Logica Modale, Temporale e Model Checking

Giorgio Delzanno
DISI Università di Genova

12 luglio 2010

Indice

1 Introduzione

La logica classica è un linguaggio formale nato per descrivere la matematica. La logica è anche alla base di concetti fondamentali dell'informatica quali la nozione di circuito e schemi di operazioni su informazioni digitali (and, or, ecc.) e la nopzione di problema di decisione alla base della teoria della complessità. Negli ultimi decenni la logica classica ha trovato anche numerose altre applicazioni all'informatica, ed intelligenza artificiale in particolare, quali rappresentazione della conoscenza, sistemi esperti, ragionamento automatico, specifica di sistemi, programmazione, e molti altri.

Sintassi Un linguaggio del prim'ordine è costituito dai seguenti ingredienti.

- Termini (costanti, simboli di funzioni, applicazioni)
- Formule atomiche $p(t_1, \ldots, t_n)$ dove p è un simbolo di predicato e t_i è un termine
- Connettivi: $\wedge$, $\vee$, $\supset$ (implicazione), $\neg$
- Quantificazione: $\forall$, $\exists$

Le *formule* proposizionali sono del tipo: $(p \wedge \neg q) \vee r$ dove p, q, r simboli proposizionali (i.e. interpretabili come $vero$ o $falso$) Al prim'ordine si aggiunge la quantificazione su variabili di tipo base: $\forall x.\exists y.\ \ (p(x, y) \wedge \neg q(x)) \vee r(y)$ Al second'ordine la quantificazione varia su simboli di predicato, e.g., $\exists p.\forall x.\exists y.\ \ (p(x, y) \wedge \neg q(x)) \vee r(y)$ Più in generale all'ordine superiore si può quantificare su variabili con tipo funzionale arbitrario, e.g., $\exists p.\exists q.\exists x.\exists y.\ \ (p(q, y) \wedge \neg q(x)) \vee r(y)$

Semantica della Logica classica La semantica della logica classica è basata sulla nozione di interpretazione dei termini e delle formule. Un'interpretazione è costituita da

- Una struttura algebrica per interpretare le costanti e simboli di funzione
- Una collezione di relazioni per interpretare i simboli di predicati
- Le tabelle di verità per i connettivi
- Le regole di istanziazione per i quantificatori

I concetti fondamentali della semantica sono: a nozione di *soddisfacibilità* (verità rispetto ad un modello), validità (verità rispetto a tutti i modelli) e conseguenza logica (i modelli di una formula sono modelli anche della formula che ne segue logicamente). La semantica della logica classica è basata su una visione *statica* della realtà. Ad esempio, data una formula F e fissato il dominio di interpretazione: (in questo momento) F è vera o falsa

Ragionamento Automatico Una teoria (insieme di formule) può essere visto come un modello formale di una certa realtà di interesse. Un sistema deduttivo rappresenta un tentativo per rendere semi-automatico (in alcuni casi completamente automatico) il ragionamento sulla teoria. Una *prova* costruita attraverso le regole del sistema deduttivo diventa quindi un possibile *testimone* della verità o falsità di un'asserzione (formula).

Ad esempio, le tabelle di verità in combinazione con una adeguata *strategia di valutazione* possono essere viste come un possibile algoritmo per testare la validità di una formula proposizionale. Ad es. se $\rightsquigarrow$ rappresenta un passo di semplificazione allora

$$true \wedge (false \supset (true \wedge false)) \ \rightsquigarrow\ true \wedge (false \supset false) \ \rightsquigarrow\ true \wedge true \ \rightsquigarrow\ true.$$

Adeguatezza della logica del prim'ordine Nonstante l'importanza della logica classica, siamo sicuri che un formalismo quale la logica del prim'ordine sia veramente la logica giusta per esprimere concetti fodamentali dell'informatica? Più precisamente si riescono a modellare (con quale complessità?) e a ragionare su aspetti quali *gestione delle risorse (tempo e spazio)*, *cambiamento di stato e computazione*, *parallelismo/sequenzialità*, *sicurezza*, *concetti di base di programmazione (terminazione, assegnamento, iterazione)*, *concetti avanzati di programmazione (oggetti, ereditarietà)*.

Ad esempio supponiamo di voler modellare formalmente la seguente asserzione

"Prima o poi gli impiegati di livello 1 passeranno al livello 2

Per formalizzare tale concetto dobbiamo considerare i seguenti aspetti: nozione di stato, tempo, e cambiamento di stato. Lo stato potrebbe essere modellato tramite una formula atomica $impiego(x, y, t)$ dove $x = nome$, $y = livello$, $t = tempo$. Il tempo potrebbe essere modellato tramite costanti: 98, 99, ...e predicati: $t > 98$, $t > t'$, $t = t'$. Il cambiamento di stato sarà rappresentato tramite asserzioni su tali formule: $impiegato(a, 1, 98)$, $impiegato(b, 2, 99)$, ... Sulla base di questo linguaggio dovremmo poi considerare assunzioni quali:

- Il tempo scorre: $\forall t.\exists t'.t' > t$
- In ogni instante il livello di un impiegato è unico:

$$\forall x.\neg(\exists s.\exists s'.\exists t.impiegato(x, s, t) \wedge impiegato(x, s', t) \wedge s \neq s')$$

La nostra asserzione sarà dunque l'insieme delle assuzioni in congiunzione con la formula

$$\forall t.\forall x.impiegato(x, 1, t) \supset \exists t' > t.impiegato(x, 2, t')$$

Quindi la formalizzazione anche di un concetto apparentemente semplice potrebbe comportare numerosi passaggi peraltro difficili da *automatizzare*.

Logiche non-standard Le logiche *non-standard* sono state introdotte per ovviare a limiti della logica classica o più semplicemente per facilitarne l'uso. In entrambi i casi lo scopo è di definire linguaggi logici per ottenere, in particolari domini, una descrizione più ricca e flessibile (o semplicemente di natura differente) rispetto alla logica classica. Nelle *estensioni della logica classica* quali Logica modale, dinamica, temporale, ecc. si estendono il linguaggio e la semantica, ad es., con nuovi connettivi e assunzioni riguardo la loro semantica (ad es. tramite nuovi assiomi). Le logiche *alternative alla logica classica* quali la logica intuizionista, multi-valore, lineare, ecc., si pongono invece in contrapposizione alla logica classica falsificando teoremi della logica classica, ad es. la *legge del terzo escluso* $A \vee \neg A$ non vale più in logica intuizionista. Le logiche non-standard hanno una lunga tradizione in informatica. Ecco alcuni esempi significativi: logica modale per provare proprietà di programmi [23], logica temporale per specifica e verifica di sistemi concorrenti [34], logica intuizionista per specifica, costruzione e verifica di programmi [33]; logica temporale per verifica *automatica* di sistemi hardware [15, 16]; logica lineare e nozione di risorsa [20] logica modale per sistemi mobili [8]

Le logiche non-standard sono state applicate anche a svariati aspetti dell'intelligenza artificiale. Alcuni esempi sono: logica modale per rappresentare azioni e credenze [38], logica temporale per modellare eventi, piani [3, 35], temporal logic programming [19, 18], logica modale per agenti [22], intuitionistic logic programming [37], logica temporale per agenti [17, 45].

2 Parte I: Logica Modale

Vogliamo distinguere tra formule che *devono* essere *sempre* vere e formule la cui verità dipende dal contesto nel quale si valutano. In altre parole vogliamo *qualificare* la verità di una formula e modellare asserzioni quali *è necessario (vero in ogni contesto) che*; *è possibile che*; *è obbligatorio*; *è permesso*; *è proibito*; *sara sempre vero che*; *potrà essere vero che*. Il contesto di interpretazione di una formula modale dipende dall'interpretazione intuitiva della nostra logica: istante di tempo, mondo in cui ci si trova, stato di un programma durante l'esecuzione, ecc. Ad esempio: Il mio nome è Giorgio Delzanno! *vera in qualunque contesto*. Il programma $leggo(x); y := 0;$ non assegna mai 1 a y! *vera per ogni possibile esecuzione*. Possiedo una Ferrari! *vera solo se cambio lavoro*. Il programma $leggo(x); y := x;$ assegna 1 a y! *dipende dall'ambiente esterno*.

2.1 Struttura della logica modale

L'idea generale è di arricchire la logica classica con nuovi operatori cosiddetti *modali* per qualificare la verità di una formula: $\mathbf{L}\varphi = \varphi$ *è necessariamente vera*; $\mathbf{O}\varphi = \varphi$ *è obbligatorio*; $\mathbf{G}\varphi = \varphi$ sarà *sempre* vera; ecc. La semantica intuitiva degli operatori viene data tramite nuovi assiomi e un sistema deduttivo (i.e. una logica modale) In molti casi è difficile trovare una controparte *model theoretic* che estenda in modo naturale la semantica della logica classica. Storicamente la logica modale è nata per modellare le asserzioni:

- $\mathbf{L}\varphi = \varphi$ è necessario
- $\mathbf{M}\varphi = \varphi$ è possibile

Il linguaggio della logica classica viene arricchito con i due operatori $\mathbf{L}$ e $\mathbf{M}$. Otteniamo formule come: $\mathbf{M}(a \wedge b)$; $(\mathbf{L}a) \wedge (\mathbf{M}b)$; $\forall x.\mathbf{M}p(x)$; $\mathbf{M}a \supset \mathbf{M}(b \supset c)$. Il significato di $\mathbf{L}$ e $\mathbf{M}$ viene modellato tramite assiomi: infinite discussioni su quali assiomi modellano in modo soddisfacente la nozione di necessità ed eventualità.

2.2 Logica K (Saul Kripke 60's)

Possiamo definire $\mathbf{M}$ in funzione di $\mathbf{L}$:

$$\mathbf{Def_M}. \qquad \mathbf{L}A \equiv \neg\mathbf{M}\neg A$$

Inoltre sembra ragionevole che: *se A è necessariamente vera e da A segue B in ogni circostanza, allora B è necessariamente vera*

$$\textbf{Assioma K}. \qquad \mathbf{L}(A \supset B) \supset (\mathbf{L}A \supset \mathbf{L}B).$$

Infine, sembra ragionevole che *se F è un teorema della nostra logica allora anche $\mathbf{L}F$ è un teorema.*

$$\textbf{Necessitation Rule.} \qquad \text{Se } \vdash F, \text{ allora } \vdash \mathbf{L}F$$

2.3 Sistema Deduttivo per la Logica K

Il sistema $\mathbf{K}$ è formato dagli *assiomi*: $\mathbf{Def}_M$**,** $\mathbf{K}$, e $\mathbf{PL}(= A$, se A è una tautologia rispetto ai connettivi classici); e dalle regole: Necessitation Rule e Modus Ponens (Se $A \supset B$ e A, allora B).

$\mathbf{K}$ modella veramenta la nozione di necessità? Ad esempio, possiamo dedurre che se A è necessaria allora A è vera? (Di nuovo sembra un'assuzione ragionevole.) Tuttavia, $\mathbf{L}A \supset A$ non è un teorema di $\mathbf{K}$. La logica $\mathbf{K}$ è troppo debole per questa nozione di necessità.

2.4 Altre logiche

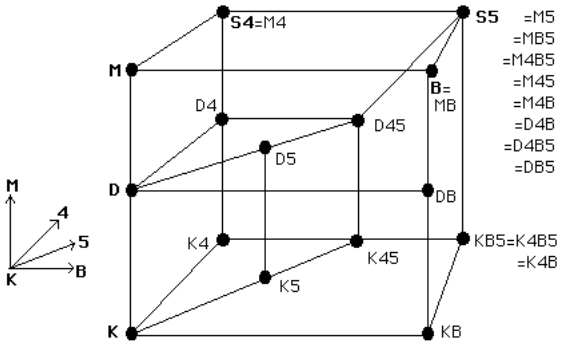
Aggiungiamo altri assiomi.

Se A è necessaria, allora è vera **Assioma T**. $\mathbf{L}A \supset A$

Se A è possibile, allora è possibile in ogni circostanza. **Assioma 5**. $\mathbf{M}A \supset \mathbf{LM}A$

Otteniamo una classe di logiche denominate $\mathbf{T}$, $\mathbf{S5}$, ecc.

La tabella in Fig. 1 mostra le logiche ottenute aggiungendo la lista di assiomi indicata nel nome alla logica $\mathbf{K}$.



(disegno di J. W. Garson, Copyright 2000,2001)

Logic		*Axioms*
K		**K**
M	chiamata anche **T**	**K + M**
D		**K + D**
B		**K + M + B**
S4		**K + M + 4**
S5		**K + M + 5**

Figura 1: Diagramma dell logiche modali più comuni.

2.5 Sistema Deduttivo S5

Gli assiomi del sistema deduttivo di S5 sono: $\mathbf{Def}_M$**.** $\mathbf{L}A \equiv \neg\mathbf{M}\neg A$; **T.** $\mathbf{L}A \supset A$; **K.** $\mathbf{L}(A \supset B) \supset (\mathbf{L}A \supset \mathbf{L}B)$; **5.** $\mathbf{M}A \supset \mathbf{LM}A$; **PL.** A, se A è una tautologia in logica classica Le regole inferenziali di **S5** sono: **Necessitation Rule:** Se A, allora $\mathbf{L}A$; **Modus Ponens:** Se $A \supset B$ e A, allora B.

Il sistema deduttivo **S5** è sound e completo rispetto ai modelli di Kripke che vedremo nella prossima sezione.

Example 1 (Esempio di Deduzione in S5) $A \supset \mathbf{M}A$*:* se A è vero ora, allora A è possibile *Prova utilizzando sistema S5:*

$$\dfrac{\dfrac{\dfrac{}{\mathbf{L}\neg A \supset \neg A}\ T}{A \supset \neg\mathbf{L}\neg A}\ \textit{Equivalenza logica } \supset}{A \supset \mathbf{M}A}\ \textit{Applicando } Def_M : \ \mathbf{M}A \equiv \neg\mathbf{L}\neg A$$

3 Semantica Model-Theory

Per capire meglio il significato degli operatori modali occorrerebbe avere una semantica stile *tabelle di verità* per i connettivi classici. Tuttavia le modalità introducono una componente dinamica nella nostra logica: *vero in ogni possibile contesto*, *esiste un contesto in cui è vero*, ecc. Saul Kripke ha proposto una semantica in cui il dominio di interpretazione delle formule è strutturato: una formula va interpretata a seconda del punto di vista dal quale guardiamo l'universo (costituito da un'insieme di possibili mondi). Per ottenere completezza: varie semantiche a seconda del sistema considerato.

3.1 Modelli di Kripke per S5

Interpretiamo le formule su un insieme di *mondi possibili*: un'*osservatore esterno* valuta la verità delle formule modali.

Definition 1 *Un* frame *è una tupla* $M = \langle W, D, F \rangle$ *dove*

- W *è l'insieme dei* mondi
- D *è l'insieme degli individui*
- F *è la funzione di interpretazione di funzioni (uguale in tutti i mondi) e predicati (varia a seconda del mondo considerato)*

Ad es. $W = \{w_1, w2\}$, $F(w_1, p) = true$, $F(w_2, p) = false$, $F(w_1, p(a)) = false$, ...

Soddisfacibilità in un frame Dato frame M e un mondo w:

- $M, w \models p$ se p è vero in w $(F(w, p) = true)$
- $M, w \models A \vee B$ se $M, w \models A$ oppure $M, w \models B$
- $M, w \models \neg A$ se $M, w \not\models A$
- ...
- $M, w \models \mathbf{L}A$ se $\forall w' \in W$, $M, w' \models A$
- $M, w \models \mathbf{M}A$ se $\exists w' \in W$ t.c. $M, w' \models A$

Una formula è *valida* se è vera per ogni frame M e mondo w. Il Sistema S5 è sound e completo rispetto a questa nozione di validità! Consideriamo alcuni esempi di modelli per formule di **S5**.

Example 2 *Fig. 2 mostra un esempio di modello di Kripke. In questo modello la formula* $\mathbf{M}p$ *risulta vera mentre la formula* $\mathbf{L}p$ *risulta falsa.*

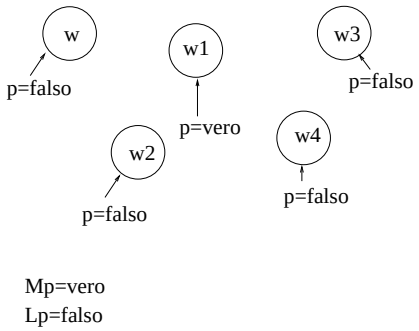


Figura 2: Esempio di Modello.

Vediamo ora dei controesempi a formule non valide in **S5**.

Example 3 (Formule Non Valide in S5) *La formula* $A \supset \mathbf{L}A$ *non è valida in* **S5***: infatti, in generale, se A vale ora non è detto che A sia necessariamente vero. Un possibile controesempio è illustrato in Fig. 3. Notate la differenza tra questa formula e la* **Necessitation Rule***:* $\models A$ *implica* $\models \mathbf{L}A$.

Altri esempi di formule non valide e relativi controesempi sono illustrati in Fig. 4 e Fig. 5.

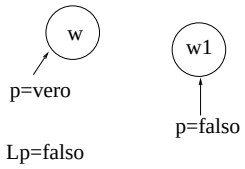


Figura 3: Controesempio per $A \supset \mathbf{L}A$

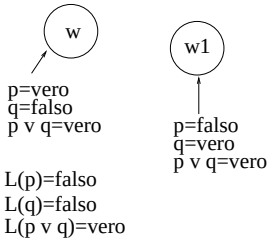


Figura 4: Controesempio per $\mathbf{L}(A \vee B) \supset \mathbf{L}A \vee \mathbf{L}B$

3.2 Semantica generalizzata

La semantica (e quindi la logica) di S5 non cattura nozioni quali *tempo* o *sequenza di esecuzione*: occorre una semantica più generale. I Modelli di Kripke generalizzati hanno le seguenti caratteristiche: insieme di mondi possibili collegati tra loro attraverso una relazione di accessibilità. Una funzione di interpretazione che *dipende dal mondo in cui ci si trova*. Una semantica per le modalità **L** e **M** che dipende dal punto di vista a partire da un determinato mondo. Ad es. la relazione di accessibilità dei modelli di **S5** è semplicemente una *relazione di equivalenza* tra i mondi. Formalmente i modelli di Kripke generalizzati sono definiti come segue.

Definition 2 (Modal Frame) *Un modal frame è una ennupla* $M = \langle W, D, F, R \rangle$ *dove*

- W *è l'insieme dei* mondi
- D *è l'insieme degli individui*
- R *è la relazione (binaria) di accessibilità:* $W^2 \rightarrow \{0, 1\}$
- F *è la funzione di interpretazione di termini (fissata per tutti i mondi) e predicati (una famiglia di interpretazioni* F_w *con* $w \in W$*)*

La relazione di soddisfacibilità è definita come segue. Dato un frame M e un mondo w:

- $M, w \models p$ se p è vero in w $(F(w, p) = true)$
- $M, w \models A \vee B$ se $M, w \models A$ oppure $M, w \models B$'
- $M, w \models \neg A$ se $M, w \not\models A$
- (simil. per gli altri connettivi booleani)
- $M, w \models \mathbf{L}A$ se $\forall w' \in W$ t.c. wRw', $M, w' \models A$
- $M, w \models \mathbf{M}A$ se $\exists w' \in W$ t.c. wRw' e $M, w' \models A$

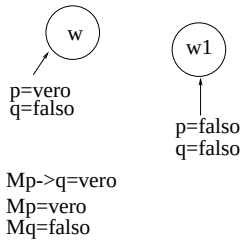


Figura 5: Controesempio per $\mathbf{M}(A \supset B) \supset (\mathbf{M}A \supset \mathbf{M}B)$

Una formula è valida in una *classe* C di frames se è valida per ogni $M \in C$. Notate che sulla base delle precedenti definizioni vale che $\mathbf{L}A \equiv \neg\mathbf{M}\neg A$.

Example 4 *In Fig. 6 mostriamo un esempio di Modal Frame in cui la relazione R è è anti-simmetrica, ma non è transitiva (wRw_1 e w_1Rw_2 ma non è vero che wRw_3) nè tanto meno riflessiva.*

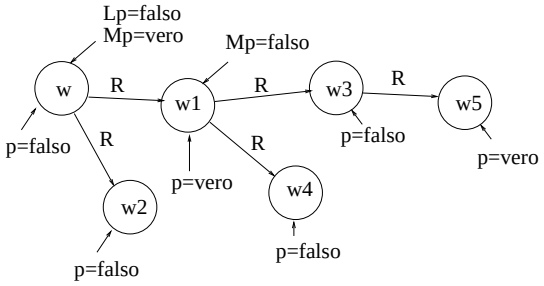


Figura 6: Esempio di Modal Frame

Assiomi, relazione di accessibilità e classi di frames

- Esiste una corrispondenza tra assiomi in logica modale, formule del primo ordine su R (proprietà della relazione di accessibili) e classi di frames:

$$
\begin{array}{ll}
-\mathbf{L}\varphi \supset \mathbf{M}\varphi & \exists u.wRu \\
 & =\text{'Seriale'} \\
-\mathbf{L}\varphi \supset \varphi & wRw \\
 & =\text{'Riflessivo'} \\
-\mathbf{L}\varphi \supset \mathbf{LL}\varphi & wRv \wedge vRu \supset WRu \\
 & =\text{'Transitivo'} \\
-\varphi \supset \mathbf{LM}\varphi & wRv \supset vRw \\
 & =\text{'Simmetrico'} \\
-\mathbf{LL}\varphi \supset \mathbf{L}\varphi & wRv \supset \exists u(wRu \wedge uRv) \quad =\text{'Denso'}
\end{array}
$$

Utilizzando la definizione di soddisfacibilità, dimostriamo, ad esempio, che la validità dell'assioma $\mathbf{L}\varphi \supset \mathbf{M}\varphi$ forza R ad essere *seriale*, cioè $\forall w.\exists u.wRu$. Supponiamo che $\mathbf{L}\varphi \supset \mathbf{M}\varphi$ sia valido allora $\forall M.\forall w.\ M, w \not\models \mathbf{L}\varphi \vee M, w \models \mathbf{M}\varphi$ sse $\forall M.\forall w.\ \neg(\forall u \in W.wRu \supset M, u \models \varphi) \vee (\exists u \in$

$W.wRu \wedge M, u \models \varphi)$ sse $\forall M.\forall w.\ (\exists u \in W.wRu \wedge M, u \not\models \varphi) \vee (\exists u \in W.wRu \wedge M, u \models \varphi)$ sse $\forall M.\forall w.\ \exists u \in W.wRu \wedge (M, u \not\models \varphi \vee M, u \models \varphi)$ sse $\forall w.\exists u \in W.wRu$.

Per dimostrare la relazione tra assiomi e proprietà di R possiamo anche usare un ragionamento per contrapposizione: quale classe di modelli falsifica l'assioma in considerazione? Consideriamo l'assioma $\mathbf{L}\varphi \supset \mathbf{M}\varphi$. Per falsificarlo occore un modello M, un mondo w ed una formula f tale che $M, w \models \mathbf{L}f$ (f è necessariamente vera in w) e $M, w \not\models \mathbf{M}f$ (f non è possibile in w) Cioè dobbiamo soddisfare le due formule:

- $M, w \models \mathbf{L}f \ \equiv\ \forall u \in W.\ wRu \supset M, u \models f$
- $M, w \not\models \mathbf{M}f \ \equiv\ \forall u \in W.\ wRu \supset M, u \not\models f$

ossiamo prendere un modello con R vuota rispetto al mondo w. Per poter includere l'assioma nella nostra logica modale dobbiamo escludere modelli di questo tipo.

Il teoprema di Scott-Lemmon fornisce un'interessante caratterizzazione degli assiomi in termini di proprietà di R e viceversa.

Theorem 1 (Scott-Lemmon) *Lo schema di assioma*

$$\underbrace{\mathbf{M}\ldots\mathbf{M}}_{h}\underbrace{\mathbf{L}\ldots\mathbf{L}}_{i}\varphi \supset \underbrace{\mathbf{L}\ldots\mathbf{L}}_{j}\underbrace{\mathbf{M}\ldots\mathbf{M}}_{k}\varphi$$

corrisponde alla seguente proprietà di R

$$wR^h v \ \wedge\ wR^j u \ \supset\ \exists x.(vR^i x \ \wedge\ uR^k x)$$

- *dove* $R^k = \underbrace{R \circ \ldots \circ R}_{k}$
- *e la composizione di relazioni $\circ$ è definita come segue: $w(R \circ R')v$ sse esiste u t.c. wRu e $uR'v$*

Ad es. con $h = 0, i = 1, j = 0, k = 1$ otteniamo $\mathbf{L}\varphi \supset \mathbf{M}\varphi$ e $wR^0 v \wedge wR^0 u \supset \exists x.(vR^1 x \wedge uR^1 x)$ cioè $\exists u.(wRu)$

Restringendo il tipo di relazione di accessibilità otteniamo quindi diverse logiche modali:

- R riflessiva: logica T;
- R riflessiva+transitiva: logica $S4$; (Assioma 4: $\mathbf{L}A \supset \mathbf{LL}A$);
- R equivalenza: logica $S5$.

e così via.

