hintikka

Una prima definizione che possiamo dare è quella che una sequenza di Hintikka è una parola infinita su Atomi. In altre parole l'alfabeto che viene usato per costruire la parola sono gli atomi stessi.

Più Formalmente:

$$\alpha \in (\text{atm}(\varphi))^w$$

Indicheremo con $\text{Hint}(\varphi)$ l'insieme delle sequenze di Hintikka per una formula φ .

α è una sequenza di Hintikka per φ sse $\forall i \in N$ valgono le seguenti:

1. $\varphi \in \alpha_0$
 La sequenza deve essere vera all'inizio, quindi φ dev'essere contenuta nell'atomo iniziale, poichè l'atomo ci segnala che per una generica posizione i tutte le formule sono vere fino quella posizione. Inoltre si vuole verificare che φ sia soddisfacibile, quindi si deve costruire un modello unitamente alla sequenza di Hintikka, e la sequenza dovrà essere vera dall'inizio.
2. Se $X\varphi' \in \alpha_i \Rightarrow \varphi' \in \alpha_{i+1}$
 Che rappresenta la nostra legge di coerenza tra uno stato e lo stato successivo.
3. Se $\varphi_1 \cup \varphi_2 \in \alpha_i \Rightarrow \exists j \in N$ tale che $\varphi_2 \in \alpha_{i+j}$
 Cioè esiste un istante j nel futuro, istante ovviamente finito, in cui troverò φ_2 . Questo ci assicura che l'until non verrà rimandato all'infinito senza mai trovare una φ_2 . (intuitivamente, se trovo infinitamente spesso l'until è perché stò trovando infinitamente spesso anche la φ_2 . Invece se trovassi infinitamente spesso l'until, significa che lo stò svolgendo infinitamente spesso. Ma se non trovo φ_2 sto' portando infinitamente spesso lo svolgimento, che non è quello che ci viene detto dalla semantica).

Riassumendo: la formula presente nella sequenza d'Hintikka sarà vera nell'istante iniziale, cioè la 1), viene verificata localmente nella 2) ed è valida globalmente nella 3).

3.4 Equivalenza tra esistenza di un modello ed esistenza di una sequenza di Hintikka.

Il prossimo teorema mostra che se una formula è soddisfacibile, allora esiste almeno una sequenza di Hintikka, così come possono esistere più modelli per una formula soddisfacibile.

Teorema 3.1. φ è SAT sse $\text{Hint}(\varphi) \neq \emptyset$

Dimostrazione. (\Rightarrow) se φ è SAT allora esiste una parola appartiene a $(2^{AP})^w$ che la soddisfa per definizione.

Formalmente φ è SAT $\Rightarrow \exists w \in (2^{AP})^w \ w \models \varphi$.

Così noi siamo in grado di costruire la sequenza di Hintikka a partire dalla parola w , per fare ciò definiamo un insieme Satseq :

$\text{Satseq}_\varphi(w, i) \stackrel{\text{def}}{=} \{\varphi' \in \text{cl}(\varphi) \text{ tale che } w_{\geq i} \models \varphi'\}$, cioè dall'istante i in poi. (1)

La parola risultante è un atomo, poichè se per assurdo non lo fosse, possono non esser soddisfatte le condizioni degli atomi. Ad esempio supponiamo che non può esser soddisfatta la condizione "and" dell'atomo, ciò è un assurdo poichè entrambe appartengono alla $\text{cl}(\varphi)$. Questo ragionamento può esser fatto anche per le altre regole.

Costruiamo la sequenza di Hintikka:

$$\alpha = \prod_{i \in \mathbb{N}} \text{satseq}_\varphi(w, i)$$

Questa è una parola infinita sugli atomi, per controllare che sia una sequenza di Hintikka dobbiamo verificare le tre proprietà. La prima proprietà è verificata prendendo w stesso come α_0 . Quindi sappiamo che w soddisfa φ , ciò corrisponde a prendere la formula con $i=0$ (cioè quando prendo α_0).

Per il Next (il punto 2) prendiamo in considerazione l'indice i , dunque nella formula si ha un $X\varphi$. Semanticamente ciò significa che al passo successivo sarà vera φ . Quindi φ sarà contenuto in $\text{satseq}_\varphi(w, i+1)$ dunque è soddisfatta.

Per l'until si ha un punto i -esimo per il quale è vero l'until. Per semantica so che esiste un $w_{\geq i+j}$ che soddisfa φ_2 quindi sarà nel $\text{satseq}_\varphi(w, i+1)$ ed è quello che voglio per la definizione.

(\Leftarrow)

Se so che l'insieme di Hintikka non è vuoto, allora ho già una o più sequenze di Hintikka, sia α una di queste.

$$\alpha \in \text{Hint}(\varphi)$$

α è un atomo, quindi localmente è coerente e ci dice anche quali sono le atomic proposition vere, quindi proiettando tutta la sequenza di Hintikka sull'insieme delle atomic proposition si otterrà una parola su 2^{AP} , tale parola è un modello.