4 Relazione con Logica Classica

Dalla definizione della semantica di Kripke è chiaro che è possibile codificare una logica modale in logica classica. Ad esempio possiamo codificare la relazione di soddisfacibilità attraversop un predicto *true* definito come segue.

- Interpretazione dei predicati nei mondi = un'insieme di formule $true(p, w)$, p predicato, w mondo
- Relazione di accessibilità = relazione $r(w, w')$

- Soddisfacibilità formule modali:

$$true(\mathbf{L}F, W) \ sse \ \forall W'.r(W, W') \supset true(F, W')$$
$$true(\mathbf{M}F, W) \ sse \ \exists W'.r(W, W') \wedge true(F, W')$$

Codificando gli assiomi tramite una teoria del I ordine (che stabilisce le proprietà della relazione r) otteniamo che una formula modale è valida sse la sua traduzione è valida in logica del I ordine. Notate che le equivalenze logiche di deducono anche tramite la traduzione in logica del I ordine. Se ad esempio assumiamo che $\neg true(\neg F, W')) \equiv true(F, W')$ allora si ha che

$$\begin{array}{rcl} \neg true(\mathbf{L}\neg F, W) & \equiv & \neg\forall W'.r(W, W') \supset true(\neg F, W') \\ & \equiv & \exists W'.\neg(r(W, W') \supset true(\neg F, W')) \\ & \equiv & \exists W'.r(W, W') \wedge \neg true(\neg F, W') \\ & \equiv & true(\mathbf{M}F, W). \end{array}$$

Per questa ragione gli operatori modali hanno proprietà simili alle proprietà di formule quantificate universalmente ($\mathbf{L}$) ed esistenzialmente ($\mathbf{M}$). Ad es. $\mathbf{L}F \vee \mathbf{L}G \supset \mathbf{L}(F \vee G)$, mentre $\mathbf{L}(F \vee G) \not\supset \mathbf{L}F \vee \mathbf{L}G$.

5 Quantificazione: Fixed vs World-Relative Domain

Aggiugiamo i quantificatori alla nostra logica modale: Cosa vogliono dire $\exists x.p(x)$, $\exists x.\mathbf{L}p(x)\ldots$? Vi sono diverse scuole di pensiero. Ad esempio:

(1) dominio di quantificazione fissato per tutti i mondi (come avevamo specificato nella def. di frame)

(2) domino di quantificazione dipendente dal mondo in cui ci si trova

Ogni scelta ha vantaggi e svantaggi: (1) semplifica la semantica; (2) è più flessibile Vediamo alcuni problemi relativi a queste due scelte.

Fixed Domain Supponiamo che il dominio di quantificazione sia unico per tutti gli oggetti. Consideriamo l'asserzone *esiste un uomo che ha firmato la Dichiarazione di Indipendenza*, cioè

$$\exists x.Man(x) \wedge Sign(x)$$

Siccome il dominio contiene gli individui di tutti i mondi possibili la formula è vera. Se intendiamo invece *esiste al giorno d'oggi* allora la nostra formalizzazione non è corretta. Per ottenere tale concetto una soluzione potrebbe essere quella di introdurre un altro operatore $\mathbf{E}$=*esiste attualmente* e scrivere:

$$\exists x.\mathbf{E}x \wedge Man(x) \wedge Sign(x)$$

Esistono sistemi deduttivi che gestiscono con la quantificazione e l'operatore $\mathbf{E}$.

World-relative Domain Supponiamo che il dominio di quantificazione dipenda dal mondo ($D(w)$). Allora la formula $\exists x.(x = t)$ (per un qualche termine t) è valida in logica classica. Attraverso la Necessitation Rule otteniamo che $\mathbf{L}\exists x.(x = t)$ è valida. Supponiamo che t indentifichi Abramo Lincoln. Da questa assunzione segue che stiamo asserendo che in ogni possibile mondo esiste Abramo Lincoln ($\mathbf{L}\exists x.x = lincoln$ è valida). Per ovviare a questo tipo di problemi Kripke ha proposto una soluzione drastica: togliere i termini!

Nel seguito restringeremo la nostra attenzione al caso proposizionale e quindi non incontreremo più questo tipo di problematiche.

Riepilogando abbiamo visto quali sono le idee generali della logica modale: introdurre una nozione di verità dipendente dal contesto (dinamica) la cui semantica viene formalizzata tramite modelli di Kripke. Un Modello di Kripke non è altro che un *grafo*. Possiamo raffinare ulteriormente la semantica? Ad esempio: sarebbe interessante poter modellare il *punto di vista* di *osservatori diversi* o l'*effetto* di *azioni* di tipo diverso Nella prossima sezione vedremo che esistono estensioni in tale direzione.

6 Logica Multi-modale

La logica modale permette di esprimere la dinamica nella conoscenza. Supponiamo però di voler modellare la conoscenza dinamica di un'insieme di *agenti*. Possiamo raffinare ulteriormente la logica modale introducendo famiglie di operatori modali etichettati:

$$\mathbf{L}_a, \mathbf{M}_a \ con \ a \in Agents.$$

L'insieme delle etichette $Agents$ può essere visto come l'insieme dei nomi degli agenti. $\mathbf{L}_a F$ può essere interpretata in vari modi:

- l'agente a conosce F;
- l'agente a crede F;
- F è vera dopo aver eseguito l'*azione* a;
- ...

La semantica degli operatori modali può essere data raffinando la relazione di accessibilità che diventa una relazione ternaria $R : Label \times W^2 \rightarrow \{0, 1\}$. Ogni sottorelazione R_a può avere caratteristiche *speciali*! Ad es. R_a riflessiva, R_b rifl+transitiva, ecc.

Esempio La Fig. 7 mostra un possibile modello multi-modale, cioè un grafo con archi etichettati. Prima di concludere questa breve introduzione agli aspetti fondamentali di logica modale, nella

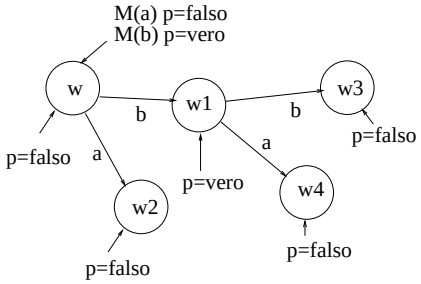


Figura 7: Esempio di modello multi-modale

prossima sezione vedremo una panoramica di logiche modali di varia natura cercando di evidenziare il loro campo di applicazione.

7 Esempi di Logiche Modali e loro Applicazioni

7.1 Logica Deontica

La logica deontica rappresenta un classico esempio di logica per modellare *stati mentali* dell'essere umano, ed ha origini di carattere prettamente filosofico. La *deontologia* è in fatti la scienza che studia la nozione di *obbligazione*. Tra le applicazioni della deontologia troviamo lo studio della *teoria del diritto* dove un sistema giuridico, o normativo, è visto come un insieme di norme con relazioni di tipo deduttivo. Storicamente in questo ambito si introducono tre operatori:

- $\mathbf{O}F$= F è obbligatorio
- $\mathbf{P}F$= F è permesso
- $\mathbf{F}F$= F è proibito

Alcuni degli assiomi utilizzati per modellare il significato di tali operatori sono:

- $\mathbf{P}F \equiv \neg\mathbf{O}\neg F$: F è permesso sse non è vero che è obbligatorio $\neg F$!
- $\mathbf{F}F \equiv \mathbf{O}\neg F$: F è proibito sse è obbligatorio $\neg F$!
- $\mathbf{O}F \supset \mathbf{F}\neg F$: F è obbligatorio implica $\neg F$ è proibito!

Per approfondimenti su logica deontica vedere ad esempio [4].

7.2 Knowledge and Beliefs

La logica multimodale trova interessanti applicazioni per la modellazione di sistemi multi-agenti. Supponiamo di voler modellare *credenze* e *conoscenza* si un'insieme di agenti $Agents = \{a, b, \ldots\}$ Cerchiamo di dare un'interpretazione intuitiva agli assiomi della logica modale per capire cosa ci serve! Utilizzeremo gli operatori

- $\mathbf{K}_a$=*'a' knows that* (es. $\mathbf{K}_{john} fatherOf(john, bill)$)
- $\mathbf{B}_a$=*'a' believes that* (es. $\mathbf{B}_{john} fatherOf(jack, bill)$)

Gli assiomi per modellare le proprietà della conoscenza possono essere scelti tra formule quali:

- $\mathbf{K}_a(F \supset G) \supset (\mathbf{K}_a F \supset \mathbf{K}_a G)$: *se 'a' conosce F e sa che da F segue G, allora conosce anche G*
- $\mathbf{K}_a F \supset F$: *Ciò che è conosciuto da 'a' è vero* (un agente non può conoscere qualcosa che è falso)
- $\mathbf{K}_a F \supset \mathbf{K}_a\mathbf{K}_a F$: *se F è noto ad 'a', 'a' sa di conoscere F* (introspezione positiva)
- $\neg\mathbf{K}_a F \supset \mathbf{K}_a\neg\mathbf{K}_a F$: *se F non è noto ad 'a', 'a' sa di non conoscere F* (introspezione negativa)
- $\mathbf{K}_a F \supset \mathbf{K}_b F$: *'b' sa tutto quello che sa 'a'*

Alcuni esempi di assiomi per modellare le proprietà delle credenze sono

- $\neg\mathbf{B}_a(false)$: *'a' non crede alle contraddizioni*

- **Non** è ragionevole considerare $\mathbf{B}_a F \supset F$: *Non è detto che ciò che è creduto da 'a' sia vero!*
- $\mathbf{B}_a F \supset \mathbf{B}_a \mathbf{B}_a F$: *se 'a' crede F, 'a' crede di credere F*
  (introspezione positiva)
- $\mathbf{B}_a F \supset \mathbf{K}_a \mathbf{B}_a F$: *se 'a' crede F, 'a' sa di credere F*
- $\mathbf{B}_a \mathbf{B}_b F \supset \mathbf{B}_a F$: *se 'a' crede che 'b' crede F, allora 'a' crede F*

Dopo aver assiomatizzato gli operatori si ragiona su conoscenza modellata tramite essi tramite sistemi deduttivi specializzati. Esistono strategie di risoluzione per logiche multi-modali simile a quelle per logica del prim'ordine. Tra le possibili applicazioni troviamo

- Ragionamento non-monotono (frame axiom)
- Specifica di sistemi ad agenti
- Verifica di proprietà di protocolli di sicurezza (dove si modella la conoscenza incompleta)
- ...

7.3 Logica Dinamica

La logca dinamica è una logica modale più viciona ad aspetti che ci interessano da vicino, cioè aspetti computazionali di sistemi informativi. In particolare la logica dinamica è stata introdotto per ragionare sul comportamento di programmi sequenziali. Più precisamente l'idea è di costruire un modello di Kripke in cui:

- i mondi sono tutti i possibili stati del programma
- la relazione di accessibilità = relazione di transizione indotta dalla semantica delle istruzioni (R={ R_G: G istruzione }, R_G=relazione di transizione da stati in stati associata a G)

Consideriamo ora uno statement S di un linguaggio imperativo. Introduciamo due operatori modali $\mathbf{L}_S$ e $\mathbf{M}_S$ per ragionare sul comportamento durante l'esecuzione di S. La formula $\mathbf{M}_S A$ sarà vera nello stato s se esiste uno stato s' raggiungibile da s attraverso l'esecuzione (terminante) di S. Sulla base di questa idea è possibile esprimere varie proprità di un programma attraverso opportune formule modali. Ad esempio,

- $M \models A \supset \mathbf{L}_G B$: correttezza parziale del programma G rispetto ad A e B
- $M \models \mathbf{M}_G A$: esistenza di un cammino che termina con uno stato che soddisfa A per il programma G
- $M \models B \supset \mathbf{M}_G A$: esistenza di un cammino che termina con uno stato che soddisfa A per il programma G sotto l'ipotesi B

Vediamo un esempio concreto.

Example 5 *Consideriamo un programma con la sola istruzione S definita come $x := x + 1$. La semantica associata a tale istruzione induce la seguente relazioni tra stati del programma: $R_S = \{\langle x, x' \rangle \mid x' = x + 1\}$*
R_S è la nostra elazione di accessibilità sull'insieme dei mondi (assegnamenti di valori alla variabile x) La formula $x = 0 \supset \mathbf{L_S} x = 1$ esprime il fatto che se il programma termina (la relazione di accessibilità è parziale) allora a partire dallo stato con $x = 0$ giungerò sempre allo stato $x = 1$.

7.4 Logica Temporale

L'obbiettivo (ambizioso) della logica temporale è formalizzare il concetto di *tempo*. La logica temporale trova numerose appliocazioni sia in logica filosofica che in intelligenza artificiale (linguaggio naturale, rappresentazione di informazione temporale) e in informatica (database temporali, verifica di hardware e sistemi reattivi). Data la difficoltà del problema esistono numerose variazioni sulla logica temporale:

- logica temporale come logica modale (tempo implicito nella semantica)
- logica temporale come logica dei predicati (tempo esplicito)

La semantica dipende dal *flusso* del tempo (finito, infinito, denso, continuo, branching, linear,...), e dall'*unità di tempo* (punti, intervalli, eventi, ...)

Logica Temporal come Modal-logic: Tense Logic La Tense Logic (tense=tempo nel senso 'grammaticale') venne introdotto da Prior per modellare *passato* e *futuro* tramite i seguenti operatori modali

- $\mathbf{P}\varphi$: in qualche instante nel passato φ
- $\mathbf{F}\varphi$: in qualche instante nel futuro φ
- $\mathbf{H}\varphi$: φ è *sempre* stata vera
- $\mathbf{G}\varphi$: φ sarà *sempre* vera

P e **F** sono chiamati weak tense operators; **H** e **G** sono chiamati strong tense operators. Alcuni esempi di Assiomi della Tense Logic sono:

- *Quello che varrà sempre, prima o poi sarà vero!*

$$\mathbf{G}\varphi \supset \mathbf{F}\varphi$$

- Se prima o poi φ, allora accadrà che primo o poi φ

$$\mathbf{F}\varphi \supset \mathbf{FF}\varphi$$

- Se p è vero, allora è sempre stato per essere vero...

$$\varphi \supset \mathbf{HF}\varphi$$

- Se p è vero, allora in futuro sarà 'stato vero'...

$$\varphi \supset \mathbf{GP}\varphi$$

Come in logica modale, la quantificazione al I ordine in Tense Logic introduce più potere espressivo ma anche svariati problemi sia concettuali che pratici. Ad es. proviamo a modellare il concetto *un filosofo sarà re!*. Ci sono varie possibilità:

- $\exists x.(Philosopher(x) \wedge \mathbf{F}King(x))$
- $\exists x.\mathbf{F}(Philosopher(x) \wedge King(x))$
- $\mathbf{F}\exists x.(Philosopher(x) \wedge \mathbf{F}King(x))$
- $\mathbf{F}\exists x.(Philosopher(x) \wedge King(x))$

Nelle ultime due formule occorre un dominio di interpretazione dipendente dal tempo (ad es. $\forall t.\exists x.p(x) \rightarrow$(skolem) $p(f(t))$!)

Semantica della Tense Logic Come in logica modale introduciamo la nozione di temporal frame M definito sulla struttura $\langle T, < \rangle$ (*flow of time*) dove:

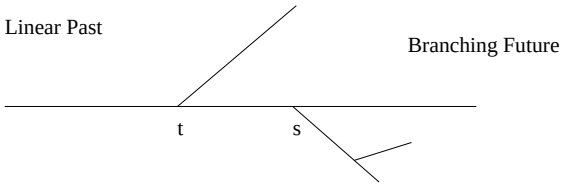
- T= eventi, $<$ ordinamento degli eventi
- $M, t \models \mathbf{P}\varphi$ sse $\exists t'\ t' < t$ such that $M, t' \models \varphi$
- $M, t \models \mathbf{F}\varphi$ sse $\exists t'\ t < t'$ such that $M, t' \models \varphi$
- $M, t \models \mathbf{H}\varphi$ sse $\forall t'\ t' < t$ implica $M, t' \models \varphi$
- $M, t \models \mathbf{G}\varphi$ sse $\forall t'\ t < t'$ implica $M, t' \models \varphi$

Inoltre introduciamo le seguenti proprietà $\mathbf{P}\varphi \equiv \neg\mathbf{H}\neg\varphi$ e $\mathbf{F}\varphi \equiv \neg\mathbf{G}\neg\varphi$.

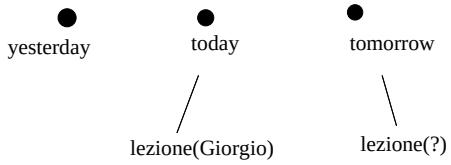
Assiomi e nozione di tempo Come in logica modale esiste una corrispondenza tra assiomi in logica temporale, formule del primo ordine su $<$ (proprietà dell'ordinamento) e classi di frames:

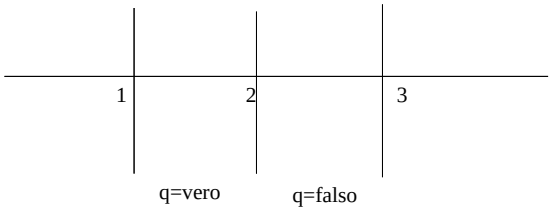
$-\mathbf{H}\varphi \supset \mathbf{P}\varphi$	$\forall t.\exists t'.(t' < t)$
	='unbounded in the past'
$-\mathbf{G}\varphi \supset \mathbf{F}\varphi$	$\forall t.\exists t'.(t < t')$
	='unbounded in the future'
$-\mathbf{F}\varphi \supset \mathbf{FF}\varphi$	$\forall t, t'.(t < t' \supset \exists t''.(t < t'' < t'))$
	='dense ordering'
$-\mathbf{FP}\varphi \supset \mathbf{P}\varphi \vee \varphi \vee \mathbf{F}\varphi$	$\forall t, t', t''.((t < t'' \wedge t' < t'') \supset (t < t' \vee t = t' \vee t' < t))$
	='linear in the past'
$-\mathbf{PF}\varphi \supset \mathbf{P}\varphi \vee \varphi \vee \mathbf{F}\varphi$	$\forall t, t', t''.((t'' < t \wedge t'' < t') \supset (t < t' \vee t = t' \vee t' < t))$
	='linear in the future'

Esempi di Logiche Temporali Vediamo alcuni esempi di logiche temporali. Supponiamo di voler formalizzare l'analisi grammaticale di frasi coniugate al tempo futuro: *il treno partirà domani*, *il treno parte domani*, ... Potremmo pensare che le differenze semantiche dipendono da possibili *futuri* diversi (nel punto t ho un futuro T_t ecc) Scegliamo quindi future *branching time*:



Il tempo discreto può essere utilizzato anche per descrivere eventi su base *giornaliera*. In questo caso il tempo è modellato attraverso gli interi (ordine totale). L'unità di tempo è quindi un intero e gli atomi hanno valori di verità ad ogni punto.





Invece di punti possiamo pensare di utilizzare come unità di tempo un intervallo chiuso $[m, n]$, $m \leq n$. In questo caso gli atomi sono veri/falsi rispetto ad un certo intervallo. In seguito studieremo con maggiore attenzione due particolari tipi di logica temporale utilizzata per studiare aspetti conmputazionali, CTL e LTL. Vediamo prima altri esempi di logiche modali.

7.5 Logica Spaziale

La logica spaziale permette di descrivere insiemi di oggetti geometrici e relazioni topologiche. Tra le sue possibili applicazioni troviamo image processing visual databases, robotics, e reti e codice mobile.

Esempio di Logica Spaziale: Compass relation Consideriamo il piano dei reali $\mathbf{R} \times \mathbf{R}$ come una mappa infinita. Definiamo la seguente *compass relation*:

$\langle x, y \rangle R_E \langle x', y' \rangle \; iff \; x < x', y = y'$ $\langle x', y' \rangle$ ad Est di $\langle x, y \rangle$
$\langle x, y \rangle R_N \langle x', y' \rangle \; iff \; x = x', y < y'$ $\langle x', y' \rangle$ a Nord di $\langle x, y \rangle$
...

La mappa infinita si può vedere come un modello di Kripke $\langle \mathbf{R} \times \mathbf{R}, R_E, R_W, R_N, R_S \rangle$. e ragionare attraverso connettivi multimodali quali $\mathbf{L}_E, \mathbf{L}_W, \mathbf{L}_W, \ldots$ e assiomi quali

$$\mathbf{L}_E \mathbf{L}_N p \supset \mathbf{L}_N \mathbf{L}_E p$$

Esempio di Logica Spaziale: RCC La *compass relation* ci permette di ragionare su punti ma non su *regioni*. Per ovviare a questa limtazione sono state proposte logiche per rappresentare regioni. Tra queste troviamo il *Region Connection Calculus* (RCC). RCC è una logica del I ordine con un solo predicato

$$\mathrm{C(X, Y)} = \text{ region X is connected to region Y}$$

dal quale si ricavano le relazioni topologiche principali. RCC è molto espressivo (provaibilità è indecidibile). Tuttavia esistono restrizioni interessanti che sono trattatibili. Alcuni frammenti del Region Connection Calculus possono essere rappresentati in particolari logiche modali. A partire dalla realziione C si possono derivare relazioni topologiche quali:

X, Y *disconnected*	$CD(X, Y) \equiv \neg C(X, Y)$
X *part of* Y	$P(X, Y) \equiv \forall Z.(C(Z, X) \supset C(Z, Y))$
X *equal to* Y	$EQ(X, Y) \equiv P(X, Y) \wedge P(Y, X)$
X *overlaps* Y	$O(X, Y) \equiv \exists Z.(P(Z, X) \wedge P(Z, Y))$
X *partially overlaps* Y	$PO(X, Y) \equiv O(X, Y) \wedge \neg P(X, Y) \wedge \neg P(Y, X))$
...	

8 Esercizi

8.1 Soundness della Necessitation Rule

- In logica modale la *necessitation rule* (NR) è definita come segue:
  se $\vdash F$ *allora* $\vdash \mathbf{L}F$
- Dimostrare che NR è sound rispetto alla *semantica generalizzata*

Soluzione

- Soundness di $\frac{A}{B}$ = se A è valida allora B è valida
- Supponiamo che A sia valida.
  Allora per ogni modello $M = \langle W, D, R, L\rangle$ e mondo $w \in W$ $M, w \models A$.
  Supponiamo ora che $\mathbf{L}A$ non sia valida.
  Allora esiste un modello M ed un mondo w tale che $M, w \models \mathbf{L}A$ *non* vale,
  cioè $\exists v \in W$ tale che wRv e $M, v \not\models A$ e ciò contraddice l'ipotesi che A è valida (cioè vera in ogni modello e ogni stato)

8.2 Deduzione in Logica Modale

- Consideriamo il seguente sistema deduttivo:
- **Assiomi:**
  Def$_M$. $\mathbf{L}A \equiv \neg\mathbf{M}\neg A$
  T. $\mathbf{L}A \supset A$
  K. $\mathbf{L}(A \supset B) \supset (\mathbf{L}A \supset \mathbf{L}B)$
  5. $\mathbf{M}A \supset \mathbf{LM}A$
  PL. A, se A è una tautologia in logica classica
- **NR: Necessitation Rule:** Se A, allora $\mathbf{L}A$
- **MP: Modus Ponens:** Se $A \supset B$ e A, allora B.

- Provare che $\vdash \mathbf{LM}A \supset \mathbf{LLMA}$
- Verificare tramita la semantica di S5 (in cui R è una relazione d'equivalenza) che tale formula è valida

Soluzione:

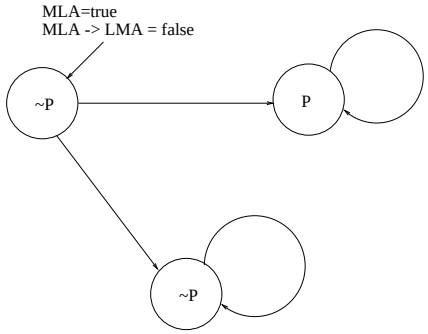
$$\dfrac{\dfrac{\dfrac{}{\mathbf{M}A \supset \mathbf{LM}A}\,5}{\mathbf{L}(\mathbf{M}A \supset \mathbf{LM}A)}\,NR \qquad \dfrac{}{\mathbf{L}(\mathbf{MA} \supset \mathbf{LM}A) \supset (\mathbf{LM}A \supset \mathbf{LLM}A)}\,K}{\mathbf{LM}A \supset \mathbf{LLM}A}\,MP$$

Soluzione

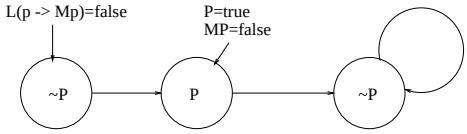
- In S5 i modelli sono tuple $M = \langle W, D, L \rangle$ (R è una relazione di equivalenza)
- $\varphi = \mathbf{LM}A \supset \mathbf{LLMA}$ è valida se per ogni M e w $M, w \models \varphi$
- Supponiamo che $M, w \models \mathbf{LM}A$
- Quindi per ogni $v \in W$ $M, v \models \mathbf{M}A$, i.e., esiste $u \in W.M, u \models A$
- Quindi $\forall v_1.\forall v_2.\exists w.\ M, w \models A$ cioè **LLMA** è vera in M, w

8.3 Modelli di Kripke

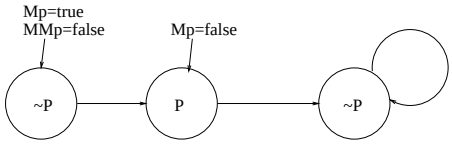
- Costruire dei modelli di Kripke che falsificano le seguenti formule
- $\mathbf{ML}A \supset \mathbf{LM}A$
- $\mathbf{L}(A \supset \mathbf{M}B)$
- $\mathbf{M}A \supset \mathbf{MM}A$



Modello che falsifica $\mathbf{ML}A \supset \mathbf{LM}A$



Modello che falsifica $\mathbf{L}(A \supset \mathbf{M}B)$



Modello che falsifica $\mathbf{M}A \supset \mathbf{MM}A$

8.4 Logica degli Alberi

- Supponiamo di voler definire una logica modale per ragionare su strutture ad albero. Supponiamo quindi che i nostri modelli siano alberi binari etichettati con due colori *bianco e nero*
- Supponiamo di voler esprimere proprietà quali
  - Esiste un cammino con soli nodi bianchi
  - L'albero ha una frontiera con soli nodi neri
  - L'albero ha solo nodi neri
  - Esiste un sottoalbero con nodi solo neri
  - Se c'e' un nodo bianco allora tutti i suoi successori sono neri
- A partire da tali modelli di Kripke introdurre i connettivi per poter definire tali proprietà, definire la loro semantica, e infine esprimere le proprietà nella logica risultante

8.5 Logica Temporale

- Considerate un modello di Kripke in cui gli stati sono i numeri interi e la relazione di accessibilità è la relazione successore
- Definiamo i connettivi **G** ed **F** come segue

  Gφ è vera se φ vale in tutti gli stati raggiungibili dallo stato corrente

  Fφ è vera se esiste uno stato raggiungibile dallo stato corrente in cui φ è vera
- Mostrare un modello con le caratteristiche viste sopre che soddisfa **GF**φ ma non **FG**φ
- Intuitivamente quali concetti rappresentano **FG**φ e **GF**φ

Esempio di modelli

- **GF**φ vale se in ogni stato esiste uno stato successore che soddisfa φ in altre parole φ deve valere infitamente spesso
- **FG**φ vale se esiste uno stato successore a partire dal quale φ vale sempre coè φ è persistente
- Il modello in cui $L(i) = p$ sse i è dispari, cioè

$$0 \rightarrow \overset{p}{1} \rightarrow 2 \rightarrow \overset{p}{3} \rightarrow 4 \rightarrow \overset{p}{5} \ldots$$

  soddisfa $GF\ p$ ma non $FG\ p$. Infatti non esiste uno stato k a partire dal quale p vale sempre, mentre p vale infinitamente spesso nel modello (ogni sue stati)

9 Inciso su Logica Costruttiva

Per i *costruttivisti* la logica classica non è adeguata per modellare aspetti computazionali. La logica classica infatti è instrinsicamente *non costruttiva*, i.e., non fornisce delle prove per la validità di una formula. Esempio classico: $A \vee \neg A$ è sempre vero indipendentemente dalla struttura di A. E se A fosse una computazione che non termina? Logiche costruttive si presentano quindi come possibili alternative alla logica classica. In questa sezione vedremo alcune aspetti intuitivi legati a due classi di logiche costruttive: la logica intuizionista, e le logiche sub-strutturali. Vedremo questi argomenti seguendo un approccio guidato dalla sintassi (proof theory).

9.1 Preliminari: Calcoli a Sequenti

I calcoli a sequenti sono una classe particolare di sistemi deduttivi per logica classica definiti come segue. Siano Δ (antecedente) e Γ (conseguente) due *liste* di formule, un *sequente* è una coppia $\Delta \longrightarrow \Gamma$. Intuitivamente, il sequente

$$B_1, \ldots, B_n \longrightarrow C_1, \ldots, C_m$$

rappresenta la formula

$$(B_1 \wedge \cdots \wedge B_n) \supset (C_1 \vee \cdots \vee C_m).$$

I sistemi inferenziali per i sequenti presentano regoale come quelle mostrate qui di seguito

$$\vee \quad \frac{A, \Gamma \longrightarrow \Delta \quad B, \Gamma \longrightarrow \Delta}{A \vee B, \Gamma \longrightarrow \Delta} \qquad \frac{\Gamma \longrightarrow A, \Delta}{\Gamma \longrightarrow A \vee B, \Delta} \qquad \frac{\Gamma \longrightarrow B, \Delta}{\Gamma \longrightarrow A \vee B, \Delta}$$

$$\wedge \quad \frac{\Gamma \longrightarrow A, \Delta}{\Gamma, A \wedge B \longrightarrow \Delta} \qquad \frac{\Gamma \longrightarrow B, \Delta}{\Gamma, A \wedge B \longrightarrow \Delta} \qquad \frac{\Gamma \longrightarrow A, \Delta \quad \Gamma \longrightarrow B, \Delta}{\Gamma \longrightarrow A \wedge B, \Delta}$$

$$\supset \quad \frac{A, \Gamma \longrightarrow B, \Delta}{\Gamma \longrightarrow A \supset B, \Delta} \qquad \frac{B, \Gamma \longrightarrow \Delta \quad \Gamma \longrightarrow A, \Delta}{\Gamma, A \supset B \longrightarrow \Delta}$$

$$\neg \quad \frac{A, \Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \neg A, \Delta} \qquad \frac{\Gamma \longrightarrow A, \Delta}{\Gamma, \neg A \longrightarrow \Delta}$$

Gli assiomi hanno la seguente forma:

$$\frac{}{A, \Gamma \longrightarrow A, \Delta} \qquad \frac{}{false, \Gamma \longrightarrow \Delta} \qquad \frac{}{\Gamma \longrightarrow true, \Delta}$$

Tutte le regole precendenti vengono chiamate *regole logiche*. Oltre ad esse occorre introdurre delle regole per manipolare sintatticamente i sequenti durante la costruzione di una derivazione. Queste regole vengono dette strutturali e hanno la seguente forma:

$$\frac{\Gamma \longrightarrow A, A, \Delta}{\Gamma \longrightarrow A, \Delta} \qquad \frac{A, A, \Gamma \longrightarrow \Delta}{A, \Gamma \longrightarrow \Delta} \qquad \frac{\Gamma \longrightarrow \Delta}{\Gamma \longrightarrow A, \Delta} \qquad \frac{\Gamma \longrightarrow \Delta}{A, \Gamma \longrightarrow \Delta}$$

$$\frac{\Gamma \longrightarrow \Delta, B, A, \Delta'}{\Gamma \longrightarrow \Delta, A, B, \Delta'}$$

Nota: con le regole strutturali Γ e Δ vengono considerati *insiemi* di formule Una *prova* è un albero costruito tramite le regole di inferenza le cui foglie sono tutti assiomi e la cui radice è il sequente iniziale (da dimostrare).

Ad esempio, supponiamo di voler dimostrare che $A \vee \neg A$, la legge del terzo escluso (*law of excluded middle*) è un teorema della logica classica. Possiamo costruire la seguente derivazione:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \longrightarrow A}\ assioma}{A \longrightarrow A \vee \neg A}\ \vee R}{\longrightarrow \neg A, A \vee \neg A}\ \neg R}{\longrightarrow A \vee \neg A, A \vee \neg A}\ \vee R}{\longrightarrow A \vee \neg A}\ contrazione$$

Notate che nella prova di $A \vee \neg A$ dobbiamo *necessariamente* utilizzare la regola di contrazione a destra

9.2 Logica Intuizionista

La logica intuizionista rappresenta un'alternativa alla logica classica, dove, ad esempio, la legge del terzo escluso (LEM) $A \vee \neg A$ non vale più. Nel 1908 Brouwer osservò che LEM seppur ragionevole per modellare proprietà per collezioni *finite* non è sempre ragionevole quando esteso a proprietà per collezioni *infinite*. Ad esempio, se x, y variano sugli interi naturali $0, 1, 2, \ldots$ e $B(x) =$*esiste* $y > x$ *tale che sia* y *che* $y + 2$ *sono numeri primi*, allora:

- non abbiamo ancora un metodo generale (algoritmo) che dato un x arbitrario decide se $B(x)$ è vero o falso!
- Quindi $\forall x.(B(x) \vee \neg B(x))$, allo stato attuale della nostra conoscenza, non può essere vera!
- Inoltre se A abbrevia l'asserzione $\forall x.B(x)$, nuovamente non possiamo dire nulla su $(A \vee \neg A)$ perchè A e $\neg A$ non sono ancora state provate.

La logica classica non è costruttiva! Non restituisce *testimoni* per la verità di una formula

In logica intuizionista, la formula $(A \vee B)$ va letta come:

sappiamo costuire una prova per A *oppure una prova per* B*.*

Una prova per A è il testimone che A è vera. Per ottenere un sistema deduttivo con tali caratteristiche si considerano restrizioni *sintattiche* e logiche *substrutturali*

Sistema di Prova in Logica Intuizionista Si considerano sequenti in cui il conseguente contiene al più una formula!

$$\Gamma \longrightarrow A$$

dove Γ è un insieme di formule ed A è una formula. In altre parole si vieta l'applicazione delle regole strutturali a *destra* in un sequente. Per provare $A \vee B$ devo costruire una prova per A o una prova per B. Quindi la regola inferenziale di introduzione di $\vee$ a destra diventa

$$\dfrac{\Gamma \longrightarrow A}{\Gamma \longrightarrow A \vee B}\ \vee R \qquad\qquad \dfrac{\Gamma \longrightarrow B}{\Gamma \longrightarrow A \vee B}\ \vee R$$

Secondo Miller et al. la logica intuizionista può essere utilizzata per interpretazioni *computazionali* della *provabilità*. Ad esempio, il provare il sequente $P \longrightarrow append(L, L', M)$ in logica intuizionista può essere interpretato come: provare il goal $append(L, L', M)$ in un programma logico P Vietare l'uso delle regole strutturali corrisponde all'idea di considerare in un sequente $\Gamma \longrightarrow A$ la formula A come un 'task'

- non si può *duplicare* arbitrariamente (come in logica classica)
- abbiamo bisogno di un testimone per la terminazione delle sue *attività*

Logica intuizionista e programmazione logica Considerate il goal $append(L, L', M)$. In logica classica è equivalente alla formula

$$append(L, L', M) \vee append(L, L', M)$$

Tuttavia, *computazionalmente*, $append(L, L', M)$ ha un effetto ben diverso dal goal

$$append(L, L', M) \vee append(L, L', M)$$

Pensate infatti al ruolo della disgiunzione (scritta ;) nel meccanismo di backtracking del Prolog: nel goal $A; B$ se A fallisce si esegue B. Nel nostro caso rischiamo di eseguire due volte la stessa computazione. Per evitare questo tipo di problemi, si potrebbe scegliere la logica intuizionista come fondamento logico della programmazione logica.

Approfondimenti: Provabilità Uniforme in Logica Intuizionista, Clausole Ereditarie di Harrop, e lavori di Dale Miller su λProlog.

9.3 Altre Logiche Substrutturali

La logica intuizionista vieta l'applicazione delle regole strutturali a destra nei sequenti Più in generale si parla di logiche substrutturali quando si ammettono solo alcune delle regole strutturali Le logiche strutturali sono interessanti perchè introducono altri aspetti interessanti per la modellazione di sistemi informativi tramite teorie logiche Esempi: Relevant Logic, Affine Logic, Logica Lineare

Logica Lineare e Risorse Consideriamo la regola *modus ponens*

$$\frac{A \quad A \supset B}{B} \; MP$$

notiamo che A vale ancora *dopo* l'applicazione della regola! Nel mondo reale le *risorse* vengono consumate con l'uso!

$$\frac{A = \text{“Ho 0.26Euro”} \quad A \supset B = \text{“Se ho 0.26Euro, compro un caffè”}}{B = \text{“Compro un caffè” (ma ho consumato 0.26Euro!!)}} \; MP$$

Per introdurre una nozione di *risorsa* ancora più forte che in logica intuizionista, in logica lineare si vieta l'uso delle regole strutturali di contrazione e weakening sia a destra che a sinistra! Di conseguenza durante la costruzione di una prova ogni formula si può utilizzare al più una volta! Per non limitare troppo la potenza espressiva della logica si utilizzano dei connettivi speciali (chiamati esponenziali) per re-introdurre aspetti di logica classica: le formule con l'esponenziale si comnportano che formule della logica classica, le formule senza esponenziali sono invece delle risorse!

Sistema di Prova in Logica Lineare I connettivi sono suddivisi in due categorie:

- *addittivi*: e.g., & (congiunzione), $\oplus$ (disgiunzione), ecc.;
- *moltiplicativi*: e.g., $\otimes$ (congiunzione), $\parr$ (disgiunzione), $\multimap$ (implicazione) etc.

Si considerano sequenti sia con componenti classiche (formule $!F$) che lineari

$$!\Gamma, \Omega \longrightarrow \Delta$$

dove $\Gamma = \{F_1, F_2, \ldots\}$ è un insieme di formule, $!\Gamma = !F_1, !F_2, \ldots$

Esempi di Regole in Logica Lineare Un sequente ha componenti classiche (formule !F) e lineari

$$\frac{!\Gamma, \Omega \longrightarrow A, B, \Delta}{!\Gamma, \Omega \longrightarrow A \parr B, \Delta}\ \parr R \qquad \frac{!\Gamma, \Omega \longrightarrow A, \Delta \quad !\Gamma, \Omega \longrightarrow B, \Delta}{!\Gamma, \Omega \longrightarrow A \,\&\, B, \Delta}\ \& R$$

dove $!\{F_1, F_2, \ldots\} = !F_1, !F_2, \ldots$

Interpretazione Computazionale di Logica Lineare Secondo Andreoli, Miller et al. la logica lineare può essere utilizzata per interpretazioni *computazionali* con aspetti di *concorrenza* e *gestione dello stato* Ad esempio, provare il sequente $!P, R \longrightarrow G_1 \parr G_2, \Delta$ in logica lineare può essere interpretato come:

> provare in concorrenza i goal G_1 e G_2 in un ambiente Δ, usando il programma P come definizione dei processi (utilizzabili più volte), e le formule in R come risorse (utilizzabili una sola volta)

Approfondimenti: Provabilità Logica Lineare (Focusing Proofs), Logica Lineare e Calcoli di Processi, Programmazione Logica Lineare.

9.4 Esercizi

- Chiamiamo una formula *multiheaded* se ha le seguente forma

  $$G \supset (A_1 \vee \ldots \vee A_k)$$

  dove A_i è una formula atomica, e G (goal) è costruita tramite formule atomiche, $\wedge$ e $\vee$

- Considerate ora sequenti del tipo $\Gamma \longrightarrow \Delta$ dove Γ è un *insieme* di formule *multiheaded* e Δ è un **multinsieme** di *goal* (cioè $p \wedge q, p \wedge q$ è diverso da $p \wedge q$)

Logiche substrutturali

- Considerate ora il seguente sistema di prova

$$\frac{\Gamma \longrightarrow A, B, \Delta}{\Gamma \longrightarrow A \vee B, \Delta} \qquad \frac{\Gamma \longrightarrow A, \Delta \quad \Gamma \longrightarrow B, \Delta}{\Gamma \longrightarrow A \wedge B, \Delta} \qquad \frac{}{\Gamma \longrightarrow \top, \Delta}$$

$$\frac{\Gamma \longrightarrow G, \Upsilon}{\Gamma, G \supset (A_1 \vee \ldots \vee A_n) \longrightarrow A_1, \ldots, A_n, \Upsilon}\ \Upsilon \textit{ un multins. di formule atomiche}$$

- Provate che la regola di weakening a destra (vedi sotto) è ammissibile nel sistema di prova precedente

$$\frac{\Gamma \longrightarrow \Delta}{\Gamma \longrightarrow A, \Delta}$$

La logica *modale* è stata sviluppata per studiare vari modelli di *verità* (necessariamente e possibilmente vero). La logica temporale è un particolare tipo di logica modale nella quale la verità di un'asserzione può variare nel *tempo*. Gli operatori modali che tipicamente troviamo in logica temporale sono

- *Prima o poi* Φ: in qualche istante nel futuro Φ è vera
- *Per sempre* Φ: Φ sarà vera in ogni possibile istante futuro

Tra le possibile applicazione troviamo la rappresentazione della dinamica nella conoscenza. Ad eesmpio per modellare le proprietà interessanti per un *semaforo*:

- una volta diventato rosso, non può tornare immediatamente verde
- prima o poi sarà di nuovo verde
- una volta rosso, diventa verde dopo essere rimasto giallo per un certo periodo (tra il rosso e il verde)

Inoltre la logica temporale è particolarmente adatta per specificare proprietà del comportamento di sistemi che si evolvono nel tempo. Ad esempio, *modelli* di *programmi software* ma anche *protocolli di comunicazione* e *sistemi hardware*. Più in generale si parla di *sistemi reattivi* cioè sistemi che interagiscono continuamente con un ambiente (ad esempio un *server* di un protocollo). Ad esempio, in un protocollo di comunicazione tra un agente S ed un agente R potrebbe essere fondamentale garantire proprietà quali:

- Se il processo S invia un messaggio, allora non invierà altri messaggi prima di ricevere un acknowledgement
- I due agenti non saranno mai contemporaneamente in uno stato di attesa di un messaggio (i.e. possibile deadlock)

Questo tipo di proprietà non sono facilmente modellabili tramite linguaggi di specifica basati su *pre* e *post* condizioni, solitamente utilizzate per specificare proprietà della relazione *input-output* di un programma sequenziale

Esistono vari tipi di logiche temporali

- Linear Temporal Logic (LTL) dove il futuro è formato da un insieme di possibili cammini: le proprietà sono definite sui *singoli percorsi*
- Computation Tree Logic (CTL) dove il futuro ha forma ad albero: le proprietà sono definite sia in *ampiezza* che sui *singoli cammini*

LTL e CTL non sono *ugualmente espressive*, cioè esistono formule di una logica non esprimibili nell'altra (e.g. esiste una formula F in CTL tale che non esiste una formula G in LTL soddisfacibile sugli stessi modelli e stati iniziali su cui F è soddisfacibile, e viceversa). LTL e CTL sono sussunte dalla logica temporale chiamata CTL*.

Fissata una struttura di Kripke M (ad esempio il modello del comportamento di un sistema

reattivo), uno *stato iniziale* s_0, e data una proprietà specificata in logica temporale φ (che ogni possibile esecuzione del protocollo dovrebbe verificare) si vuole decidere se

$$M, s_0 \models \varphi \text{ (cioè se } \varphi \text{ è vera in } M \text{ e } s_0)$$

Questo problema viene chiamato *model checking*. Esistono *algoritmi* che permettono di decidere tale problema sia per CTL che per LTL proposizionale. Come vedremo in seguito tali algoritmi si basano sulla teoria e sugli algoritmi sviluppata per i *grafi*.

Esistono vari approcci

- Metodi basati su *tableux*: si colora il modello di Kripke con le sotto-formule vere in ogni nodo tramite delle regole guidate dalla sintassi della formula
- Metodi basati su computazioni annidate di *punto fisso*: si spezza il problema in blocchi annidati e si risolve tramite calcoli di punto fisso
- Metodi basati su *teoria degli automi*: si sfrutta un'interessante collegamento tra logica temporale e proprietà dei linguaggi regolari (automi su stringhe finite ed infinite)

Nel seguito vedremo solo alcuni di questi (tableaux e fixpoint-based).

I tool esistenti utilizzano strutture date *efficienti* per rappresentare sia il modello di Kripke che i risultati intermedi calcolati dalle procedure di decisione. Ad esempio, la tecnica chiamata Symbolic Model Checking sfrutta gli *Alberi Binari di Decisione* (*Binary Decision Diagrams*, BDDs) per rappresentare in modo *compatto* lo spazio degli stati di un sistema finito (BDD=formula Booleana proposizionale). Alcuni esempi di sistemi basati sulle tecniche di model checking:

- SMV: http://www-cad.eecs.berkeley.edu/ kenmcmil/smv/
- NuSMV: http://nusmv.irst.itc.it/
- FormalCheck: http://www.cadence.com/datasheets/formalcheck.html
- XMC (logic programming-based): http://xsb.sourceforge.net/
- UPPAAL (real-time systems): http://www.uppaal.com/

Iniziamo a vedere più in dettaglio le caratteristiche fondamentali di LTL e CTL.

10 Linear Temporal Logic (LTL)

La logica LTL è costituita dalle seguenti formule.

- Proposizioni Atomiche: simboli di predicato $p, q, r, \ldots$
- Connettivi classici:
$$\neg\psi \quad \varphi \wedge \psi \quad \varphi \vee \psi \quad \varphi \supset \psi$$
- Connettivi temporali: **X**, **F**, **G**, **U**

$$\begin{aligned} \mathbf{X}\varphi &= \text{nell'istante successivo/next } \varphi \\ \mathbf{F}\varphi &= \text{primo o poi (in the future/eventually) } \varphi \\ \mathbf{G}\varphi &= \text{sempre (globally/always) } \varphi \\ \varphi\mathbf{U}\psi &= \varphi \text{ fino a che (until) } \psi \end{aligned}$$

- Una formula senza connettivo temporale al top-level si riferisce all'istante corrente

L'insieme di connettivi $\neg$, $\vee$, $\mathbf{X}$, and $\mathbf{U}$ insieme alle formule atomiche è sufficiente per definire le formule LTL Infatti

$$\begin{aligned}\mathbf{F}\varphi &\equiv \mathit{true}\ \mathbf{U}\ \varphi\\ \mathbf{G}\varphi &= \neg\mathbf{F}\neg\varphi\end{aligned}$$

Example 6 *Vediamo come si possono utilizzare formule LTL per rappresentare le proprietà del nostro semaforo.*

- *una volta diventato rosso, non può tornare immediatamente verde*

$$\mathbf{G}(\mathit{red} \supset \neg\mathbf{X}\mathit{green})$$

- *prima o poi sarà di nuovo verde*

$$\mathbf{F}\ \mathit{green}$$

- *una volta rosso, diventa verde dopo essere rimasto giallo per un certo periodo (tra il rosso e il verde)*

$$\mathbf{G}(\mathit{red} \supset ((\mathit{red}\ \mathbf{U}\ \mathit{yellow})\ \mathbf{U}\ \mathit{green}))$$

10.1 Semantica Model Theoretic

Un modello LTL è una tripla $M = \langle S, R, L\rangle$ dove

- S è un insieme numerabile non vuoto di stati
- $R : S \to S$ assegna ad ogni stato s un unico successore $R(s)$
- $L : S \to 2^{AP}$ assegna ad ogni stato $s \in S$ le formule atomiche proposizionali che sono vere in s

R permette di creare sequenze infinite di stati (non necessariamente diversi tra loro) $s\ R(s)\ R(R(s))\ \ldots$ (si potrebbe definire un modello direttamente su sequenze infinite)
Fissato $M = \langle S, R, L\rangle$, la relazione $M, s \models \varphi$ (M soddisfa φ in s) è definita come

- $s \models p$ se $p \in L(s)$
- $s \models \neg\phi$ se $s \not\models \phi$
- $s \models \varphi \vee \psi$ se $s \models \varphi$ or $s \models \psi$
- $\ldots$
- $s \models \mathbf{X}\varphi$ se $R(s) \models \varphi$
- $s \models \varphi_1\ \mathbf{U}\ \varphi_2$ se $\exists j \geq 0.\ R^j(s) \models \varphi_2,\ (\forall 0 \leq k < j.\ R^k(s) \models \varphi_1)$

da cui segue che

- $s \models \mathbf{F}\varphi$ se $\exists j. R^j(s) \models \varphi$
- $s \models \mathbf{G}\varphi$ se $\forall j. R^j(s) \models \varphi$

dove $R^j(s) = \underbrace{R(\ldots R(s)\ldots)}_{j}$.