Formalmente:

$$w \in (2^{AP})^w \text{ tale che } \forall i \in \mathbb{N} \ w_i \stackrel{\text{def}}{=} a_i \cap AP$$

In a_i sappiamo che ci sono tutte le formule vere, in particolare le atomic proposition vere fino all'istante i , queste ci interessano poichè prenderemo solo quelle che rappresenteranno la valutazione locale della formula, per ogni istante.

Dalla formula otteniamo un modello, ciò si prova per induzione sulla lunghezza della formula.

$$\forall i \in \mathbb{N} \ \forall \varphi' \in \text{Cl}(\varphi) \ \forall \varphi' \in \alpha_i \text{ sse } w_{\geq i} \models \varphi'$$

Per induzione sull'insieme $\text{Cl}(\varphi)$.

Poichè se è vera la φ' , all'istante $i=0$, φ' sarà proprio φ , si ottiene che:

$\varphi \in \alpha_0$ sse $w_{\geq 0}$ (cioè w) soddisfa φ . So che è vero, perché φ' è una sequenza di Hintikka e quindi per il punto 1 delle proprietà delle sequenze di Hintikka φ

$\in \alpha_0$ quindi w è un modello di φ' .

$\varphi' \in \alpha_i$ sse $w_{\geq i} \models \varphi$ è vera poiché, partendo con φ' che è un atomic proposition se questa stà in α allora, per costruzione deve stare anche in w , se non stà in α allora la formula w_i non soddisfa φ .

Ora mostriamo i passi successivi, cioè posta vera φ dobbiamo vedere le formule che la contengono strettamente, cioè le formule φ' che contengono φ (Tralascieremo la dimostrazione per gli operatori booleani e ci concentriamo sugli operatori LTL).

Il **next** è corretto per le proprietà delle sequenze di Hintikka. In particolare se abbiamo nell'insieme $cl(\varphi)$ il next, implica che nell'istante di tempo successivo si verifica φ , precisamente φ su una sottoformula. Avendo già dimostrato che φ è vera, allora essa sarà vera anche nell'istante successivo, quindi essendo vera nell'istante j (con $j > i$) so che nell'istante i $X \varphi$ è vera (perché verrà verificata all'istante j).

Il **Release** è vero banalmente per il suo one step one folding, quindi è sufficiente dimostrare per induzione il passo successivo. Dunque si deve dimostrare che o φ_2 è vera, e quindi sarà vera il release oppure, per costruzione, si troverà che φ_1 è vero.

Per l'**until** vale lo stesso discorso del release, con una differenza: si deve trovare un punto in cui φ_2 è vera, ma per la 3a regola delle sequenze di hintikka tale punto esiste. Notiamo che φ_2 è una sottoformula stretta, quindi per ipotesi induttiva esiste un istante $w_{\geq i+j}$ che soddisferà la formula φ_2 , inoltre per costruzione del one step one folding in tutti gli istanti precedenti si avrà anche il φ_1 che quindi soddisferà l'**until**. \square

4 Soddisfacibilità e model checking per LTL

Per la SAT e il Model Checking devo costruire un automa che costruisce ha come run delle sequenze di Hintikka. Tale automa sarà un generalizzato di Buchi.

4.1 Automa di Buchi per il riconoscimento dei modelli di una formula

L'intuizione dietro la costruzione dell'automa è sostanzialmente implementare il teorema, quindi:

- Gli stati che saranno gli atomi devono contenere φ al loro interno.
- La δ invece dovrà andare da un atomo, a tutti gli atomi voluti dal next (altrimenti non verificherei il punto 2 delle proprietà delle sequenze di Hintikka).
- Inoltre poiché è necessario verificare la sua coerenza, quindi si deve controllare che tutte le atomic proposition che servono siano nel run, il quale deve accettare soltanto le parole che sono proiezioni sulla sequenza d'hintikka implementata dall'automa stesso.

L'unica cosa che rimane è l'accettazione, la quale deve assicurare che prima o poi verifico la formula, quindi non ci devono essere until che si ripeteranno da

soli senza mai verificare la φ .

Verificata che è una sequenza di Hintikka, e che per il teorema è un modello, la satseq sarà il run dell'automata stesso.

L'automata ha la forma $A = \langle \Sigma, Q, \delta, Q_0, F \rangle$, dove F è una famiglia di sottoinsiemi, essendo un generalizzato.

$\Sigma \stackrel{def}{=} 2^{AP}$, perché bisogna accettare parole che sono dei modelli.

$Q \stackrel{def}{=} Atm(\varphi)$, perché il run dev'essere una sequenza di Hintikka.

$Q_0 \stackrel{def}{=} \{a \in Atm(\varphi) \text{ tale che } \varphi \in a\}$ poiché si deve soddisfare la 1a condizione della sequenza di Hintikka.

La δ dovrà soltanto soddisfare la 2a condizione della sequenza d'Hintikka e che la parola che l'automata stà leggendo sarà una proiezione su AP.

$\delta(q, \sigma) \stackrel{def}{=} \{a \in Atm \text{ tale che } \forall X \varphi' \in q, \varphi' \in q, \varphi' \in a\}$ se $\sigma = q \cap AP$
 \emptyset altrimenti.