10.2 Esempi

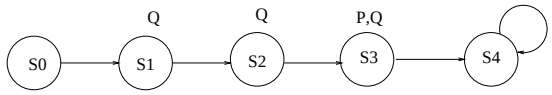
Se $\varphi \mathbf{U} \psi$ vale in s allora anche $\mathbf{F}\psi$ vale in s, cioè quando si asserisce che φ vale fino a che ψ diventa vera, implicitamente si asserische che prima o poi ψ sarà vera. Esiste un'altra tipo di operatore *until* chiamato *weak until* che vale anche se φ rimane vera per sempre:

$$\varphi \ \mathbf{W} \ \psi \ \equiv \ \mathbf{G}\varphi \ \vee \ (\varphi \ \mathbf{U} \ \psi)$$

Sia M il seguente modello ($s \to s'$ significa che $s' = R(s)$)

$$s_0 \overset{\emptyset}{\to} s_1 \overset{q}{\to} s_2 \overset{q}{\to} s_3 \overset{p,q}{\to} s_4 \overset{\emptyset}{\to} s_4 \overset{\emptyset}{\to} s_4 \overset{\emptyset}{\to} \ldots$$

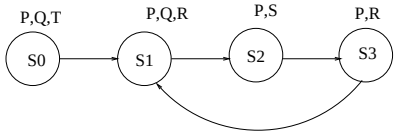
cioè $L(s_0) = \emptyset$, $L(s_1) = \{q\}$, $L(s_2) = \{q\}$, $L(s_3) = \{p, q\}$, $L(s_4) = \emptyset$ allora $\mathbf{X}p$ vale in s_2; $\mathbf{F}p$ vale



in s_0, s_1, s_2, s_3; $\mathbf{G}p$ non vale in nessun stato; $q \ \mathbf{U} \ p$ vale in s_1, s_2, s_3.
Sia M il seguente modello ($s \to s'$ significa che $s' = R(s)$)

$$s_0 \overset{p,q,t}{\to} s_1 \overset{p,q,r}{\to} s_2 \overset{p,s}{\to} s_3 \overset{p,r}{\to} \ (back\ to) \ \to s_1$$

allora $\mathbf{F}t$ vale solo in s_0; $\mathbf{G}p$ vale in ogni stato; $\mathbf{G}\ \mathbf{F}s$ vale in ogni stato; $\mathbf{X}(r \supset (q \ \mathbf{U} \ s))$ vale in



s_0, s_1, s_3.

10.3 Formule LTL interessanti

- $\varphi \supset \mathbf{F}\psi$: se inizialmente vale φ, allora prima o poi vale ψ
- $\mathbf{G}\ \mathbf{F}\varphi$: φ vale 'infinitely often'
- $\mathbf{F}\ \mathbf{G}\varphi$: φ vale 'eventually permanently'
- $\mathbf{G}(\varphi \supset \mathbf{X}\varphi)$: se φ vale in qualche stato, vale anche nello stato successivo (in LTL equivale a $\mathbf{G}(\varphi \supset \mathbf{G}\varphi)$: una volta stabilito φ, φ varrà per sempre)

Notate che $\mathbf{GF}\varphi \ \not\equiv \ \mathbf{F}\varphi$ e $\mathbf{FG}\varphi \ \not\equiv \ \mathbf{G}\varphi$.

10.4 Assiomi per LTL

- Dualità $\quad \neg\mathbf{G}\varphi \equiv \mathbf{F}\neg\varphi \qquad \neg\mathbf{F}\varphi \equiv \mathbf{G}\neg\varphi \qquad \neg\mathbf{X}\varphi \equiv \mathbf{X}\neg\varphi$
- Espansione
$$\varphi \ \mathbf{U} \ \psi \ \equiv \ \psi \ \vee \ [\varphi \ \wedge \ \mathbf{X}(\varphi \ \mathbf{U} \ \psi)]$$
$$\mathbf{F}\varphi \ \equiv \ \varphi \ \vee \ \mathbf{X} \ \mathbf{F}\varphi \qquad\qquad \mathbf{G}\varphi \ \equiv \ \varphi \ \wedge \ \mathbf{X} \ \mathbf{G}\varphi$$

- Idempotenza $\mathbf{G}\ \mathbf{G}\varphi \equiv \mathbf{G}\varphi \qquad \mathbf{F}\ \mathbf{F}\varphi \equiv \mathbf{F}\varphi$
  $\varphi\ \mathbf{U}\ (\varphi \mathbf{U} \psi) \equiv \varphi\ \mathbf{U}\ \psi \qquad (\varphi\ \mathbf{U}\ \psi)\mathbf{U}\psi \equiv \varphi\ \mathbf{U}\ \psi$
- Assorbimento $\mathbf{F}\ \mathbf{G}\ \mathbf{F}\varphi \equiv \mathbf{G}\ \mathbf{F}\varphi \qquad \mathbf{G}\ \mathbf{F}\ \mathbf{G}\varphi \equiv \mathbf{F}\ \mathbf{G}\varphi$
- Commutazione $\mathbf{X}(\varphi\ \mathbf{U}\ \psi)\ \equiv\ (\mathbf{X}\varphi)\ \mathbf{U}\ (\mathbf{X}\ \psi)$

10.5 Model Checking, Soddisfacibilità e Validità

Il *model checking problem* è definito come segue: Dato un modello LTL M, uno stato s_0 e una formula LTL φ $M, s_0 \models \varphi$? Il *satisfiability problem* è definito invece come: Data una formula LTL φ, esiste un modello LTL M, ed uno stato s_0 tale che $M, s_0 \models \varphi$? Infine il *validity problem* è definito come segue Data una formula LTL φ, vale $M, s_0 \models \varphi$ per ogni modello LTL M ed ogni stato s_0? **Importante**: tutti e tre i problemi sono *decidibili* (in particolare l'ultimo sfrutta la proprietà dei *modelli finiti* di LTL)

10.6 LTL Model Checking

Dati M, s, φ, vogliamo decidere se $M, s \models \varphi$. L'algoritmo che risolve il problema è dovuto a Lichtenstein-Pnueli ed è basato sulla costruzione di un *tableau*, cioè un grafo costruito a partire dalla formula dal quale si può estrarre un modello per la formula solo se la formula è soddisfacibile. L'algoritmo che decide se φ segue da M compone il tableau e la struttura M (prodotto). in modo da verificare se esiste un cammino in M che è anche un cammino nel tableau.

Chiusura di una formula LTL Abbiamo visto che possiamo restringere la nostra attenzione a formule solo con $\mathbf{X}$ e $\mathbf{U}$. Sia M un modello LTL e φ una formula LTL, definiamo la chiusura $CL(\varphi)$ di φ come il più piccolo insieme di formule che contiene φ e tale che le formule generiche φ_1, φ_2 soddisfano le seguenti condizioni:

- $\neg\varphi_1 \in CL(\varphi)$ sse $\varphi_1 \in CL(\varphi)$
- se $\varphi_1 \vee \varphi_2 \in CL(\varphi)$ allora $\varphi_1, \varphi_2 \in CL(\varphi)$
- se $\mathbf{X}\varphi_1 \in CL(\varphi)$ allora $\varphi_1 \in CL(\varphi)$
- se $\neg\mathbf{X}\varphi_1 \in CL(\varphi)$ then $\mathbf{X}\neg\varphi_1 \in CL(\varphi)$
- se $\varphi_1\ \mathbf{U}\ \varphi_2 \in CL(\varphi)$ allora $\varphi_1, \varphi_2, \mathbf{X}(\varphi_1\ \mathbf{U}\ \varphi_2) \in CL(\varphi)$

La dimensione di $CL(\varphi)$ è lineare in φ

Definizione dei vertici del tableau (atomi) Sia $M = \langle S, R, L\rangle$, con $L : S \to 2^{AP}$ (AP=formule atomiche). Un atomo è una tupla $\langle s_A, K_A\rangle$ tale che $s_A \in S$ e $K_A \subseteq CL(\varphi) \cup AP$ è un insieme consistente (anche rispetto ad $L(s_A)$) massimale di formule, cioè:

- per ogni $p \in AP$, $p \in K_A$ sse $p \in L(s_A)$
- per ogni $\varphi_1 \in CL(\varphi)$, $\varphi_1 \in K_A$ sse $\neg\varphi_1 \notin K_A$
- se $\varphi_1 \vee \varphi_2 \in CL(\varphi)$ allora $\varphi_1 \vee \varphi_2 \in K_A$ sse $\varphi_1 \in K_A$ oppure $\varphi_2 \in K_A$
- se $\neg\mathbf{X}\varphi_1 \in CL(\varphi)$ allora $\neg\mathbf{X}\varphi_1 \in K_A$ sse $\mathbf{X}\neg\varphi_1 \in K_A$
- se $\varphi_1\ \mathbf{U}\ \varphi_2 \in CL(\varphi)$ allora $\varphi_1\ \mathbf{U}\ \varphi_2 \in K_A$ sse:
  $\varphi_2 \in K_A$, oppure ($\varphi_1 \in K_A$ e $\mathbf{X}(\varphi_1\ \mathbf{U}\ \varphi_2) \in K_A$)

Costruzione del Tableau Un tableau è una sorta di modello *sintattico* per una formula φ combinato con tutti i possibili cammini di una struttura $M = \langle S, R, L\rangle$. Formalmente, un tableau è un grafo in cui gli atomi $\langle s_A, K_A\rangle$ sono i vertici ed esiste un arco da $\langle s_A, K_A\rangle$ a $\langle s_B, K_B\rangle$ sse

- $s_B = R(s_A)$
- per ogni formula $\mathbf{X}\varphi_1 \in CL(\varphi)$, $\mathbf{X}\varphi_1 \in K_A$ sse $\varphi_1 \in K_B$.

Proprietà del Tableau Sia G il tableau associato a M e φ una formula LTL, $M, s \models \varphi$ sse esiste un atomo $A = \langle s, K\rangle$ in G tale che $\varphi \in K$ ed esiste un cammino in G da A ad una componente fortementa connessa non elementare *self-fulfilling* C di G.

Una componente fortemente connessa C è self-fulfilling sse per ogni atomo B in C e per ogni formula φ_1 **U** $\varphi_2 \in B$ esiste un atomo B' in C tale che $\varphi_2 \in K_{B'}$. Una componente connessa è non elementare se: ha più di un nodo o ha un solo nodo con un self-loop.

10.7 Algoritmo di LTL Model Checking

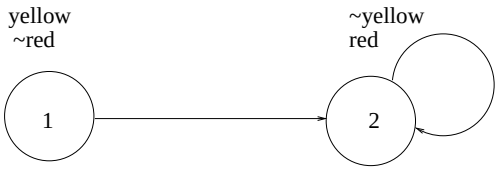
Si basa sulla proprietà appena vista. Prima costruiamo il tableau la cui dimensione è in $O((|S| + |R|) \cdot 2^{|\varphi|})$ cioè potenzialmente esponenziale nella formula φ. Poi cerchiamo le componenti fortemente connesse self-fulfilling nel tableau. Per calcolare le componenti fortemente connesse si può utilizzare l'algoritmo di Tarjan basato su visite DFS (complessità lineare nella dimensione del grafo). Infine, cerchiamo un cammino dallo stato inziale s_0 ad una delle componenti connesse non elementari self-fulfilling ad esempio con una visita DFS (complessità lineare nella dimensione del grafo). La complessità è quindi $O((|S| + |R|) \cdot 2^{|\varphi|})$

10.8 Esempio di costruzione di un Tableau

Consideriamo la seguente formula:

$$\varphi \equiv yellow\ \mathbf{U}\ red$$

ed il seguente modello LTL



Applichiamo l'algoritmo di LTL model checking basato sui tableaux per calcolare gli stati di M sui quali vale φ.

Chiusura di φ La chiusura $Cl(\varphi)$ di φ contiene le seguenti formule:

$$\begin{array}{cc} yellow & red \quad \neg yellow \quad \neg red \\ yellow\ \mathbf{U}\ red & \neg(yellow\ \mathbf{U}\ red) \\ \mathbf{X}\ (yellow\ \mathbf{U}\ red) & \neg\mathbf{X}\ (yellow\ \mathbf{U}\ red) \\ \mathbf{X}\ \neg(yellow\ \mathbf{U}\ red) & \neg\mathbf{X}\ \neg(yellow\ \mathbf{U}\ red) \end{array}$$

A questo punto dobbiamo costruire il *tableau* ottenuto considerando insiemi massimali di formule in $Cl(\varphi)$ consistenti con la etichettatura del modello M.

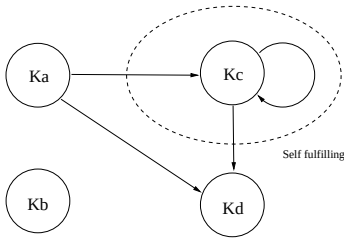
Tableau Chiamiamo f la formula ($yellow$ **U** red). Gli insiemi di formule massimali consistenti con le etichette dello stato 1 sono:

$$K_a = \{yellow,\ \neg red,\ f,\ \mathbf{X}\ f,\ \neg\mathbf{X}\ \neg f\}$$
$$K_b = \{yellow,\ \neg red,\ \neg f,\ \mathbf{X}\ \neg f,\ \neg\mathbf{X}\ f\}$$

Gli insiemi di formule massimali consistenti con le etichette dello stato 2 sono:

$$K_c = \{\neg yellow,\ red,\ f,\ \mathbf{X}\ f,\ \neg\mathbf{X}\ \neg f\}$$
$$K_d = \{\neg yellow,\ red,\ f,\ \neg\mathbf{X}\ f,\ \mathbf{X}\ \neg f\}$$

Il tableau e' costruito come segue:



Poiche dall'atomo $\langle 1, K_a\rangle$ esiste un cammino ad una componente self-fulfilling e $f \in K_a$, segue che f è soddisfatta in 1. La stessa cosa vale per 2. Quindi $M, 1 \models f$ e $M, 2 \models f$.

10.9 Modiche alle Regole del Tableau per gestire F

Abbiamo visto che $\mathbf{F}\ \varphi \equiv true\ \mathbf{U}\ \varphi$. Sulla base di questa idea notiamo che per gestire direttamente formule del tipo $\mathbf{F}\ \varphi$ nel tableau dobbiamo aggioungere le seguenti regole.

Chiusura Se $\mathbf{F}\ \varphi \in Cl(\psi)$ sse $\varphi \in Cl(\psi)$ e $\mathbf{X}\ \ \mathbf{F}\ \varphi \in Cl(\psi)$

Insiemi Massimali Consistenti Dato un atomo A, se $\mathbf{F}\ \varphi \in Cl(\psi)$ allora, $\mathbf{F}\ \varphi \in K_A$ sse ($\varphi \in K_A$ oppure $\mathbf{X}\ \ \mathbf{F}\ \varphi \in K_A$)

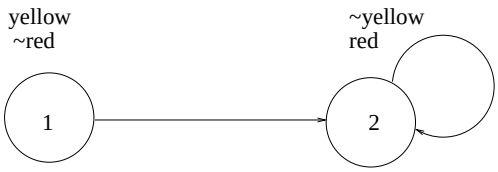
Cammini di Accettazione Dobbiamo aggiungere alla nozione di SCC self-fulfilling la condizione sulla soddisfacibilità di formule del tipo $\mathbf{F}\ \varphi$: cioè per ogni formula $\mathbf{F}\ \varphi \in K_A$ dove A è un'atomo della componente fortemente connessa deve esistere un atomo B nella stessa componente tale che $\varphi \in B$.

10.10 Esempio

Consideriamo la seguente formula:

$$\varphi = \mathbf{F}\ \ yellow$$

ed il precedente modello LTL



Applichiamo l'algoritmo di LTL model checking basato sui tableaux per calcolare gli stati di M sui quali vale φ.

Chiusura di φ La chiusura $Cl(\varphi)$ di φ contiene le seguenti formule:

$$\begin{array}{cc} yellow & \neg yellow \\ \mathbf{F}\ yellow & \neg\ \mathbf{F}\ yellow \\ \mathbf{X}\ \ \mathbf{F}\ yellow & \neg\mathbf{X}\ \ \mathbf{F}\ yellow \\ \mathbf{X}\ \neg\ \mathbf{F}\ yellow & \neg\mathbf{X}\ \neg\ \mathbf{F}\ yellow \end{array}$$

A questo punto dobbiamo costruire il *tableau* ottenuto considerando insiemi massimali di formule in $Cl(\varphi)$ consistenti con la etichettatura del modello M.

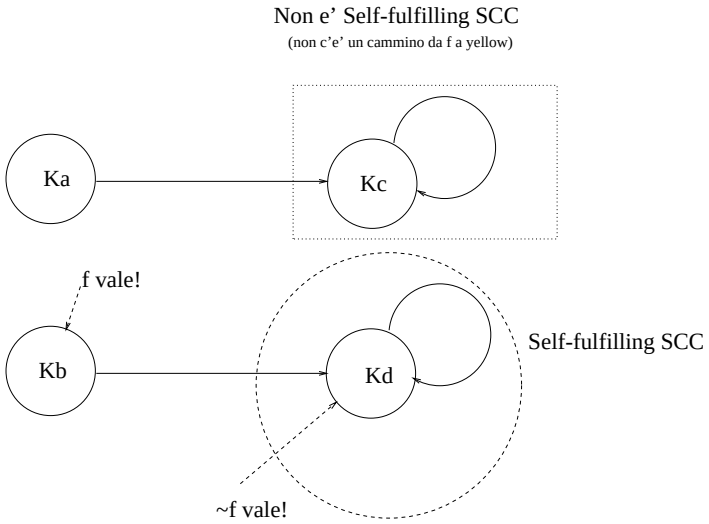
Tableau Chiamiamo f la formula $\mathbf{F}\ yellow$. Gli insiemi di formule massimali consistenti con le etichette dello stato 1 sono:

$$\begin{array}{rcl} K_a & = & \{yellow,\ \neg red,\ f,\ \mathbf{X}\ f,\ \neg\mathbf{X}\ \neg f\} \\ K_b & = & \{yellow,\ \neg red,\ f,\ \mathbf{X}\ \neg f,\ \neg\mathbf{X}\ f\} \end{array}$$

Gli insiemi di formule massimali consistenti con le etichette dello stato 2 sono:

$$\begin{array}{rcl} K_c & = & \{\neg yellow,\ red,\ f,\ \mathbf{X}\ f,\ \neg\mathbf{X}\ \neg f\} \\ K_d & = & \{\neg yellow,\ red,\ \neg f,\ \neg\mathbf{X}\ f,\ \mathbf{X}\ \neg f\} \end{array}$$

Il tableau e' costruito come segue:



Poiche dall'atomo $\langle 1, K_b\rangle$ esiste un cammino ad una componente self-fulfilling e $f \in K_b$, segue che f è soddisfatta in 1; per la stessa ragione $\neg f$ è soddisfatta in 2. Quindi $M, 1 \models f$ e $M, 2 \models \neg f$. Notiamo che $f \in K_a$ ma non esiste un cammino ad una SCC self-fulling ($\langle 2, K_c\rangle$ non è selfulfilling perchè $\mathbf{F}\ yellow \in K_c$ ma $yellow \notin K_c$.

11 Linear Time vs Branching Time

In LTL la nozione qualitativa del tempo è *lineare*: in ogni istante esiste un unico possibile successore (attraverso la funzione di accessibilità R di un modello LTL) e quindi un solo possibile futuro. Intuitivamente, una sequenza di stati $s\ R(s)\ R(R(s))\ldots$ rappresenta una computazione; gli operatori temporali descrivono l'ordine degli eventi (descritti tramite formule atomiche) lungo

un singolo cammino. Esiste un'altra scuola di pensiero in cui la nozione di tempo non è lineare bensì ha una struttura ad albero, i.e., $R(s)$ è un insieme di possibili stati successori, da cui la Computation Tree Logic (CTL) LTL e CTL non sono comparabili, alcune proprietà che si possono esprimere in LTL non si possono esprimere in CTL e vice versa Le technique di deduzione automatica utilizzate per le due logiche hanno natura diversa (vedremo in seguito la tecnica chiamata symbolic model checking per CTL).

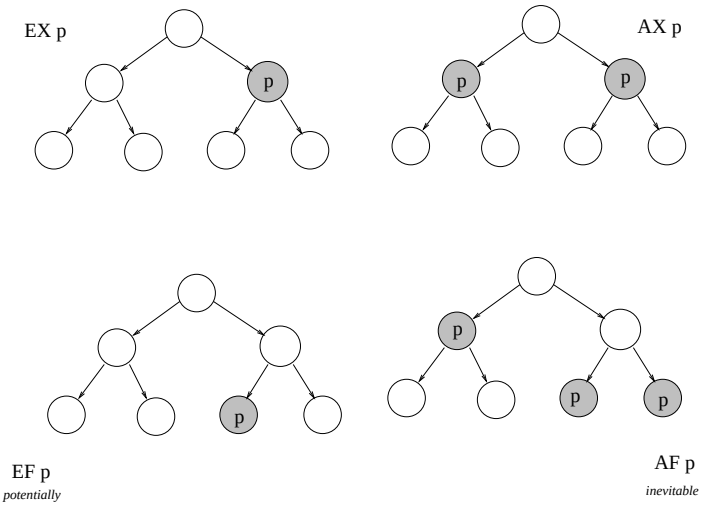
12 Computation Tree Logic: CTL

Le formule CTL sono definite come segue

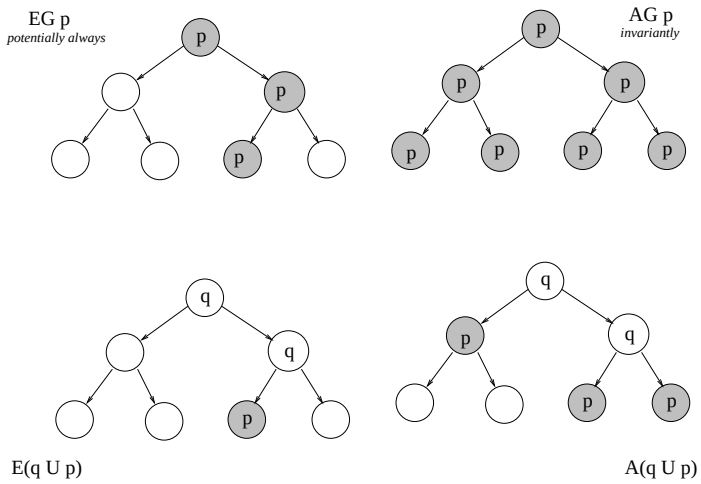
- Proposizioni Atomiche: simboli di predicato $p, q, r, \ldots$
- Connettivi classici: $\neg\psi \quad \varphi \wedge \psi \quad \varphi \vee \psi \quad \varphi \supset \psi$
- Due tipi di connettivi (da usarsi in combinazione)
  - Connettivi sui cammini: **E**, **A**
  - Connettivi temporali: **X**, **F**, **G**, **U**
- Le formule temporali CTL hanno forma $Op_{cammino}\ Op_{tempo}\varphi$

Una formula senza connettivo temporale al top-level si riferisce all'istante corrente **A** (*per tutti (all) i cammini*) e **E** (*per qualche (exists) cammino*) vanno usati in combinazione con **X** (next) e **U** (until).

EXφ = per qualche cammino nello stato successivo (exists next) φ
EFφ = per qualche cammino primo o poi (exists in the future) φ
AXφ = per tutti i cammini nello stato successivo (always next) φ
AFφ = per tutti i cammini primo o poi (always in the future) φ



EGφ = per qualche cammino sempre (exists globally) φ
AGφ = per tutti i cammini sempre (always globally) φ
E(φ**U**ψ) = per qualche cammino φ fino a che (exists until) ψ
A(φ**U**ψ) = per tutti i cammini φ fino a che (always until) ψ



L'insieme di connettivi $\neg$, $\vee$, **EX**, **E**, **A** and **U** insieme alle formule atomiche è sufficiente per definire le formule CTL. Infatti

$$\begin{array}{rcl}
\mathbf{EF}\varphi & \equiv & \mathbf{E}(\mathit{true}\ \mathbf{U}\ \varphi) \text{ potentially} \\
\mathbf{AF}\varphi & = & \mathbf{A}(\mathit{true}\ \mathbf{U}\ \varphi) \text{ inevitable} \\
\mathbf{EG}\varphi & \equiv & \neg\mathbf{AF}\neg\varphi \text{ potentially always} \\
\mathbf{AG}\varphi & \equiv & \neg\mathbf{EF}\neg\varphi \text{ invariantly} \\
\mathbf{AX}\varphi & = & \neg\mathbf{EX}\neg\varphi \text{ for all paths next}
\end{array}$$

Durante l'esecuzione di un sistema concorrente si può giungere ad uno stato in cui vale *Start* ma non *Ready*

$$\mathbf{EF}(\mathit{Start} \wedge \neg \mathit{Ready})$$

In ogni esecuzione ogni richiesta viene infine soddisfatta

$$\mathbf{AG}(\mathit{Req}\ \supset\ \mathbf{AF}\ \mathit{Ack})$$

Un certo processo viene abilitato infinitamente spesso in ogni esecuzione

$$\mathbf{AG}(\mathbf{AF}\ \mathit{Enabled})$$

Da ogni stato è possibile tornare allo stato *Restart*

$$\mathbf{AG}(\mathbf{EF}\ \mathit{Restart})$$

Qualsiasi cosa succeda, il sistema alla fine sarà in *deadlock* permanente

$$\mathbf{AG}(\mathbf{EF}\ \mathit{Restart})$$

Qualsiasi cosa succeda, il sistema alla fine sarà in *deadlock* permanente

$$\mathbf{AF}(\mathbf{AG}\ \mathit{Deadlock})$$

12.1 Semantica Model Theoretic

Un modello CTL è una tripla $M = \langle S, R, L\rangle$ (modal frame alla Kripke) dove

- S è un insieme numerabile non vuoto di stati

- $R \subseteq S \to S$ è una relazione *totale* che mette in relazione uno stato s con l'insieme dei suoi successori
- $L : S \to 2^{AP}$ assegna ad ogni stato $s \in S$ le formule atomiche proposizionali che sono vere in s

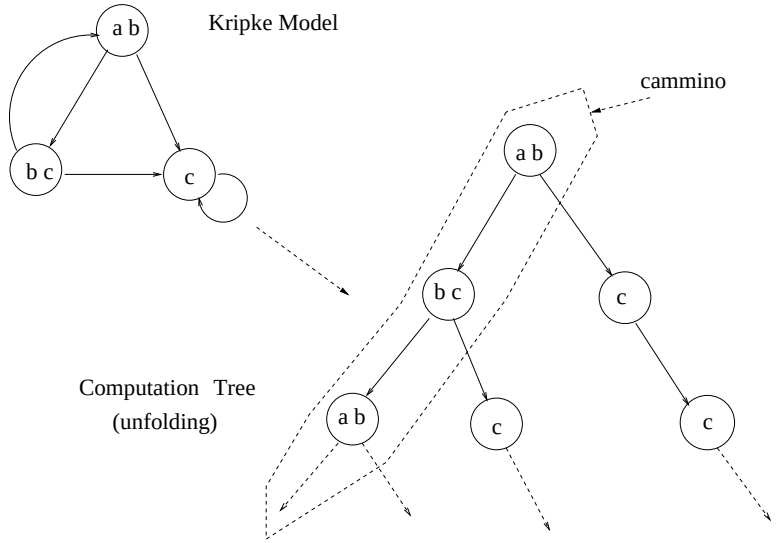
R è totale cioè per ogni $s \in S$ esiste almeno un s' t.c. $\langle s, s' \rangle \in R$. R permette di costruire branching future. Un cammino σ è una sequenza infinita di stati

$$s_0 s_1 \dots s_i \dots$$

tale che $\langle s_k, s_{k+1} \rangle \in R$ per ogni $k \geq 0$. $\sigma[i]$ identifica l'i-esimo stato della sequenza σ L'insieme dei cammini che partono da s in M è definito come

$$P_M(s) = \{\sigma \mid \sigma \text{ è un cammino e } \sigma[0] = s\}$$

per ogni modello CTL M esiste un *albero infinito di computazioni* (Computation Tree) dove ogni nodo è etichettato con uno stato $s \in S$ e tale che $\langle s', s'' \rangle$ è un arco dell'albero sse $\langle s', s'' \rangle \in R$.



Soddisfacibilità in CTL Fissato $M = \langle S, R, L \rangle$, la relazione $M, s \models \varphi$ (M soddisfa φ in s) è definita come

- $s \models p$ sse $p \in L(s)$
- $s \models \neg\phi$ sse $s \not\models \phi$
- $s \models \varphi \vee \psi$ sse $s \models \varphi$ or $s \models \psi$
- $s \models \mathbf{EX}\varphi$ sse $\exists \sigma \in P_M(s)$ t.c. $\sigma[1] \models \varphi$
- $s \models \mathbf{E}(\varphi\ \mathbf{U}\ \psi)$ sse $\exists \sigma \in P_M(s)$ t.c. $\exists j \geq 0.\ \sigma[j] \models \psi \wedge (\forall 0 \leq k < j.\ \sigma[k] \models \varphi)$
- $s \models \mathbf{A}(\varphi\ \mathbf{U}\ \psi)$ sse $\forall \sigma \in P_M(s)$ t.c. $\exists j \geq 0.\ \sigma[j] \models \psi \wedge (\forall 0 \leq k < j.\ \sigma[k] \models \varphi)$

dove $P_M(s)$ è l'insieme dei cammini infiniti di M che parte dallo stato s.

Dalle definizioni precedenti ricaviamo la semantica dei connettivi derivati **EF**, **EG**, ecc.

- $s \models \mathbf{EF}\varphi$ se $\exists \sigma \in P_M \quad (\exists j \geq 0.\sigma[j] \models \varphi)$
- $s \models \mathbf{EG}\varphi$ se $\exists \sigma \in P_M \quad (\forall j \geq 0.\sigma[j] \models \varphi)$
- $s \models \mathbf{AF}\varphi$ se $\forall \sigma \in P_M \quad (\exists j \geq 0.\sigma[j] \models \varphi)$
- $s \models \mathbf{AG}\varphi$ se $\forall \sigma \in P_M \quad (\forall j \geq 0.\sigma[j] \models \varphi)$

LTL vs CTL Consideriamo due linguaggi logici L e L' con la stessa nozione di modello e stato. Allora L e L' sono ugualmente espressive se per tutti i modelli M e stati s valgono le due seguenti condizioni

$$\forall \varphi \in L.\ \exists \psi \in L'.\ (M, s \models \varphi \ sse\ M, s \models \psi)$$
$$\forall \varphi \in L'.\ \exists \psi \in L.\ (M, s \models \varphi \ sse\ M, s \models \psi)$$

Se vale solo la prima parte della congiunzione ma non la seconda allora L è strettamente meno espressiva di L'.

LTL e CTL non sono comparabili rispetto alla precedente relazione. Per interpretare una formula LTL su una struttura CTL si suppone che la formula sia implicitamente quantificata universalmente su tutti i cammini, cioè la formula LTL φ viene interpretata in CTL come $\mathbf{A}(\varphi)$.

Utilizzando questa comune nozione di modello, si può dimostrare che non esistono formule CTL equivalenti alla formula LTL

$$\mathbf{A}(\mathbf{FG}p)$$

Vice versa, non esistono formule LTL equivalenti alla formula CTL

$$\mathbf{AG}\ \mathbf{EF}p$$

LTL $\not\subseteq$ CTL: Intuizione Fig. 8 illustra l'intuizione sul tipo di modelli che si possono utilizzare per mostrare che LTL non è meno espressiva di CTL. Il modello in questione ha un cammino infinito lungo il quale p è sempre vero. Poichè le formule LTL vengono analizzate cammino per cammino la formula $\mathbf{A}(\mathbf{FG}p)$ è vera in tale modello. Una possibile formulazione di tale proprietà in CTL potrebbe essere $\mathbf{AF}(\mathbf{AG}p)$. Tuttavia tale formula non è vera nel modello visto sopra. Infatti la formula tiene in considerazione le scelte in ogni nodo del modello, e nel primo nodo del grafo una delle due strade possibili porta sempre ad una violazione della proprietà. La dimostrazione che LTL $\not\subseteq$ CTL è chiaramente più complessa. Bisogna infatti dimostrare che non esiste una formula CTL equivalente alla formula LTL in questione.

CTL $\not\subseteq$ LTL: Intuizione Fig. 9 illustra l'intuizione sul tipo di modelli che si possono utilizzare per mostrare che CTL non è meno espressiva di LTL. Il modello in questione ha un cammino infinito lungo il quale p è sempre falso. Tuttavia in ogni nodo di tale cammino esiste una strada che nel quale p diventa stabilmente vera. Quindi la formula CTL $\mathbf{AG}(\mathbf{EF}p)$ è vera in tale modello. Una possibile formulazione di tale proprietà in LTL potrebbe essere $\mathbf{A}(\mathbf{GF}p)$. Poichè le formule LTL vengono analizzate cammino per cammino tale formula è falsa nello stesso modello. Il problema è duale al caso della Fig. 8: la formula LTL non tiene conto delle scelte non deterministiche in un modello branching. Di nuovo, la dimostrazione che CTL $\not\subseteq$ CTL è chiaramente più complessa. Bisogna infatti dimostrare che non esiste una formula CTL equivalente alla formula LTL in questione.

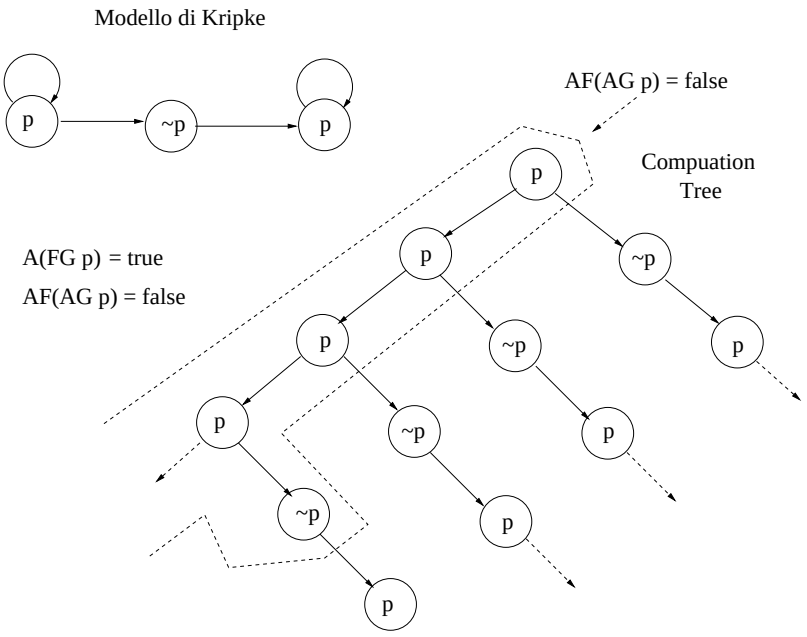


Figura 8: Intuizione su LTL $\not\subseteq$ CTL.

La Logica CTL* Estiste una logica temporale più espressiva sia di CTL che di LTL chiamata CTL*. In CTL* si possono annidare connettivi booleane e temporali prima di applicare i quantificatori sui cammini **A** e **E**. Ad esempio in CTL* si possono scrivere proprietà quali $\mathbf{E}(\mathbf{GF}p)$ (esiste un cammino lungo il quale p vale infinitamente spesso) che non sono esprimibili nè in LTL nè in CTL

13 Model Checking CTL

Il problema di *model checking* per CTL è definito come segue:

Dato un modello di Kripke finito M, uno stato s ed una formula CTL φ, vogliamo decidere se $M, s \models \varphi$. Il problema è decidibile ed esistono vari metodi per risolverlo. Vedremo due possibili approcci:

- Algoritmo esplicito basato sull'enumerazione degli stati del modello
- Algoritmo simbolico basato su una rappresentazione simbolica dello spazio degli stati

13.1 CTL e Computazioni di Punto Fisso

Per poter capire gli algoritmi di model checking per CTL occorre innanzitutto vedere la relazione tra formule CTL ed espressioni di punto fisso definite su operatori di transformazione di insiemi di stati L'idea è quella di associare ad una formula CTL la sua denotazione cioè l'insieme di stati che la soddisfano e poi vedere gli operatori temporali come transformazioni definite su insiemi di stati

Sia $\sqsubseteq \subseteq D \times D$ una relazione d'ordine parziale, cioè *riflessiva*, *anti-simmetrica*, e *transitiva*. Dato $A \subseteq D$: $a \in D$ è un upper bound per A se per ogni $b \in A$ vale $b \sqsubseteq a$; a è il *least upper bound* per

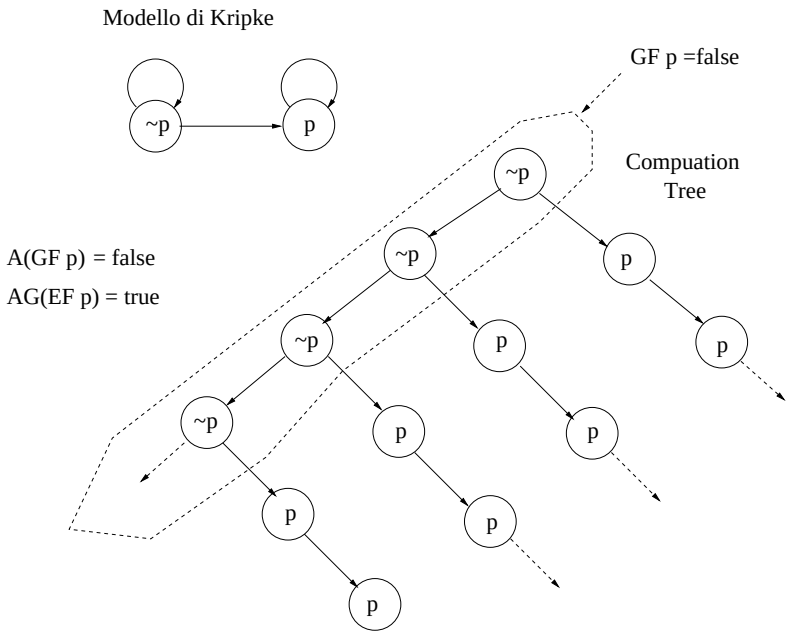


Figura 9: Intuizione su CTL $\not\subseteq$ LTL.

A se a è un upper bound, e $a \sqsubseteq b$ per ogni altro upper bound b di A. Dato $A \subseteq D$: $a \in D$ è un lower bound per A se per ogni $b \in A$ vale $a \sqsubseteq b$; a è il *greatest upper bound* per A se a è un lower bound, e $b \sqsubseteq a$ per ogni altro lower bound b di A.

Reticoli completi Un *reticolo* $\langle D, \sqsubseteq \rangle$ è un ordine parziale in cui per ogni sottoinsieme $A \subseteq D$ esistono il *least upper bound* e il *greatest lower bound* per A. In particolare un reticolo ha quindi un elemento minimo (*bottom*) ed un elemento massimo (*top*).

Example 7 *l'insieme delle parti su un insieme D ordinato rispetto all'inclusione di insiemi è un lattice completo: bottom=$\emptyset$, top=D, greatest lower bound=$\cap$, least upper bound=$\cup$.*

Funzioni monotone e punti fissi Un funzione F definita su un reticolo $\langle D, \sqsubseteq \rangle$ è *monotona* sse per ogni $a, b \in D$ se $a \sqsubseteq b$ allora $F(a) \sqsubseteq F(b)$. Un *punto fisso* per F è un elemento $a \in D$ tale che $F(a) = a$.

Teorema di Knaster-Tarski Ogni funzione monotona $F : D \to D$ su un reticolo completo $\langle D, \bot, \top, \sqsubseteq \rangle$ ammette un lattice completo di punti fissi, e quindi in particolare ha un minimo e massimo punto fisso (rispetto all'ordinamento $\sqsubseteq$).

Consideriamo ora un dominio D *finito*. Il minimo punto fisso (least fixpoint, lfp) può essere calcolato come *least upper bound* ($\sqcup$) della sequenza (totalmente ordinata)

$$\bot \ \sqsubseteq \ F(\bot) \ \sqsubseteq \ F(F(\bot)) \ \ldots$$

Cioè

$$lfp(F) = \sqcup_{k \geq 0} F^k(\bot)$$

Il massimo punto fisso (greatest fixpoint, gfp) può essere calcolato come *greatest lower bound* ($\sqcap$) della sequenza decrescente

$$\top \sqsupseteq F(\top) \sqsupseteq F(F(\top)) \ \ldots$$

Cioè

$$gfp(F) = \sqcap_{k \geq 0} F^k(\top)$$

Per domini non finiti queste caratterizzazioni potrebbero non valere (in particolar modo per il gfp).

Lattice delle formule CTL Dato un modello $M = \langle S, R, L \rangle$ e una formula φ definiamo

$$[\![\varphi]\!] = \{s \mid M, s \models \varphi\}$$

Ordiniamo le formule rispetto alle denotazioni come segue:

$$\varphi \sqsubseteq \psi \ \text{ sse } \ [\![\varphi]\!] \subseteq [\![\psi]\!]$$

L'insieme delle formule CTL ordinato con $\sqsubseteq$ è un *reticolo completo*

- Bottom $[\![false]\!] = \emptyset$
- Top $[\![true]\!] = S$
- Upper bound $= \vee$, infatti, $[\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$
- Lower bound $= \wedge$, infatti, $[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$

Formule come trasformazioni di insiemi di stati Per capire come si possono interpretare i connettivi CTL come trasformazioni sulle denotazioni delle formule è importante notare che valgono i seguenti *assiomi di espansione* (come in LTL)

- $\mathbf{EG}\varphi \equiv \varphi \wedge \mathbf{EX}(\mathbf{EG}\varphi)$
- $\mathbf{AG}\varphi \equiv \varphi \wedge \mathbf{AX}(\mathbf{AG}\varphi)$
- $\mathbf{EF}\varphi \equiv \varphi \vee \mathbf{EX}(\mathbf{EF}\varphi)$
- $\mathbf{AF}\varphi \equiv \varphi \vee \mathbf{AX}(\mathbf{AF}\varphi)$
- $\mathbf{E}(\varphi \ \mathbf{U} \ \psi) \equiv \psi \vee (\varphi \wedge (\mathbf{EX}(\mathbf{E}[\varphi \ \mathbf{U} \ \psi]))$
- $\mathbf{A}(\varphi \ \mathbf{U} \ \psi) \equiv \psi \vee (\varphi \wedge (\mathbf{AX}(\mathbf{A}[\varphi \ \mathbf{U} \ \psi]))$

L'assioma di espansione

$$\mathbf{E}(\varphi \ \mathbf{U} \ \psi) \equiv \psi \vee (\varphi \wedge (\mathbf{EX}(\mathbf{E}[\varphi \ \mathbf{U} \ \psi]))$$

suggerisce di considerare l'espressione $\mathbf{E}(\varphi \ \mathbf{U} \ \psi)$ come punto fisso della funzione G da formule CTL in formule CTL definita come segue

$$G(Z) = \psi \vee (\varphi \wedge \mathbf{EX}Z)$$

oppure della funzione F da insiemi di stati in insiemi di stati definita come segue

$$g(Z) = [\![\psi]\!] \cup ([\![\varphi]\!] \cap \{s \in S \mid \exists s' \in R(s) \cap Z\})$$

dove $R(s) = \{s' \in S \mid \langle s, s' \rangle \in R\}$.
Nota: Z è un punto fisso di G se $G(Z) = Z$.

Formule come punti fissi In effetti le denotazioni delle formule CTL possono essere viste come segue

- $[\![\mathbf{EG}\varphi]\!]$ è il *massimo* punto fisso (greatest fixpoint) della funzione $F(Z) = [\![\varphi]\!] \cap \{s \in S \mid \exists s' \in R(s) \cap Z\}$
- $[\![\mathbf{AG}\varphi]\!]$ è il *massimo* punto fisso (greatest fixpoint) della funzione $F(Z) = [\![\varphi]\!] \cap \{s \in S \mid \forall s' \in R(s) \cap Z\}$
- $[\![\mathbf{EF}\varphi]\!]$ è il *minimo* punto fisso (least fixpoint) della funzione $F(Z) = [\![\varphi]\!] \cup \{s \in S \mid \exists s' \in R(s) \cap Z\}$
- $[\![\mathbf{AF}\varphi]\!]$ è il *minimo* punto fisso (greatest fixpoint) della funzione $F(Z) = [\![\varphi]\!] \cup \{s \in S \mid \forall s' \in R(s) \cap Z\}$

Algoritmo di CTL Model Checking Esplicito Dato un modello di Kripke finito M,uno stato s ed una formula CTL φ, vogliamo decidere se $M, s \models \varphi$. L'algoritmo (esplicito) che risolve il problema è dovuto a Emerson-Clarke ed è basato sulla seguente idea: si etichetta ogni stato s in M con l'insieme di *sottoformule* di φ che sono valide in s. L'etichettatura è costruita iterativamente a partire dalle sottoformule di dimensione minima (i simboli proposizionali) e poi procedendo per induzione strutturale

Sottoformule di una formula CTL Abbiamo visto che possiamo restringere la nostra attenzione a formule solo con **EX**, **A**, **E** e **U**. L'insieme $Sub(\varphi)$ delle sottoformule di φ è definito come segue

- $Sub(p) = \{p\}$, $p \in AP$ (formula atomica)
- $Sub(\neg\varphi) = Sub(\varphi) \cup \{\neg\varphi\}$
- $Sub(\varphi \vee \psi) = Sub(\varphi) \cup Sub(\psi) \cup \{\varphi \vee \psi\}$
- $Sub(\mathbf{EX}\varphi) = Sub(\varphi) \cup \{\mathbf{EX}\varphi\}$
- $Sub(\mathbf{E}(\varphi\ \mathbf{U}\ \psi)) = Sub(\varphi) \cup Sub(\psi) \cup \{\mathbf{E}(\varphi \mathbf{U} \psi)\}$
- $Sub(\mathbf{A}(\varphi\ \mathbf{U}\ \psi)) = Sub(\varphi) \cup Sub(\psi) \cup \{\mathbf{A}(\varphi \mathbf{U} \psi)\}$

Algoritmo Sia $M = \langle S, R, L\rangle$ ed AP l'insieme delle sue formule atomiche, l'algoritmo di model checking si basa sulla funzione Sat che data una formula CTL φ restituisce l'*insieme degli stati* dove la formula vale

```
function Sat(φ : CTL formula) : set of states
begin
  if φ = true then return S
  if φ = false then return ∅
  if φ ∈ AP then return { s | φ ∈ L(s)}
  if φ = ¬ψ then return S \ Sat(ψ)
  if φ = φ1 ∨ φ2 then return Sat(φ1) ∪ Sat(φ2)
  if φ = EXφ1 then return { s ∈ S | ⟨s, s'⟩ ∈ R ∧ s' ∈ Sat(φ1)}
  if φ = E(φ1 U φ2) then return Sat_EU(φ1, φ2)
  if φ = A(φ1 U φ2) then return Sat_AU(φ1, φ2)
end.
```

La funzione Sat_{EU} è definita come minimo punto fisso

```
function Sat_EU(φ1, φ2 : CTL formula) : set of states
var Q, Q' : set of states
begin
Q := Sat(φ2)
Q' := ∅
while Q ≠ Q' do
    Q' := Q;
    Q := Q ∪ {s | s ∈ Sat(φ1) and ∃s' ∈ R(s) ∩ Q };
endw;
returnQ;
end.
```

La funzione Sat_{AU} è definita come minimo punto fisso

```
function Sat_AU(φ1, φ2 : CTL formula) : set of states
var Q, Q' : set of states
begin
Q := Sat(φ2)
Q' := ∅
while Q ≠ Q' do
    Q' := Q;
    Q := Q ∪ {s | s ∈ Sat(φ1) and ∀s' ∈ R(s).s' ∈ Q };
endw;
returnQ;
end.
```

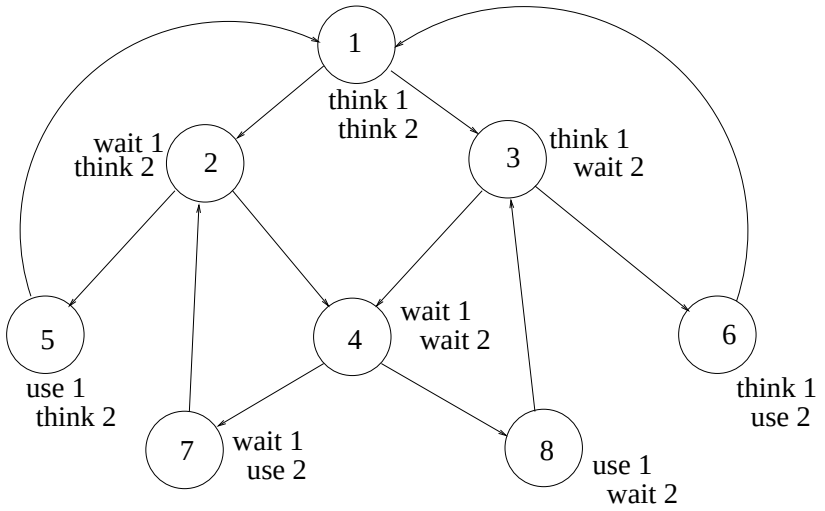
13.2 Esempio

Consideriamo il seguente algoritmo di mutua esclusione.

repeat
- *think* : $turn_1 := turn_2 + 1$;
- *wait* : **when** $turn_1 < turn_2$ **or** $turn_2 = 0$ **do**
- *use* : **critical section**; $turn_1 := 0$

forever

In modo astratto possiamo modellare l'insieme degli stati di dei due processi tramite i flag booleani $think_i$=il processo i è nella fase *think*, $wait_i$=il processo i è nella fase *wait*, use_i=il processo i è nella fase *use* per $i : 1, 2$. Allora otteniamo il seguente modello di Kripke che modella le esecuzioni 'astratte' del protocollo.