Inizialmente, nello stato q si controlla che σ sia una proiezione in AP, se non è così significa che si sta creando un run errato, quindi non si stà costruendo la parola giusta, allora "devo buttare via il run".

Se invece il run che stò costruendo è giusto devo andare in tutti gli atomi che effettivamente soddisfano il next dell'atomo precedente.

Naturalmente non vi è un solo run, infatti l'automata non è deterministico (e non può essere determinizzabile, inoltre nota che l'automata non può essere costruito in tempo meno che esponenziale perché LTL è chiusa rispetto al complemento e perché gli atomi sono sottoinsiemi).

L'accettazione,

$F = \{F_{\varphi_1 U \varphi_2} \subseteq Q \mid \varphi_1 U \varphi_2 \in cl(\varphi)\}$

Si ha un insieme per ogni until, quest'insieme mantiene tutti gli atomi che se "vedono" l'until, ciò significa che hanno visto anche φ_2 questo implica che non potrò mai andare all'infinito senza vedere φ_2 (non potrò mai costruire una parola che non è una Sequenza di Hintikka).

$F_{\varphi_1 U \varphi_2} = \{a \in Q \text{ tale che } \varphi_1 U \varphi_2 \in a \Rightarrow \varphi_2 \in cl(\varphi)\}$

Cioè se si vede l'until implica che è vera anche la formula φ_2 .

Se l'automata accetta vuol dire che ogni until prima o poi sarà soddisfatto poiché ne troverò la φ_2 corrispondente, ovviamente, infinitamente spesso.

Ricapitolando:

gli stati rispettano il punto 1

la delta rispetta il punto 2

la F rispetta il punto 2 e 3.

Quindi le parole sono modelli per il teorema.

4.2 Soluzione del problema della soddisfacibilità

Tale problema può essere risolto utilizzando un automata e su di esso applicando la liveness.

- Se la liveness porta al vuoto questo implica che la formula non è soddisfacibile.

- altrimenti vuol dire che accetta una parola, e tale parola è proprio un modello per la formula.

φ è Sat sse $L(N\varphi) \neq \emptyset$

(\Rightarrow) Se Sat ammette un modello (per il teorema “equivalenza di un modello ed esistenza di una sequenda di Hintikka”) esiste una sequenda di Hintikka e proprio questa è un Run dell’automa. Se ne deriva che quindi questa parola è accettata.

(\Leftarrow) Se $L(N\varphi)$ è diverso dal vuoto prendiamo il Run (che non è altro che una sequenda di Hintikka) per il teorema (equivalenza di un modello ed esistenza di una sequenda di Hintikka) tale parola non è altro che un modello per la formula.

4.3 Soluzione del problema del model checking

$K \models \varphi$ SSE $L(K) \subseteq L(\varphi)$ SSE $L(A_K) \subseteq L(A_\varphi)$

$L(A_\varphi)$ ed $L(A_K)$ non sono altro che i relativi linguaggi degli automi A_φ e K (già visti precedentemente).

Se voglio che K soddisfi φ allora devo avere che il linguaggio accettato da K non è altro che un sottoinsieme di $L(\varphi)$ ma ciò non è altro che equivalente a dire:

$$L(A_k) \cap L(A_\varphi) = \emptyset \text{ sse } L(A_k) \cap L(\varphi) = \emptyset$$

Dove:

- $L(A_\varphi)$ è equivalente a $L(A_{\neg\varphi})$
- $L(A_k) \cap L(A_{\neg\varphi})$ non è altro che $L(A_k \cap A_{\neg\varphi}) = \emptyset$

Quindi K soddisfa φ sse il linguaggio dell’intersezione dell’automa per K e per $\neg\varphi$ è uguale al vuoto. In altre parole non esiste in K un modello per la formula negata, quindi tutte le parole di K sono modelli per φ

4.4 Complessità dei problemi di soddisfacibilità e model checking

Entrambi i problemi visti appartengono alla classe PSpace-Complete ciò lo si può dimostrare utilizzando una riduzione da una macchina di Turing a spazio polinomiale. A tal punto, mostrato essere uno dei due problemi PSpace-complete si riduce l’altro a questo.

Per intuizione:

Il problema della soddisfacibilità è PSpace perché l’automa preso in considerazione $N\varphi$ è esponenziale, il vuoto di un automa ND di Buchi si effettua in LogSpace il logaritmo di un esponenziale è banalmente un polinomiale (Nota: naturalmente non mantengo tutto l’automa in memoria, ma man mano che l’algoritmo del vuoto mi richiede uno stato questo viene creato).

Similmente anche per il problema del model checking

- A_k è un automa polinomiale in K
- $A_{\neg\varphi}$ è un automa esponenziale nella lunghezza della formula

Applicando l'algoritmo si ottiene il logaritmo della taglia di $A_k \cap A_{\neg\varphi}$ dove l'intersezione non è altro che il prodotto delle due taglie, quindi logaritmo nella taglia del modello e polinomiale nella taglia della formula.