Tra le proprietà CTL interessanti che vogliamo studiare troviamo

Mutua esclusione

$$\mathbf{AG}[\neg(use_1 \wedge use_2)]$$

Freedom from Starvation

$$\mathbf{AG}[wait_i \supset \mathbf{AF}use_i]$$

per $i : 1, 2$. Proviamo a studiare tali proprietà utilizzando l'algoritmo di model checking per CTL. Innanzitutto notiamo che $\mathbf{AG}[\neg(use_1 \wedge use_2)]$ é equivalente alla formula

$$\neg\mathbf{EF}(use_1 \wedge use_2).$$

Per calcolare gli stati che soddisfano tale formula (cioé le sue denotazioni) occorre considerare le denotazioni delle sottoformule use_1, use_2 e poi della formula stessa. Allora abbiamo che:

$$[\![use_1]\!] = \{5, 8\}$$

Inoltre

$$[\![use_2]\!] = \{6, 7\}$$

E quindi

$$[\![use_1 \wedge use_2]\!] = \emptyset$$

A questo punto dobbiamo calcolare le denotazioni della formula principale, cioé il minimo punto fisso della trasformazione da insiemi di stati in insiemi di stati $g(Z) = [\![use_1 \wedge use_2]\!] \cup Pre_{\mathbf{EX}}(Z)$, dove $Pre_{\mathbf{EX}}(S)$ denota tutti gli stati del modello M che in un passo raggiungono uno stato in S, cioé gli stati **predecessori** degli stati in S Il minimo punto fisso corrisponde al limite della catena (calcolato tramite l'unione) $\emptyset, g(\emptyset), g^2(\emptyset), \ldots$ Quindi

$$\begin{aligned} &S_0 = \emptyset \\ &S_1 = \emptyset \cup Pre_{\mathbf{EX}}(\emptyset) = \emptyset \\ &\ldots \\ &S_i = \emptyset \end{aligned}$$

Notate infatti che l'insieme degli stati predecessori dell'insieme vuoto é ancora vuoto. E quindi il minimo punto fisso é vuoto cioé $[\![\mathbf{EF}(use_1 \wedge use_2)]\!] = \emptyset$. Infine $[\![\neg\mathbf{EF}(use_1 \wedge use_2)]\!] = \{1, 2, 3, 4, 5, 6, 7, 8\}$ cioé tutti gli stati soddisfano la formula in questione e quindi nel nostro modello vale la proprietá di mutua esclusione.

Freedom from Starvation

$$\mathbf{AG}[wait_1 \supset \mathbf{AF}use_1]$$

Notiamo che la formula precedente é equivalente alla formula

$$\neg\mathbf{EF}\neg(\neg wait_1 \vee \neg(\mathbf{EG}\neg use_1))$$

cioè

$$\neg\mathbf{EF}(wait_1 \wedge (\mathbf{EG}\neg use_1))$$

Per calcolare gli stati che soddisfano tale formula (cioé le sue denotazioni) occorre considerare le denotazioni delle sottoformule $wait_1$, use_1, $\neg use_1$, $\mathbf{EG}(\neg use_1)$, $wait_1 \wedge (\mathbf{EG}\neg use_1)$ e poi della formula stessa. Calcoliamo le denotazioni delle varie formule.

Sottoformula $\mathbf{EG}\neg use_1$ Allora abbiamo che:

$$[\![\neg use_1]\!] = \{1, 2, 3, 4, 6, 7\}$$

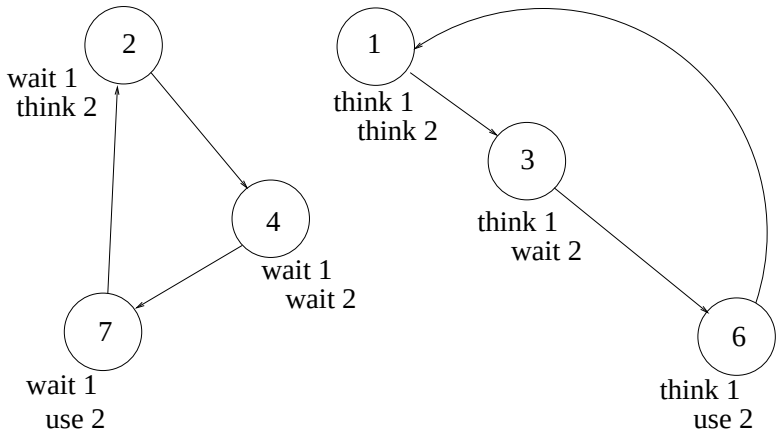
A questo punto dobbiamo calcolare le denotazioni della formula **EG**, cioé il massimo punto fisso della trasformazione da insiemi di stati in insiemi di stati $f(Z) = [\![\neg use_1]\!] \cap Pre_{\mathbf{EX}}(Z)$, dove $Pre_{\mathbf{EX}}(Z)$ denota gli stati **predecessori** degli stati in Z. Formalmente

$$Pre_{\mathbf{EX}}(Z) = \{s \mid \exists s' \in S \ s.t. \ \langle s, s' \rangle \in R, \ s' \in Z\}$$

Sia $S = \{1, 2, 3, 4, 5, 6, 7, 8\}$ (l'insieme di tutti gli stati). Il massimo punto fisso corrisponde al limite (calcolato come intersezione (greatest lower bound (gfp)) della catena $S, g(S), g^2(S), \ldots$. Quindi

$$\begin{aligned} &S_0 = S = \{1, 2, 3, 4, 5, 6, 7, 8\} \\ &S_1 = \{1, 2, 3, 4, 6, 7\} \cap Pre_{\mathbf{EX}}(S) = \{1, 2, 3, 4, 6, 7\} \cap S = \{1, 2, 3, 4, 6, 7\} \\ &S_2 = S_1 = S_i = gfp(f) \end{aligned}$$

Quindi il massimo punto fisso corrisponde all'insieme di stati $\{1, 2, 3, 4, 6, 7\}$. Notate che $\mathbf{EG}\neg use_1$ viene soddisfatta se esistono cammini infiniti lungo i quali il primo processo non entra mai nella regione critica. Il massimo punto fisso rappresenta l'insieme degli stati lungo tali cicli: I cicli semplici sono



Passiamo ora alle altre sottoformule.

Sottoformula $wait_1 \wedge (\mathbf{EG}\neg use_1)$ Allora abbiamo che

$$[\![wait_1]\!] = \{2, 4, 7\}$$

e quindi

$$T = [\![wait_1 \wedge (\mathbf{EG}\neg use_1)]\!] = \{2, 4, 7\}$$

Sottoformula $\mathbf{EF}(wait_1 \wedge (\mathbf{EG}\neg use_1))$ Ci rimane ora da calcolare il minimo punto fisso (lfp) associato alla trasformazione $h(Z) = T \cup Pre_{\mathbf{EX}}(Z)$. Allora abbiamo che:

$$\begin{array}{l} S_0 = \emptyset \\ S_1 = \{2, 4, 7\} \cup Pre_{\mathbf{EX}}(\emptyset) = \{2, 4, 7\} \\ S_2 = \{2, 4, 7\} \cup Pre_{\mathbf{EX}}(\{2, 4, 7\}) = \{1, 2, 3, 4, 7\} \\ S_3 = \{2, 4, 7\} \cup Pre_{\mathbf{EX}}(\{1, 2, 3, 4, 7\}) = \{1, 2, 3, 4, 5, 6, 7, 8\} = S \\ S_i = S = lfp(h) \end{array}$$

Quindi questa sottoformula é vera in ogni stato, infatti da ogni stato possiamo raggiungere uno stato in cui il primo processo aspetta di entrare nella regione critica e a partire dal quale non entrera' mai nella regione critica.

Formula $\mathbf{AG}[wait_1 \supset \mathbf{AF}use_1]$ Negando la sottoformula precedente otteniamo la formula di partenza. L'insieme delle sue denotazioni è vuoto e quindi la proprietá di starvation freedom non vale in nessun stato del nostro modello (neanche nello stato inziale). La cosa può sembrare strana perchè l'algoritmo di mutua-esclusione è corretto. In effetti ciò che manca nella specifica è l'assunzione della *fairness* dell'esecuzione del sistema: dobbiamo assumere che infinitamente spesso se un processo è nella regione critica allora la rilascia. Tale proprietà non si puó esprimere in CTL (si può esprimere in LTL tramite **GF**). Esistono però modelli CTL con fairness e algoritmi di model checking estesi a tali modelli (intutivamente occorre isolare le componenti fortemente connesse che soddifano

Altri esempi Consideriamo ora la formula

$$\mathbf{AG}[wait_1 \supset \mathbf{EF}use_1]$$

Notiamo che la formula precedente é equivalente alla formula

$$\neg\mathbf{EF}\neg(\neg wait_1 \vee \mathbf{EF}use_1)$$

cioè

$$\neg\mathbf{EF}(wait_1 \wedge (\neg\mathbf{EF}use_1))$$

Per calcolare gli stati che soddisfano tale formula (cioé le sue denotazioni) occorre considerare le denotazioni delle sottoformule $wait_1$, use_1, $\mathbf{EF}use_1$, $wait_1 \wedge (\neg\mathbf{EF}use_1)$ e poi della formula stessa. Calcoliamo le denotazioni delle varie formule.

Sottoformula $\mathbf{EF}\neg use_1$ Allora abbiamo che:

$$[\![use_1]\!] = \{5, 8\}$$

A questo punto dobbiamo calcolare le denotazioni della formula **EF**, cioé il minimo punto fisso della trasformazione da insiemi di stati in insiemi di stati $f(Z) = [\![use_1]\!] \cup Pre_{\mathbf{EX}}(Z)$, dove $Pre_{\mathbf{EX}}(Z)$

denota gli stati **predecessori** degli stati in Z. Il minimo punto fisso corrisponde al limite (calcolato tramite l'unione) della catena $\emptyset, g(\emptyset), \ldots$. Quindi

$$
\begin{aligned}
&S_0 = S = \emptyset \\
&S_1 = \{5,8\} \cup Pre_{\mathbf{EX}}(S_0) = \{5,8\} \\
&S_2 = \{5,8\} \cup Pre_{\mathbf{EX}}(S_1) = \{2,4,5,8\} \\
&S_3 = \{5,8\} \cup Pre_{\mathbf{EX}}(S_2) = \{1,2,3,4,5,7,8\} \\
&S_4 = \{5,8\} \cup Pre_{\mathbf{EX}}(S_2) = \{1,2,3,4,5,6,7,8\} = S = S_i
\end{aligned}
$$

Quindi il minimo punto fisso coincide con tutti gli stati di M.

Sottoformula $(wait_1 \wedge (\neg \mathbf{EF} use_1))$ Calcolando il complemento di $[\![\mathbf{EF} use_1]\!]$ ottieniamo quindi l'insieme vuoto e quindi $[\![(wait_1 \wedge (\neg \mathbf{EF} use_1))]\!] = \emptyset$.

Sottoformula $\mathbf{EF}(wait_1 \wedge (\neg \mathbf{EF} use_1))$ Dobbiamo calcolare un nuovo punto fisso (minimo) per la trasformazione $l(Z) = \emptyset \cup Pre_{\mathbf{EX}}(Z)$. Chiaramente in questo caso otteniamo $lfp(l) = \emptyset$ e quindi $[\![\mathbf{EF}(wait_1 \wedge (\neg \mathbf{EF} use_1))]\!] = \emptyset$.

Concludiamo quindi che $[\![\mathbf{AG}[wait_1 \supset \mathbf{EF} use_1]]\!] = S$ cioè la formula vale in ogni stato del nostro modello M.
Nota: notate la differenza tra **EF** e **AF** nelle sue formule! Risulta essenziale in questo esempio considerare il branching del modello!

14 Symbolic Model Checking

L'algoritmo di model checking per CTL è basato su computazioni annidate di punto fisso. Ogni computazione di punto fisso corrisponde ad una visita del grafo associato al modello di Kripke e quindi la complessità dell'algoritmo è lineare in $|M| \times |\varphi|$ dove $|M|$ è la dimensione del modello (numero di archi e nodi) e $|\varphi|$ è la dimensione della formula Il problema è che in molti casi pratici il modello ha dimensioni enormi, infatti in genere il modello viene descritto in modo implicito tramite linguaggi ad alto livello Ad esempio un sistema concorrente verrà descritto da un'insieme di processi; il modello di Kripke corrisponderà alla composizione del comportamento di tali processi (interleaving delle loro esecuzioni) e potrebbe avere dimensione esponenziale nella specifica

Anche se il modello viene descritto in modo implicito, durante l'esecuzione dell'algoritmo di model checking si rischia di dover enumerare tutti gli stati del modello di Kripke. Questo problema viene chiamato esplosione degli stati e rappresenta l'ostacolo maggiore per la scalabilità delle tecniche di model checking. Per cercare di alleviare questo problema si utilizzano strutture dati specializzate chiamate BDDs (Binary Decision Diagrams). Vediamo prima come si rappresentano modelli di Kripke e insiemi di stati in logica proposizionale (i BDD si usano per rappresentare funzioni booleane).

14.1 Rappresentazione di Funzioni Booleane

- *Tabelle di verità*: rappresentazione di funzioni booleane non compatta; operazioni: difficili
- *Formule*: compatta; soddisfacibilità e validità: difficile; altre operazioni: facili
- *Formule in DNF*: a volte compatta; soddisfacibilità e disgiunzione: facile; altre operazioni: facile
- *Formule in CNF*: a volte compatta; validità e congiunzione: facile; altre operazioni: difficili

14.2 Rappresentazione di Modelli di Kripke

Sia $M = \langle S, R, L\rangle$ un modello di Kripke tale che $L(s) \subseteq AP$. Dobbiamo rappresentare un grafo con un numero finito di stati etichettato con simboli presi da $AP = \{p_1, \ldots, p_n\}$. Assumiamo che ogni stato abbia etichette diverse allora possiamo rappresentare lo stato s tramite un vettore di valori booleani $\langle v_1, \ldots, v_n\rangle$ dove $v_i = true$ sse $p_i \in L(s)$. In logica proposizionale possiamo quindi rappresentare ogni stato come una formula $l_1 \wedge \ldots \wedge l_n$, dove $l_i = p_i$ se $p_i \in L(s)$, e $l_i = \neg p_i$ se $p_i \notin L(s)$.

Transizioni Le transizioni si possono rappresentare introducendo i simboli $p'_1, \ldots, p'_n$ che rappresentano le postcondizioni sugli stati raggiunti dalle transizioni (archi del modello). Quindi un arco da s a s' si rappresenta come la formula

$$l_1 \wedge \ldots \wedge l_n \wedge l'_1 \wedge \ldots \wedge l'_n$$

dove $l_i = p_i$ se $p_i \in L(s)$, $= \neg p_i$ altrimenti $l'_i = p'_i$ se $p_i \in L(s')$, $= \neg p'_i$ altrimenti. Quindi l'insieme delle transizioni (archi) del modello si rappresenta nel suo complesso come una disgiunzione di congiunzioni

$$\bigvee_{i=1}^{k} (l_{i1} \wedge \ldots \wedge l_{in} \wedge l'_{i1} \wedge \ldots \wedge l'_{in})$$

Insiemi di Stati Usando la stessa idea possiamo rappresentare un'insieme finito di stati tramite una disgiunzione di formule booleane (la funzione caratteristica dell'insieme). Un insieme $s_1, \ldots, s_k$ si rappresenta infatti come la disgiunzione

$$\bigvee_{i=1}^{k} (l_{i1} \wedge \ldots \wedge l_{in})$$

dove $l_{ij} = p_{ij}$ se $p_{ij} \in L(s_i)$ e $l_{ij} = \neg p_{ij}$ se $p_{ij} \notin L(s_i)$.
In altre parole possiamo utilizzare disgiunzioni di congiunzioni (DNF) come rappresentazione simbolica di modelli e insiemi di stati

Operazioni per l'algoritmo di model checking Per implementare l'algoritmo di model checking occorre implementare le operazioni per la manipolazione di insiemi di stati: unione, intersezione, equivalenza, e l'operatore $Pre_{\mathbf{EX}}$ che dato un'insieme di stati restituisce l'insieme dei suoi successori

- Unione=disgiunzione
- Intersezione=congiunzione
- Equivalenza=doppia implicazione

Rimane da implementare l'operatore $Pre_{\mathbf{EX}}$: useremo formule booleane quantificate.

Formule booleane quantificate (QBF) Introduciamo una forma semplice di quantificazione su simboli proposizionali

$$\exists p.F \equiv (F_{|p=true}) \vee (F_{|p=false})$$

dove $F_{|p=v}$ rappresenta la formula calcolata assegnando v ($true$ o $false$) al simbolo p.

Operatore $Pre_{\mathbf{EX}}$ Usando quest'idea e dato un'insieme di stati S rappresentato da una formula $\bigvee_{i=1}^{k} F_i$ (definita su $p_1, \ldots, p_n$) e data un modello M le cui transizioni sono rappresentate tramite la formula $\bigvee_{j=1}^{n} G_j$ (definita su $p_1, \ldots, p_n, p'_1, \ldots, p'_n$). Ridenominiamo la formula F_i come F'_i sostituendo p_r con p'_r per $r : 1, \ldots, n$. Allora $Pre_{\mathbf{EX}}(S)$ è rappresentato dalla seguente formula

$$\exists p'_1 \ldots . \exists p'_n . \bigvee_{j=1}^{n} G_j \wedge \bigvee_{i=1}^{k} F'_i$$

dopo aver applicato la definizione di $\exists$ dobbiamo mettere la formula in norma normale (disgiunzione di congiunzioni)

Example 8 *Sia* $AP = \{idle, busy, wait, think, use\}$*, consideriamo la transizione*

$$\begin{aligned} F = (&idle \wedge \neg busy \wedge wait \wedge \neg think \wedge \neg use) \wedge \\ &[(\neg idle' \wedge busy' \wedge \neg wait' \wedge \neg think' \wedge use') \vee \\ &(\neg idle' \wedge busy' \wedge \neg wait' \wedge think' \wedge \neg use')] \end{aligned}$$

e l'insieme

$$\begin{aligned} G = (&\neg idle \wedge busy \wedge \neg wait \wedge think \wedge \neg use) \vee \\ &(\neg idle \wedge \neg busy \wedge \neg wait \wedge think \wedge use) \end{aligned}$$

allora $Pre_{\mathbf{EX}}(G)$ *è rappresentato dalla formula*

$$\begin{aligned} &\exists idle' \ldots . \exists use' . \\ &\quad [F \wedge ((\neg idle' \wedge busy' \wedge \neg wait' \wedge think' \wedge \neg use') \vee \\ &\qquad (\neg idle' \wedge \neg busy' \wedge \neg wait' \wedge think' \wedge use'))] \end{aligned}$$

Rappresentazione basata su QBF e state explosion La rapppresentazione basata su formule booleane non elimina il problema dell'esplosione degli stati. Se manteniamo le formule come disgiunzioni di congiunzioni rischiamo di produrre formule di grandi dimensioni. Abbiamo bisogno di rappresentare formule booleane nel modo più compatto possibile: utilizzeremo i BDDs.

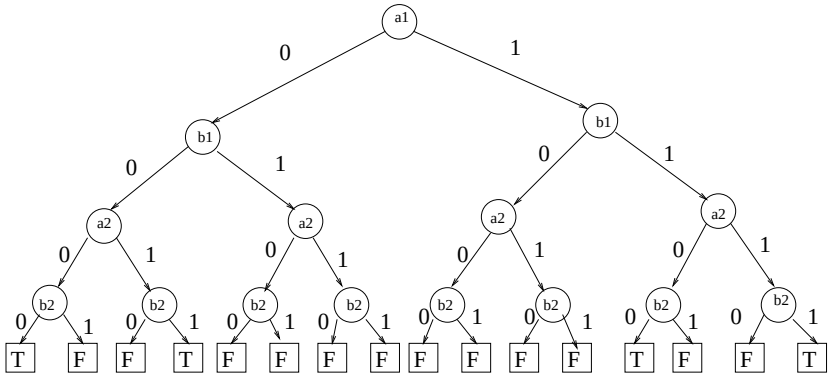
Binary Decision Diagrams Struttura dati per rappresentare funzioni booleane cioè funzioni $f, g, \ldots$ da $Bool^n$ a $Bool$. Esempio di funzione booleana $f(x, y) = (x \wedge y) \vee (\neg y)$. Ricordiamo che determinare la *soddisfacibilità* di una formula è un problema NP-completo, mentre determinare la sua *validità* è un problema co-NP completo. Quindi nel caso peggiore soddisfacibilità e validità hanno costo esponenziale. In molti casi pratici però possiamo fare meglio

Binary Decision Trees Una formula booleana si può rappresentare tramite un albero binario di decisione. I nodi non terminali sono etichettati da proposizioni atomiche e hanno due archi (*low* e *high*) uscenti etichettati rispettivamente *true* e $false$ (valori possibili della proposizione che etichetta il nodo). Le foglie sono etichettate con *true* o $false$. Il valore della funzione sui suoi input si ottiene attraversando i nodi di decisione dell'albero dalla radice fino alle foglie.

Teorema di Espansione di Shannon Gli alberi binari di decisione si basano sul teorema di espansione di Shannon

$$f(x_1, \ldots, x_i, \ldots, x_n) = (x_i \wedge f_{|x_i = true}) \vee (\neg x_i \wedge f_{|x_i = false})$$

dove $f_{|x_i = v} = f(x_1, \ldots, x_{i-1}, v, x_{i+1}, \ldots, x_n)$ è chiamato *cofattore*. La seguente figura illustra un esempio di albero di decisione.



f(a1,a2,b1,b2) = (a1 <--> b1) and (a2 <--> b2)

Binary Decision Diagrams (BDDs) Gli alberi binari di decisione hanno dimensione esponenziale nella dimensione della formula. Per compattare la rappresentazione dobbiamo eliminare le ridondanze:

- Combiniamo tutti i sottoalberi isomorfi.
- Tutti i nodi con figli isomorfi vengono eliminati.

Otteniamo così un D.A.G (grafo diretto aciclico). Rimane il problema di trovare una rappresentazione canonica

Ordered Binary Decision Diagrams (OBDDs) Ordiniamo le variabili in modo che:

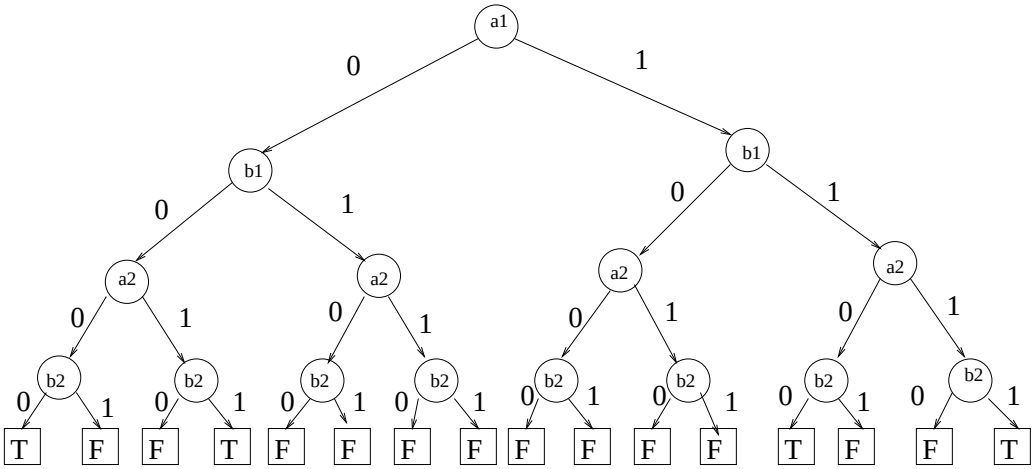
- In ogni cammino dalla radice alle foglie manteniamo lo stesso ordine, senza ripetizioni
- Non tutte le variabili devono comparire lungo un cammino
- Non devono esserci sottoalberi isomorfi o vertici ridondanti nel diagramma

Otteniamo una rappresentazione *canonica* di una funzione booleana chiamata OBDD (Ordered BDD). I BDDs rappresentano un'euristica per rappresentare in modo compatto funzioni booleane, l'ordine delle variabili è fondamentale. Tuttavia esistono funzioni booleane con BDDs esponenziali (es. moltiplicazione) per ogni ordine delle variabili

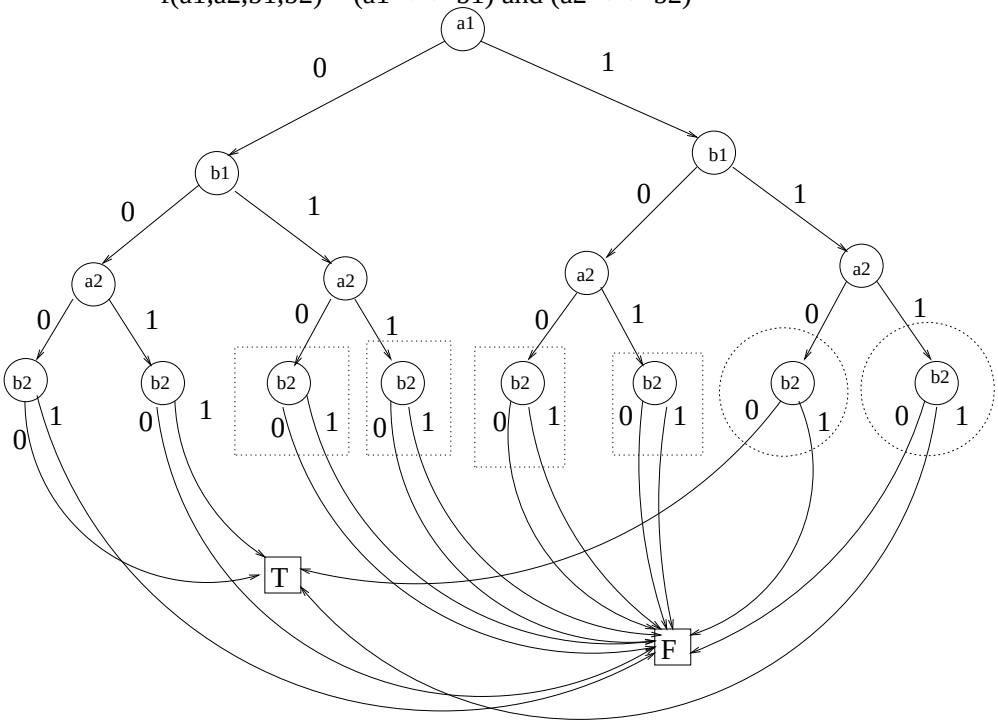
Algoritmo per ottenere un OBDD Dopo aver ordinato le variabili, per ottenere un OBBD basta applicare le seguenti regole fino a che è possibile

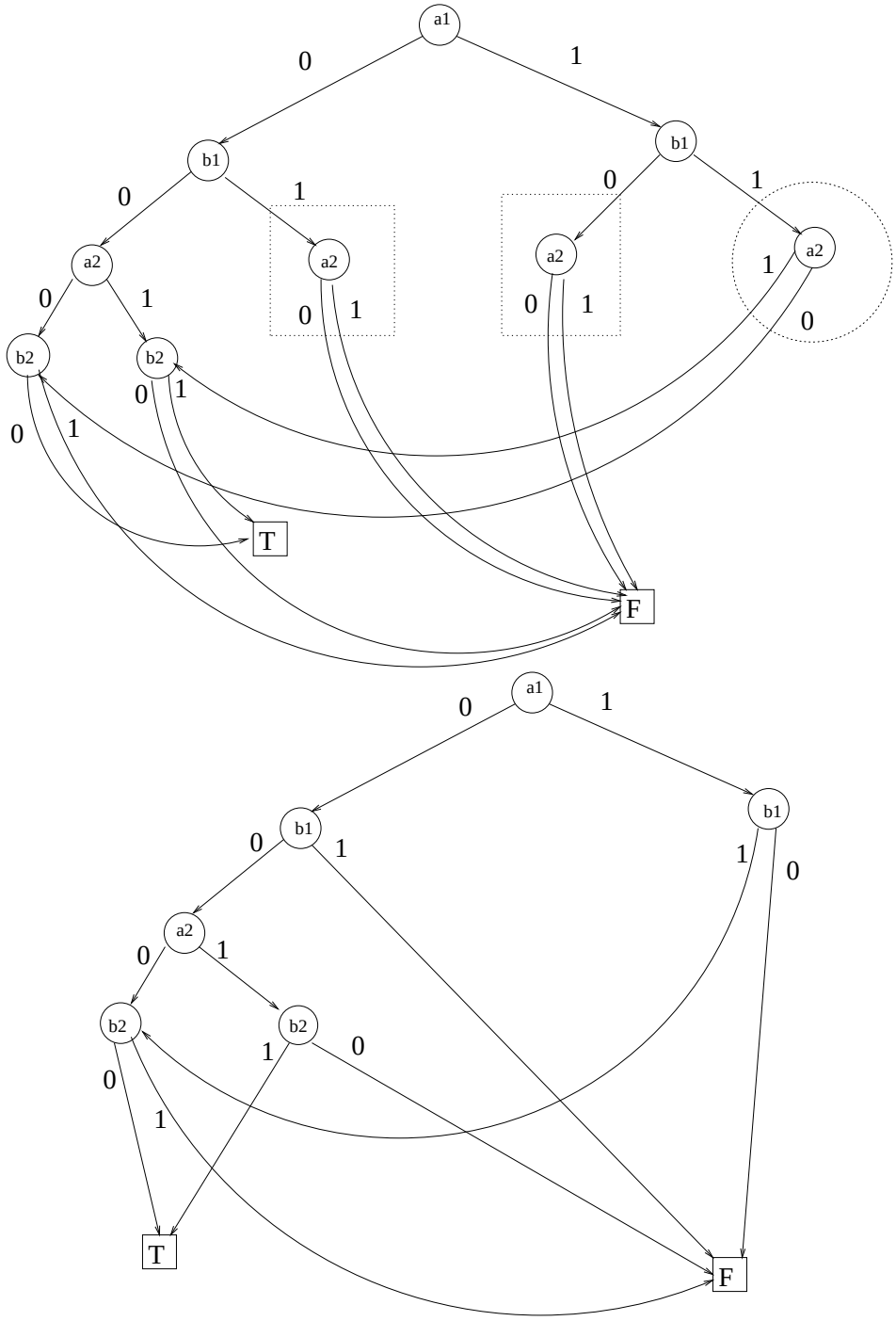
- Per ogni etichetta ($true$, $false$), eliminare tutti i nodi *terminali* tranne uno
- Se due *non terminali* u e v hanno la stessa etichetta (variabile) e gli stessi figli, eliminare uno dei due e ridirezionare i suoi archi entranti all'altro vertice
- Se un *non terminale* u ha due figli uguali, allora eliminare u e ridirezionare i nodi entranti all'unico figlio

Questo algoritmo si può implementare tramite una visita bottom up del grafo. Le seguenti figure illustrano i vari passi di riduzione.



f(a1,a2,b1,b2) = (a1 <--> b1) and (a2 <--> b2)





Operazioni su BDDs Due formule rappresentate da OBDDs sono equivalenti sse gli OBDDs sono isomorfi. Inoltre possiamo controllare se una funzione è soddisfacibile confrontando il suo OBDD con l'OBDD per $false$. Le altre operazioni booleane si possono implementare in modo efficiente usando l'espansione di Shannon e la funzione ricorsiva *Apply* (docuta a Bryant) definita

come segue. Sia $\Box$ un operatore booleano, f e f' due funzioni booleane, v e v' le radici dei loro OBDDs con etichette x e x', vogliamo calcolare lo OBDD per $f \Box f'$.

Funzione Apply

- Se v e v' sono terminali allora $f \Box f' = value(v) \Box value(v')$
- Se $x = x'$ allora usiamo la formula di Shannon per ridurre il problema in sottoproblemi risolti ricorsivamente

  $$f \Box f' \ = \ (\neg x \wedge (f_{|x=false} \Box f'_{|x=false})) \vee (x \wedge (f_{|x=true} \Box f'_{|x=true}))$$

  l'OBDD risultante avrà un nuovo nodo etichettato con x e low e $high$ che puntano agli OBDD calcolati ricorsivamente
- Se $x < x'$ allora $f_{|x=true} = f'_{|x=false} = f'$ poichè f' non dipende da x.
  Quindi

  $$f \Box f' \ = \ (\neg x \wedge (f_{|x=false} \Box f')) \vee (x \wedge (f_{|x=true} \Box f'))$$

  e i sotto OBDDs sono calcolati ricorsivamente
- Si procede in modo simile se $x > x'$

Ottimizzazioni per la Funzione Apply L'algoritmo diventa polinomiale se si mantegono in una tabella di hashing tutti i sottoproblemi calcolati durante l'esecuzione della funzione *Apply* (*caching* come in programmazione dinamica). Tali OBDDs si possono utilizzare prima di invocare la funzione sui sottocasi. Il numero di sottoproblemi dipende dal numero di nodi dell'OBDD (ogni sotto grafo è determinato dalla sua radice). Usando caching la complessità diventa lineare nel prodotto della dimensione degli OBDDs per f e f'. Se si estende la tecnica in modo da gestire *foreste* di OBDDs (con le condizioni viste sopre) molte operazioni diventano costanti (ad es. equivalenza diventa costante: basta controllare se i due OBDDs puntano allo stesso grafo della foresta). I pacchetti per gestire OBDDs possono gestire grafi con milioni di nodi (e quindi con un numero astronomico di stati).

Riepilogo: Rappresentazione di Funzioni Booleane

- *Tabelle di verità*: non compatta; operzioni: difficili
- *Formule*: compatta; soddisfacibilità e validità: difficile; altre operzioni: facili
- *Formule in DNF*: a volte compatta; soddisfacibilità e disgiunzione: facile; altre operazioni: facile
- *Formule in CNF*: a volte compatta; validità e congiunzione: facile; altre operazioni: difficili
- OBDDs: spesso compatta; soddisfacibilità e validità: facile; altre operazioni: facili

BDDs-based Model Checking Possiamo utilizzare i BDDs per rappresentare sia i modelli di Kripke che gli insiemi di stati calcolati durante il model checking. Abbiamo visto infatti che un modello si può rappresentare come una formula DNF e quindi tramite un OBDD (sulle variabili $AP = p_1, \ldots, p_n$ e $p'_1, \ldots, p'_n$). La stessa cosa per gli insiemi di stati. L'operazione delicata è ancora una volta $Pre_{\mathbf{EX}}$ che abbiamo visto si può rappresentare tramite una formula QBF $\exists p'_1 \ldots . \exists p'_n . H$. La formula H è rappresentazbile tramite la congiunzione dell'OBDD per il modello con l'OBDD per l'insieme di stati di partenza. La quantificazione esistenziale si risolve usando ancora una volta la formula di Shannon.

Symbolic Model Checking L'algoritmo di Symbolic Model Checking utilizza gli OBBDs come rappresentazione simbolica di transizioni e insiemi di stati. Ogni computazione di punto fisso viene implementata tramite le operazioni sugli OBDDs. Quindi il symbolic model checking si può vedere come un calcolo iterativo di trasformazione di grafi (OBDDs). Queste tecniche sono implementate in tool come SMV e nuSMV.

15 Constraint-based Model Checking

Gli algoritmi di model checking e symbolic model checking per logica temporale si possono implementare in modo efficiente tramite strutture dati per rappresentare funzioni booleane. Queste tecniche operano su conoscenza rapppresentabile tramite modelli di Kripke con un numero finito di stati. A causa di questa restrizione in molti casi pratici occorre applicare delle *astrazioni* per passare dal modello reale al modello formale che poi viene trattato automaticamente. Sembra naturale chiedersi se sia possibile estendere tali tecniche a modelli più espressivi

Finite-state Model Checking Sia M un modello di Kripke finito, φ una proprietà *temporale* ed s uno stato, il problema di model checking: $M, s \models \varphi$?
Stato = assegnamento a variabili booleane
Relazione di transizione = formula booleana
Operatore per calcolare predecessori Pre = Formula booleana quantificata
Algoritmi di Model Checking: Computazione di punto fisso utilizzando formule booleane come rappresentazione simbolica di *insiemi finiti* di stati.

Ad Esempio: Safety Properties

$M \models \mathbf{AG}(\Phi)$
Sia Φ un insieme di stati **buoni**, allora la proprietà vale se
"per ogni cammino $\tau = s_1 s_2 \ldots$ in M e *tutti* gli stati s_i in τ, $s_i \in \Phi$"

Proprietà di Logica Temporale
$\mathbf{AG}(\Phi) \equiv \neg\mathbf{EF}(\neg\Phi)$ e $\mathbf{EF}(\Psi) = \mu x.\ \Psi \vee Pre(x)$
Dove $\Psi = \neg\Phi$ è l'insieme di stati **cattivi**!

Verifica della proprietà = Raggiungibilità
Sia s_0 lo stato iniziale in M, allora $\mathbf{AG}(\Phi)$ vale sse $s_0 \notin \mathbf{EF}(\neg\Phi)$

Punti fondamentali

Terminazione
Almeno in principio non è un problema per sistemi 'finiti'

Esplosione degli stati
La terminazione è un problema in pratica!

Solutioni/euristiche
Symbolic Model checking [McM93] si basa su rappresentazioni di funzioni booleane con Ordered Binary Decision Diagrams (OBDDs)

Infinite-state Model Checking

Nuovo Problema

Sia M una struttura *infinita* e φ una proprietà temporale: $M \models \varphi$?

Rappresentazione simbolica

Stato = assegnamento a variabili con tipi eterogenei (integers, reals, ...)
Relazione di transizione = ?
Relazione di Predecessori Pre = ?

Algoritmi di Model Checking (?)

Computazione di punto fisso basate su ? come rappresentazione simbolica di insiemi *infiniti* di stati

Da formule booleane a constraints I constraints (vincoli) sono relazioni definite du dati eterogenei (ad es disequazioni lineari). Un constraint solver (risolutore di vincoli) offre operazioni efficienti per manipolare simbolicamente tali relazioni. Possiamo usare i *contraints* per rappresentare simbolicamente

- una relazione di transizione = digiunzione di constraints
- un insieme di stati = constraint
- la relazione dei predecessori = disgiunzione di constraints quantificati esistenzialmente

Abbiamo gia visto degli esempi di constraints: formule proposizionali e BDDs (entrambi per rappresentare insiemi finiti di stati). Per insiemi infiniti di stati: constraint aritmetici, uguaglianza su termini del prim'ordine, espressioni regolari. Ad esempio: constraint aritmetici lineari

$$x + 2y + z \geq 1, x \geq 3, y \leq 6$$

Operazioni: congiunzione, disgiunzione, quantificazione esistenziale. Soddisfacibilità = ad es. simplesso.

Procedure di Verifica di Proprietà Temporale

Per riformulare gli algoritmi basati su computazioni di punto fisso usati per sistemi finiti tramite constraints occorrono i seguenti requisiti:

- La proprietà e gli stati iniziali si devono poter esprimere attraverso constraints
- I constraints devono essere effettivamente chiusi rispetto alla disgiunzione
- L'equivalenza di constraints deve essere decidibile
- Deve esistere un algoritmo per calcolare la relazione dei predecessori

Safety Properties Se possiamo rappresentare $\mathbf{Pre}(\Phi)$ per una disgiunzione di constraint Φ, allora applicamdo l'assioma di espansione per **EF** ottemiamo

$$\mathbf{EF}(\Psi) = \Psi \vee \mathbf{Pre}(\Phi) \vee \ldots \vee \mathbf{Pre}^k(\Phi)$$

Note: Invece di risolvere un solo *constraint problem*, usiamo il constraint solver *ad ogni passo* di una computazione di punto fisso per risolvere *molti* constraint problems di dimensione relativamente piccola (per il calcolo di **Pre** ad ogni passo). In generale la terminazione delle procedure di model checking basate su constraints non può essere garantita a priori (semi-algoritmi).

Terminazione
La teoria dei well quasi orderings (WQOs) può essere usata per trovare frammenti decidibili

Denotazione la denotazione $[\![\varphi]\!]$ di $\varphi \in \mathcal{C}$ è l'insieme delle sue soluzioni *solutions* w.r.t. a fixed domain

Ad esempio: $[\![x \geq 1, y \geq 3]\!] = \{\langle x:1, y:3\rangle, \langle x:2, y:7\rangle, \langle x:3, y:4\rangle, \ldots\}$

Entailment $\sqsubseteq$ è un ordinamento tra constraints $\mathcal{C}$ tale che $\varphi \sqsubseteq \psi$ implies $[\![\psi]\!] \sqsubseteq [\![\varphi]\!]$

Ad esempio: $x \geq 1, y \geq 3 \sqsubseteq x \geq 0, y \geq 1$

WQO
$\langle \mathcal{C}, \sqsubseteq \rangle$ è un **Well Quasi Ordering** sse per ogni sequenza infinita $\varphi_1\varphi_2\ldots$ in $\mathcal{C}$ esistono due indici $i < j$ tali che $\varphi_i \sqsubseteq \varphi_j$

Applicazione ai Sistemi Sincroni Parametrici

Protocolli per rappresentare sistemi con collezioni di tanti processi identici (clienti) e un processo monitor

Il protocollo è definito tramite una collezione di macchine locali

Ogni componente individuale viene modellata come una macchina a stati finiti

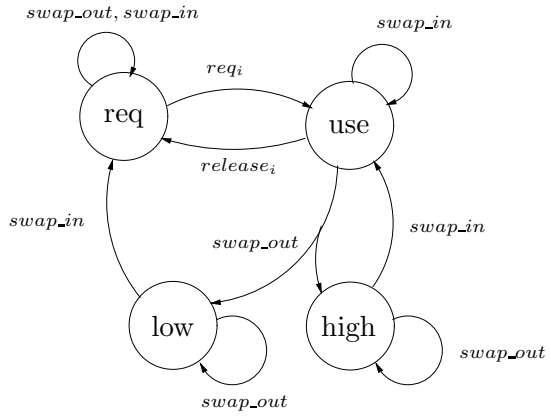
Le macchine interagiscono tramite scambio di messaggi

- Rendez-vous (il mittente e ricevente si sincronizzano su un messaggio)
- Broadcast (il mittente e tutti i riceventi si sincronizzano su un messaggio)
- La reazione ad un messaggio broadcast può essere non deterministica

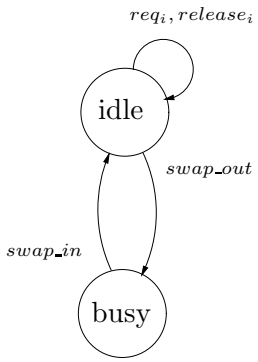
Esempio: *Load Balancing* Monitor

- Vogliamo modellare un sistema con un monitor di *load balancing*
- Nella configurazione iniziale del sistema i processi sono nello stato *req*, mentre il monitor è in stato *idle*.
- Quando il monitor manda il messaggio *swap_out* l'esecuzione di tutti i processi viene sospesa
- Due tipi di priorità sono assegnati ai processi sospesi
- Quando il monitor rilascia la CPU essa viena assegnata ai processi con priorità maggiore

Modello formale



Process P_i



Monitor

Possibile Astrazione

- Applichiamo la seguente astrazione
- Dato uno stato globale

$$Monitor : \langle Idle \rangle,\ Clients : \langle Req, Low, Low, Use, \ldots \rangle$$

contiamo quanti processi ci sono in ogni stato locale

$$Monitor : \langle Idle \rangle,\ Clients : \langle x_{Req} = 1, x_{Low} = 2, x_{Use} = 1, \ldots \rangle$$

Protocollo Astratto = Estensioni di Reti di Petri Petri

- Transizioni
  - Locazioni (stati del monitor),
  - variabili intere (x_s per ogni stato locale s);
- Nel nostro esempio

$$\text{(1) } x_{req} \geq 1,\ x_{idle} \geq 1 \longrightarrow x'_{req} = x_{req} - 1,\ x'_{use} = x_{use} + 1.$$

$$\text{(2) } x_{use} \geq 1,\ x_{idle} \geq 1 \longrightarrow x'_{use} = x_{use} - 1,\ x'_{req} = x_{req} + 1.$$

$$\text{(3) } x_{idle} \geq 1,\ \mathbf{x'_{high}} \geq \mathbf{x_{high}},\ \mathbf{x'_{low}} \geq \mathbf{x_{low}} \longrightarrow x'_{idle} = x_{idle} - 1,\ x'_{busy} = x_{busy} + 1,\ \mathbf{x'_{high}} + \mathbf{x'_{low}} = \mathbf{x_{high}} + \mathbf{x_{low}} + \mathbf{x_{use}},\ x'_{use} = 0.$$

$$\ldots$$

Verifica Parametrica

Stati Iniziali rappresentati come

$$I \ = \ \ell_M = Idle \ \wedge x_{Req} \geq 1 \wedge x_{Use} = 0 \wedge \ldots$$

Mutua esclusione **AG**(*Un solo cliente ha accesso esclusivo*)

Violazioni $\quad U \;=\; (x_{Use} \geq 1 \wedge x_{Use} \geq 1) \;\vee\; (x_{Use} \geq 2)$

Verifica = Raggiungibilità

Possiamo raggiungere uno stato in U da un'istanza di I?

Constraint-based Symbolic Model Checking

Rappresentazione Simbolica = Constraints su interi

$$\begin{array}{rcl} [\![x_{req} \geq 2]\!] & = & \{\langle Idle, 2, 0, \ldots\rangle, \langle Busy, 3, 1, \ldots\rangle, \ldots\} \\ & = & \{\langle Idle, Req, Req\rangle, \langle Busy, Req, Req, Req, Use\rangle, \ldots\} \end{array}$$

Entailment Test

$\varphi \sqsubseteq \psi \quad$ *if and only if* $\quad [\![\psi]\!] \subseteq [\![\varphi]\!]$

Symbolic Predecessor Operator

$$\mathbf{sym\_pre}(\varphi(\mathbf{x}')) = \bigvee_{i \in I} \;\; \exists\, \mathbf{x}'.\; \psi_\tau(\mathbf{x}, \mathbf{x}') \;\wedge\; \varphi(\mathbf{x}')$$

Backward Reachability

Proc Symb-Reach(E : EFSM, Φ_o, Φ_f : sets of constraints)

$\Phi := \Phi_f$
$\Psi := \emptyset$
while $\Phi \neq \emptyset$ **do**
  $\Phi := \Phi \setminus \{\varphi\}$
  if there are <u>no</u> $\psi \in \Psi$ **s.t.** $\psi \sqsubseteq \varphi$ **then**
    $\Psi := \Psi \cup \{\varphi\}$
    $\Phi := \Phi \cup \mathbf{sym\_pre}(E, \varphi)$
endwhile
if $sat(\Phi_o \wedge \Psi)$ **then return** $'\Phi_f$ *is reachable from* Φ'_o;
else return $'\Phi_f$ *is not reachable from* Φ'_o.

Per rendere le operazioni efficienti

Polyhedra

Si possono usare *poliedri* per rappresentare insiemi di punti di interi

Relaxation integer-reals

Test di soddisfacibilià, entailment, ed eliminazione di variabili sono eseguiti sui *reali* invece che sugli interi

Backward Reachability sui *Reali*

Proc Symb-Reach-over-$\mathbb{R}(E : \text{EFSM}, \Phi_o, \Phi_f : \text{sets of constraints})$

$\Phi := \Phi_f$
$\Psi := \emptyset$
while $\Phi \neq \emptyset$ **do**
  $\Phi := \Phi \setminus \{\varphi\}$
  if there are <u>no</u> $\psi \in \Psi$ **s.t.** $\psi \sqsubseteq_{\mathbb{R}} \varphi$ **then**
    $\Psi := \Psi \cup \{\varphi\}$
    $\Phi := \Phi \cup \mathbf{sym\_pre}_{\mathbb{R}}(E, \varphi)$
endwhile
if $sat_{\mathbb{R}}(\Phi_o \wedge \Psi)$ **then return** $'do\ not\ know'$;
else return $'\Phi_f\ is\ not\ reachable\ from\ \Phi'_o$.

Proprietà Interessanti

- La procedura di Backward reachability termina sempre quando i constraint hanno la seguente forma:

$$x_1 \geq c_1, \ldots, x_n \geq c_n$$

c_i è un intero non negativo $i : 1, \ldots, n$

16 Parte IV: Planning as Symbolic Model Checking

Nella parte finale del corso abbiamo visto un'introduzione a problemi di planning, un campo di ricerca classico in intelligenza artificiale. In questa sezione vedremo le connessioni tra planning per domini non deterministici, logica CTL e model checking [11].

Planning in Domini Non-deterministici Dato un modello formale del comportamento di un agente (stati, azioni, ...) un problema di *planning* consiste nel derivare un *piano* (sequenza di azioni) che da uno stato iniziale porta ad un certo goal. In questo ambito uno degli argomenti di ricerca più interessanti riguarda problemi di planning *non deterministico*, cioè dove il dominio di planning (comportamento degli agenti) non è deterministico. Questa estensione pone numerosi problemi concettuali e pratici

- Un piano dipende dal non determinismo del dominio e quindi può avere più di una sequenza di azioni e stati
- Vi sono diverse nozioni di soluzione per un problema di planning, ad esempio, un piano in cui esiste la possibilità di raggiungere il goal, un piano che garantisce il raggiungimento dei goal nonostante il non determinismo nel dominio, ecc.

Recentemente è stata stabilita una connessione tra planning e symbolic model checking che permette di gestire problemi per domini non deterministici come problemi di model checking per formule CTL. Questa estensione permette di formulare in modo concettualmente pulito diversi variazioni di problemi di planning tramite opportune formule CTL. Ad esempio

- un goal quale **AF** g permette di specificare condizioni che devono essere raggiunte da un piano indipendentemente dal non-determinismo nel modello (strong plan)
- al contrario un goal del tipo **EF** g permette di specificare condizioni che possono essere raggiunte ma senza garanzie di successo (weak plan)

La connessione tra problemi di planning e model checking permette inoltre di formulare algoritmi efficienti per trattare domini di planning di grandi dimensioni

16.1 Dominio di Planning non deterministico

Un dominio di planning non deterministico è una tupla $\langle P, S, A, R\rangle$ dove

- P è un insieme di proposizioni
- $S \subseteq 2^P$ è un insieme di stati
- A è l'insieme (finito) di azioni
- $R \subseteq S \times A \times S$ è la relazione di transizione

Uno stato s è rappresentato tramite la collezione di proposizioni vere in s La relazione di transizione descrive l'effetto dell'esecuzione di un azione. L'insieme di azioni eseguibili in $s \in S$ viene indicato

$$Act(s) = \{a : \exists s'.R(s, a, s')\}$$

L'insieme di stati raggiungibili da s tramite l'azione a viene indicato

$$Exec(s, a) = \{s' : R(s, a, s')\}.$$

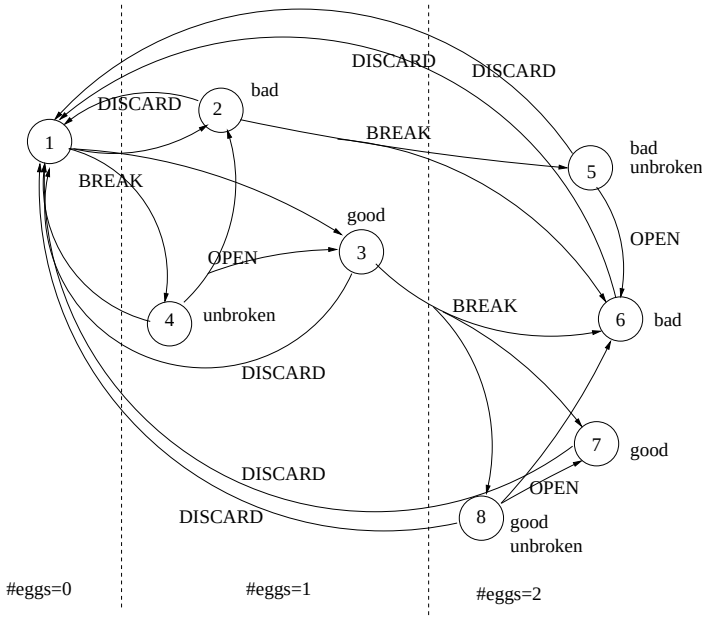


Figura 10: Dominio non deterministico: Preparazione di una Omelette

Example 9 *La Fig. 10 illustra un esempio di dominio non deterministico. L'esempio rappresenta possibili strategie per preparare una omelette. Per preparare una omelette con n uova utilizziamo una scodella nella quali dobbiamo riuscire a rompere n uova, e tutte le uova devono essere buone. Bad indica che la scodella contiene almeno un uovo cattivo, good che tutte le uova sono buone, unbroken che c'e' un uovo non ancora rotto. Il flag #eggs=i denota il numero di uova nella scodella (buone o cattive). L operazioni sono BREAK (tentativo di rompere un uovo), OPEN (apre un uovo unbroken), DISCARD (in ogni momento si può buttare via il contenuto della scodella e riiniziare tutto da capo). Il non determinismo di azioni quali BREAK modella il fatto che non possiamo sapere se un uovo è buono o cattivo prima di aprirlo o se riusciremo ad aprirlo buttandolo nella scodella.*

Quando eseguito un piano deve essere in grado di monitorare l'ambiente circostante e fare delle selte dipendenti dallo stato corrente dei sensori. Questo tipo di piani si possono rappresentare tramite una State-Action Table π definita come un'insieme di coppie $\{\langle s, a\rangle \mid s \in S,\ a \in Act(s)\}$. Una state-action table è non deterministica se per un qualche stato s esistono più coppie $\langle s, a\rangle$. A partire da una state-action table π (un piano) per un dominio $\langle P, S, A, R\rangle$ possiamo rappresentare le possibili esecuzioni del piano a partire dagli stati iniziali $I \subseteq S$ attraverso una sorta di modello di Kripke in cui i nodi sono gli stati che si possono raggiungere a partire da stati del piano, e gli archi rappresentano una possibile esecuzione di un'azione.

Formalmente la struttura di esecuzione di π a partire da $I \subseteq S$ è una coppia $K = \langle Q, T\rangle$ dove $Q \subseteq S$ e $T \subseteq S \times S$ e

- se $s \in I$ allora $s \in Q$
- se $s \in Q$ e esiste una coppia $\langle s, a\rangle \in \pi$ tale che $R(s, a, s')$ allora $s' \in Q$ e $T(s, s')$

$s \in Q$ è uno *stato terminale* se non esiste $s' \in Q$ tale che $T(s, s')$ Un *cammino di esecuzione* in K è una sequenza (anche infinita) di stati $s_0 s_1 \ldots$ tali che o s_i è uno stato terminale oppure $T(s_i, s_{i+1})$ K è aciclico se tutti i suoi cammini sono finiti

Example 10 *La Fig. 11 mostra la struttura di esecuzione per il piano* $\pi = \{\langle 1, break\rangle, \langle 3, break\rangle$.

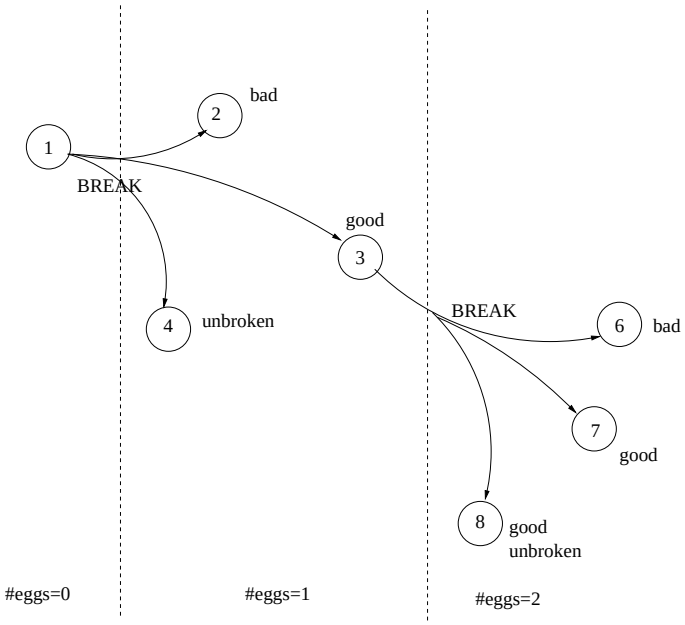


Figura 11: Struttura di esecuzione

16.2 Planning Problem, Weak and Strong Solutions

Dato un dominio $D = \langle P, A, S, R\rangle$, un problema di planning per D è una tripla $\langle D, I, G\rangle$ dove $I \subseteq S$ (stati iniziali) e $G \subseteq S$ (stati obbiettivo). Una soluzione ad un problema di planning può avere diverse caratteristiche. Sia π un piano (state-action table) deterministico con struttura di esecuzione $K = \langle Q, T\rangle$

- Una *weak solution* è un piano π che può raggiungere il goal ma non garantisce tale risultato, cioè per ogni stato in I esiste un cammino in cui lo stato terminale è in G
- Una *strong solution* è un piano π che garantisce il raggiungimento del goal (tutti i cammini di esecuzione di K terminano con un goal), cioè K è aciclico e tutti gli stati terminali di K sono in G

Le strong cyclic solutions rappresentano un'ulteriore possibile nozione di soluzione ad un problema di planning nella quale vengono presi in considerazione solo cicli *buoni*. Una strong cyclic solution formalizza l'idea di piano basato su una strategia *iterativa trial-and-error* nella quale è possibile ripetere una sequenza di azioni fino a che ha successo (ad esempio pick up a block until succeed). In questo caso si considerano ammissibili sono i piani per i quali tutti i cammini di esecuzione hanno sempre la possibilità di terminare (vogliamo evitare sequenze infinite di insuccessi) e quando

terminano garantiscono il raggiungimento del goal Formalmente una *strong cyclic solution* π è tale che: tutti gli stati terminali della sua struttura di esecuzione K sono in G ed inoltre per ogni stato q in Q esiste uno stato terminale raggiungibile da q

16.3 Algoritmi per Estrarre Piani

Per semplicità ci limiteremo al caso di weak e strong solutions che come accennato in precedenza possono essere ottenuti attraverso goal formulati come le formule CTL **EF** g e **AF** g. Sulla base di questa idea non è difficile immaginare che gli algoritmi siano basati su computazioni di minimo punto fisso effettuate tramite breadth first search a partire dalla condizione g (cioè come in model checking si utilizza ragionamento backwards).

Algoritmi per Estrarre Weak Solutions L'algoritmo costruisce incrementalmente una state-action table SA tramite una visita BFS a partire dal goal. La funzione WeakPreImage viene utilizzata per calcolare gli stati predecessori degli stati della state-action table ($State(SA)$) corrente

$$WeakPreImage(S) = \{\langle s, a\rangle \mid Exec(s, a) \cap S \neq \emptyset\}$$

Il risultato di $WeakPreImage$ viene passato alla funzione $PruneState$ che rimuove tutte le coppie $\langle s, a\rangle$ per le quali si conosce già una soluzione per s

$$PruneState(\pi, S) = \{\langle s, a\rangle \in \pi \mid s \notin S\}$$

L'algoritmo termina con successo se l'insieme di stati iniziali è contenuto negli stati della state-action table uniti ai goal. Se si invece si raggiunge un punto fisso e la condizione precedente non è soddisfatta l'algoritmo fallisce. Ricordate che le weak solution sono caratterizzabili come **EF**.

```
function WeakPlan(I, G);
begin
var OldSA, SA : state action tables;
OldSA := Fail;
SA := ∅;
while (OldSA ≠ SA ∧ I ⊈ (G ∪ States(SA))) do
    PreImage := WeakPreImage(G ∪ States(SA));
    NewSA := PruneState(PreImage, G ∪ States(SA));
    OldSA := SA;
    SA := SA ∪ NewSA;
endw;
if (I ⊆ G ∪ States(SA)) then returnSA;
else return Fail;
end
```

Algoritmi per Estrarre Strong Solutions L'algoritmo per estrarre una soluzione strong è identico a WeakPlan tranne che per il calcolo della preimage. La funzione WeakPreImage viene sostituita con la funzione StrongPreImage utilizzata per calcolare gli stati predecessori che garantiscono il raggiungimento degli stati della state-action table ($State(SA)$) corrente

$$StrongPreImage(S) = \{\langle s, a\rangle \mid Exec(s, a) \neq \emptyset,\ Exec(s, a) \subseteq S\}$$

Ricordate che le strong solutions sono ottenute tramite la formula **AF**.

```
function StrongPlan(I, G);
begin
var OldSA, SA : state action tables;
OldSA := Fail;
SA := ∅;
while (OldSA ≠ SA ∧ I ⊈ (G ∪ States(SA))) do
    PreImage := StrongPreImage(G ∪ States(SA));
    NewSA := PruneState(PreImage, G ∪ States(SA));
    OldSA := SA;
    SA := SA ∪ NewSA;
endw;
if (I ⊆ G ∪ States(SA)) then returnSA;
else return Fail;
end
```

Strong Cyclic Solutions Mentre weak e strong plans non restituiscono piano non iterativi una strong cyclic solution deve *controllare* i cicli e scartare quelli cattivi. Un ciclo è cattivo se una volta entrati non è più possibile raggiungere il goal. Intuitivamente l'algoritmo si può evincere dalla formulazione di una strong cyclic solutions in termini della seguente formula CTL

$$\mathbf{A}((\mathbf{EF}\ g)\ \ \mathbf{W}\ \ g)$$

cioè in ogni cammino da tutti gli stati è sempre possibile raggiungere il goal g e se il cammino termina allora si raggiunge g Nota: $\mathbf{A}(g_1 \mathbf{W} g_2)$ significa che in ogni cammino o g_1 vale in tutti gli stati oppure $\mathbf{A}(g_1 \mathbf{U} g_2)$ ($\mathbf{W}$=weak until).

16.4 Symbolic Model Checking e Planning

Sulla base delle connessioni tra problemi di planning, problemi di raggiungibilità e formule CTL, sembra naturale chiedersi se sia possibile utilizzare algoritmi efficienti basati su OBDDs per calcolare soluzioni a problemi di planning di grandi dimensioni. Per fare ciò occorre codificare un problema di planning in modo simbolico esattamente come abbiamo fatto per problemi di model checking per formule CTL e modelli di Kripke.

Rappresentazione Simbolica di Domini di Planning Sia $\langle P, S, A, R\rangle$ un dominio di planning con $P = \{p_1, \ldots, p_k\}$. Gli stati vengono codificati come fomule proposizionali (definite sulle variabili proposizionali $\mathbf{x} = \langle x_1, \ldots, x_k\rangle$) $l_1 \wedge \ldots \wedge l_k$ dove l_i è il letterale x_i se p_i è vero in s e l_i è il letterale $\neg x_i$ se l_i è falso in s. Le azioni vengono rappresentate con formule proposizionali tramite un'ulteriore insieme di variabili proposizionali α (numeriamo le azioni e poi le codifichiamo in binario) La relazione di tranzione R si ottiene introducendo l'insieme di variabili proposizionali $\mathbf{x}' = \langle x'_1, \ldots, x'_n\rangle$.

- Una tripla $\langle s, a, s'\rangle \in R$ si definisce tramite una formule del tipo:

  $$F_a \ = \ l_1 \wedge \ldots \wedge l_k \wedge a_1 \wedge \ldots \wedge a_n \wedge l'_1 \wedge \ldots \wedge l'_k$$

  dove $l_1 \wedge \ldots \wedge l_k$ codifica s, $a_1 \wedge \ldots \wedge a_n$ codifica a e $l'_1 \wedge \ldots \wedge l'_k$ codifica s'.

- Eventuali vincoli di mutua esclusione (per evitare conflitti o forzare la sequenzialità delle azioni) si esprimono come formule disgiuntive S_a su $\alpha = \langle \alpha_1, \ldots, \alpha_n\rangle$

- La relazione R è quindi rappresentata dalla disgiunzione $R(\mathbf{x}, \alpha, \mathbf{x}') \ = \ \bigvee_{a \in A} F_a \wedge S_a$

Una state-table action è una relazione tra stati e azioni ed è quindi facilmente descrivibile tramite formule booleane sulle variabili α e $\mathbf{x}$. Chiamiamo $SA(\mathbf{x}, \alpha)$ la formula associata alla state-action table SA. Le operazioni si definiscono come segue

- l'insieme di stati associati ad una state-action table è rappresentato dalla formula $\exists\alpha.SA(\mathbf{x}, \alpha)$.
- $WeakPreImage(Q)$ è rappresentato dalla formula

$$\exists\mathbf{x}'.(R(\mathbf{x}, \alpha, \mathbf{x}') \wedge Q(\mathbf{x}')$$

- $StrongPreImage(Q)$ è rappresentato dalla formula

$$\forall\mathbf{x}'.(R(\mathbf{x}, \alpha, \mathbf{x}') \supset Q(\mathbf{x}')) \wedge \exists\mathbf{x}'.R(\mathbf{x}, \alpha, \mathbf{x}')$$

- $PruneState(SA, Q)$ è rappresentate dalla formula $SA(\mathbf{x}, \alpha) \wedge \neg Q(\mathbf{x})$.

Le operazioni insiemistiche (unione, intersezione, test di contenimento, ecc) si definiscono tramite operatori booleani in modo standard. Sulla base di tale encoding possiamo formulare gli algoritmi di weak and strong planning in domini non deterministici in modo simbolico. Utilizzando le tecniche di *symbolic model checking* basate su OBDDs (viste nel corso) possiamo infine utilizzare strutture dati efficienti per rappresentare domini e piani. La connessione tra model checking e planning rappresenta quindi un'ulteriore campo applicativo per logica modale e per le corrispondenti procedure di decisione.

Riferimenti bibliografici

[1] P. A. Abdulla, K. Cerāns, B. Jonsson and Y.-K. Tsay. General Decidability Theorems for Infinite-State Systems. In *Proc. LICS '99*, pp. 313-321, 1996.

[2] P. A. Abdulla and A. Nylén. Better is Better than Well: On Efficient Verification of Infinite-State Systems. In *Proc. LICS '00*, pp. 132-140, 2000.

[3] J. F. Allen. Towards a general theory of action and time. Artificial Intelligence, volume 23, pages 123-154, 1984.

[4] L. Aqvist. Deontic Logic. In D. Gabbay and F. Guenthner, ed., Handbook of Philosophical Logic: Volume II Extensions of Classical Logic. Kluwer. 1994.

[5] R. Bull and K. Segerberg. Basic Modal Logic. In Gabbay, D., and Guenthner, F. (eds.) Handbook of Philosophical Logic, Dordrecht: D. Reidel (1984): 2.1.

[6] T. Bultan, R. Gerber, and W. Pugh. Model-checking concurrent systems with unbounded integer variables: Symbolic representations, approximations, and experimental results. *ACM TOPLAS*, 21(4):747–789, 1999.

[7] M. Burrows, M. Abadi, R. M. Needham. A Logic of Authentication. TOCS 8(1): 18-36 (1990).

[8] L. Cardelli, A. D. Gordon. Anytime, Anywhere: Modal Logics for Mobile Ambients. In Proc. POPL 2000: 365-377, 2000.

[9] R. Carnap. Meaning and Necessity, Chicago: U. Chicago Press, 1947.

[10] B. Chellas. Modal Logic: An Introduction. Cambridge, England: Cambridge University Press (1980).

[11] A. Cimatti, M. Roveri, P. Traverso. Strong Planning in Non-Deterministic Domains Via Model Checking. AIPS 1998: 36-43, 1998.

[12] E. M. Clarke, E. A. Emerson, A. P. Sistla. Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach. In Proc. POPL 1983: pag. 117-126, 1983.

[13] G. Delzanno. Automatic Verification of Parameterized Cache Coherence Protocols. In *Proc. CAV '00*, LNCS 1855, pp. 53-68, 1996.

[14] G. Delzanno, J. Esparza, and A. Podelski. Constraint-based Analysis of Broadcast Protocols. In *Proc. CSL'9)*, LNCS 1683, pp. 50-66, 1999.

[15] E. A. Emerson, E. M. Clarke. Characterizing Correctness Properties of Parallel Programs Using Fixpoints. In Proc. ICALP 1980: pag 169-181, 1980.

[16] E. Allen Emerson, Edmund M. Clarke: Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons. Science of Computer Programming 2(3): 241-266 (1982).

[17] M. Fisher and M. Wooldridge. Executable Temporal Logic for Distributed AI. In Proc. of the Twelfth International Workshop on Distributed Artificial Intelligence (IWDAI-93), 1993.

[18] D. Gabbay Temporal Logic: Mathematical Foundations and Computational Aspects. New York: Oxford University Press (1994).

[19] D. Gabbay Investigations in Modal and Tense Logics, Dordrecht: D. Reidel (1976).

[20] Girard, J.-Y. (1987). Linear logic. *Theoretical computer science*, **50:1**, 1–102.

[21] N. Halbwachs, Y-E. Proy, and P. Roumanoff. Verification of Real-time Systems using Linear Relation Analysis. *Formal Methods in System Design*, 11(2):157–185, 1997.

[22] J. Y. Halpern and Y. Moses. A Guide to the Modal Logics of Knowledge and Belief: Preliminary Draft. In Proc. IJCAI 1985: 480-490, 1985.

[23] D. Harel. First Order Dynamic Logic. Lecture Notes in Computer Science, vopl 68, 1979

[24] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: a Model Checker for Hybrid Systems. In *Proc. CAV'97*, LNCS 1254, pp. 460–463, 1997.

[25] R. Hilpinen. Deontic Logic: Introductory and Systematic Readings. Dordrecht: D. Reidel (1971).

[26] J. Hintikka Knowledge and Beliefs. Cornell University Press, 1962.

[27] G. Hughes and M. Cresswell, M., A New Introduction to Modal Logic. London: Routledge (1996).

[28] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, E. Shahar. Symbolic Model Checking with Rich Assertional Languages. In *Proc. CAV '97*, pp. 424-435, 1997.

[29] S. Kripke. Semantical Considerations on Modal Logic. Acta Philosophica Fennica, 16, (1963): 83-94

[30] K. G. Larsen, P. Pettersson, and W. Yi. uppaal in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[31] E. Lemmon and D. Scott. An Introduction to Modal Logic, Oxford: Blackwell (1977).

[32] O. Lichtenstein, A. Pnueli. Checking That Finite State Concurrent Programs Satisfy Their Linear Specification. In Proc. POPL 1985: pag. 97-107, 1985.

[33] P. Martin-Löf. An Intuitionsistic Theory of Types: Predicative part. Logic colloquium 1973. North Holland. 1973.

[34] Z. Manna and A. Pnueli. The Modal Logic of Programs. Proc. of Int'l Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science, vol 71, pag 395-411, 1979.

[35] R. McDermott. A Temporal Logic for Reasoning about actions and plans. KCognitive Science 6, 101-155, 1982.

[36] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* Kluwer Academic, 1993.

[37] D. Miller. A Logic Programming Language with Lambda-Abstraction, Function Variables, and Simple Unification. Journal of Logic and Computation, Vol. 1, No. 4 (1991), 497 – 536.

[38] R. C. Moore. A Formal Theory of Knowledge and Action. Formal theories of the common sense world. Ablex Pub. Corp., 1984.

[39] The Omega project, http://www.cs.umd.edu/projects/omega/.

[40] A. Podelski. Model Checking as Constraint Solving In *Proc. SAS 2001*, LNCS 1824, pp. 22–37, 2001.

[41] A. Pnueli. The temporal logic of programs. In Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, pages 46-67.

[42] A. N. Prior. Past, Present and Future. Oxford: Clarendon Press, 1967.

[43] J. F. van Benthem. The Logic of Time, Dordrecht: D. Reidel (1982).

[44] J. F. van Benthem. Temporal Logic, in D. M. Gabbay, C. J. Hogger, and J. A. Robinson, Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 4, Oxford: Clarendon Press, pages 241-350.

[45] M. Wooldridge. Temporal Belief Logics for Modelling Distributed AI Systems. In Foundations of DAI. Wiley Interscience, 1995.

[46] J. J. Zeman Modal Logic. Oxford Pub., 1973.