μ-calculus Model Checking

Rita D'Ambrosio

Università degli Studi di Salerno

Corso di Tecniche Automatiche per la Correttezza del Software

December 13, 2016

Overview

1. Introduzione
2. Modal Logic
   - Hennesy-Milner logic
   - Fixed Point Logic
   - HML con ricorsione (1HML)
   - μ-calculus
3. Model Checking
   - Parity Games e μ-calculus
   - Logiche temporali
   - Conclusioni

Introduzione

Il μ-calculus è una logica ampiamente utilizzata sia in campo informatico che matematico.

Introduzione

Il μ-calculus è una logica ampiamente utilizzata sia in campo informatico che matematico.
La sua caratteristica principale è l'aggiunta di ricorsione alla logica modale, ottenendo così un grande aumento di potenza espressiva ma anche un aumento in termini di difficoltà di comprensione.

Introduzione

Il μ-calculus è una logica ampiamente utilizzata sia in campo informatico che matematico.
La sua caratteristica principale è l'aggiunta di ricorsione alla logica modale, ottenendo così un grande aumento di potenza espressiva ma anche un aumento in termini di difficoltà di comprensione.

La logica, introdotta da Dexter Kozen nel 1983 (originariamente da D. Scott e J. de Bakker),è ampiamente utilizzata per la specifica formale e la verifica di sistemi, venendo impiegata per il problema del Model Checking.

Modal Logic nella verifica dei programmi

La logica modale è un tipo di logica formale sviluppata nel 1960 che estende la logica dei predicati e delle proposizioni includendo operatori che *esprimono modalità*.

Modal Logic nella verifica dei programmi

La logica modale è un tipo di logica formale sviluppata nel 1960 che estende la logica dei predicati e delle proposizioni includendo operatori che *esprimono modalità*.

Jhon is happy.
Jhon is usally happy.

usally ha la funzione di operatore modale.
$\Box, \Diamond$

Modal Logic nella verifica dei programmi

Negli anni succesivi, si è cercato di ampliare la Modal Logic con la Dynamic logic.

- Modal logic
  - $\Box$ p : Necessariamente p.

Modal Logic nella verifica dei programmi

Negli anni succesivi, si è cercato di ampliare la Modal Logic con la Dynamic logic.

- Modal logic
  - $\Box$ p : Necessariamente p.

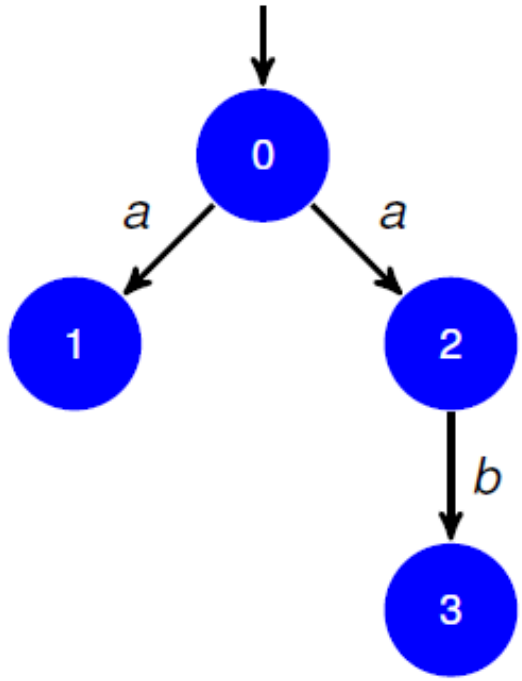
La Dynamic logic estende questo concetto associando ad ogni azione a l'operatore modale [a] creando una multimodal logic.

- Dynamic logic
  - $[a]p$: dopo l'esecuzione dell'azione a, necessariamente p

Labelled Transition System

Definiamo una LST come una tupla:
$M : (S, s_0, Act, \delta)$

- S è un insieme finito
- s_0 è l'insieme di partenza
- Act è un insieme finito di azioni
- $\delta \subseteq (SxAxS)$



Hennesy-Milner logic

Sintassi HML

Una formula HML è data da:

- *true* e *false*
- $\varphi \wedge \psi; \varphi \vee \psi; \neg\varphi$
- $[a]\varphi; \quad \langle a \rangle \varphi; \quad (a \in A)$

Una formula HML è data da:

- *true* e *false*
- $\varphi \wedge \psi; \varphi \vee \psi; \neg\varphi$
- $[a]\varphi; \quad \langle a\rangle\varphi; \quad (a \in A)$

Quindi dato M un LTS,

$$M, v \models \langle a\rangle\varphi$$

sse $\exists w \quad t.c \quad v \xrightarrow{a} w$, abbiamo $M, w \models \varphi$

Ad ogni formula associamo un insieme di stati in cui è valida
$||\varphi|| \subseteq S$ è definita:

Ad ogni formula associamo un insieme di stati in cui è valida
$||\varphi|| \subseteq S$ è definita:

- $||\mathit{true}|| = S$
- $||\mathit{false}|| = \varnothing$
- $||\varphi \wedge \psi|| = ||\varphi|| \cup ||\psi||$
- $||\varphi \vee \psi|| = ||\varphi|| \cap ||\psi||$

Ad ogni formula associamo un insieme di stati in cui è valida
$||\varphi|| \subseteq S$ è definita:

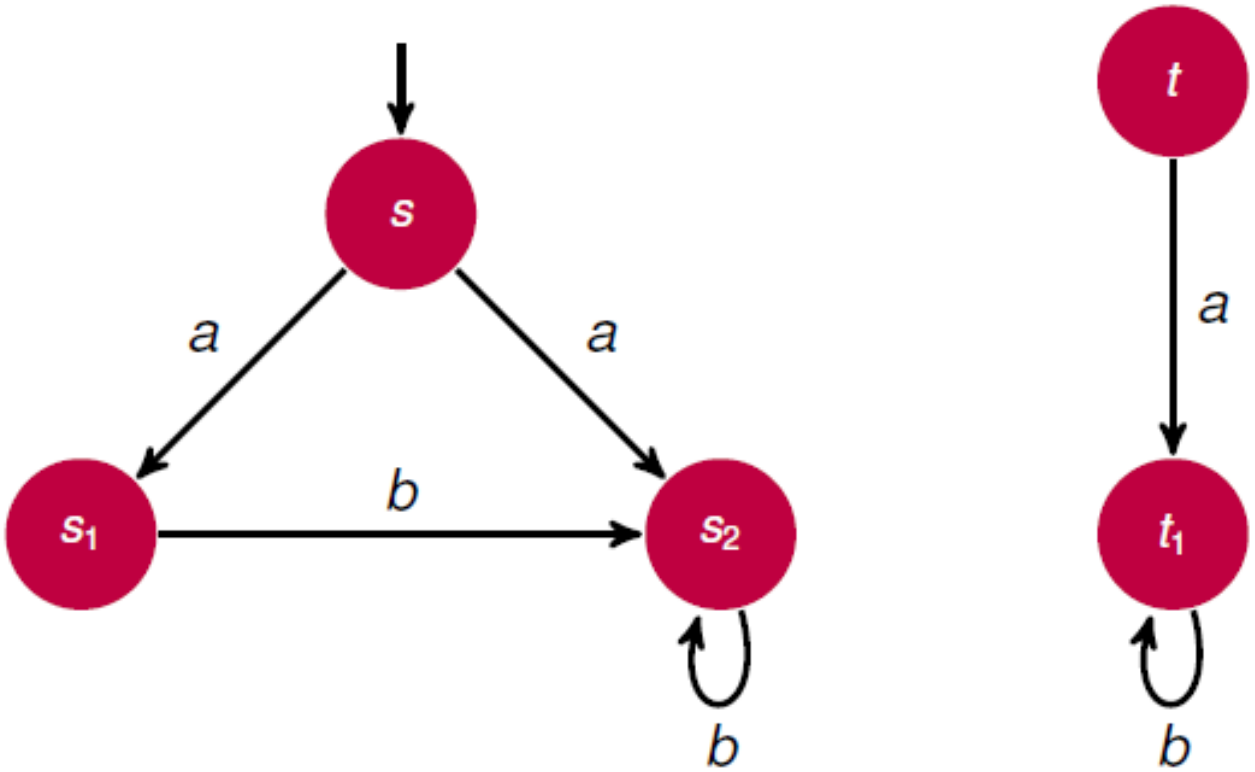
- $||true|| = S$
- $||false|| = \varnothing$
- $||\varphi \wedge \psi|| = ||\varphi|| \cup ||\psi||$
- $||\varphi \vee \psi|| = ||\varphi|| \cap ||\psi||$
- $||\langle a \rangle \varphi|| = \langle \cdot a \cdot \rangle ||\varphi||$
  - dove $\langle \cdot a \cdot \rangle T = \{p \in S | \exists p'.p \xrightarrow{a} p' \quad and \quad p' \in T\}$

Ad ogni formula associamo un insieme di stati in cui è valida
$||\varphi|| \subseteq S$ è definita:

- $||true|| = S$
- $||false|| = \varnothing$
- $||\varphi \wedge \psi|| = ||\varphi|| \cup ||\psi||$
- $||\varphi \vee \psi|| = ||\varphi|| \cap ||\psi||$
- $||\langle a \rangle \varphi|| = \langle \cdot a \cdot \rangle ||\varphi||$
  - dove $\langle \cdot a \cdot \rangle T = \{p \in S | \exists p'.p \xrightarrow{a} p' \quad and \quad p' \in T\}$
- $||[a]\varphi|| = [\cdot a \cdot]||\varphi||$
  - dove $[\cdot a \cdot] T = \{p \in S | \forall p'.p \xrightarrow{a} p' \implies p' \in T\}$

Hennesy-Milner logic

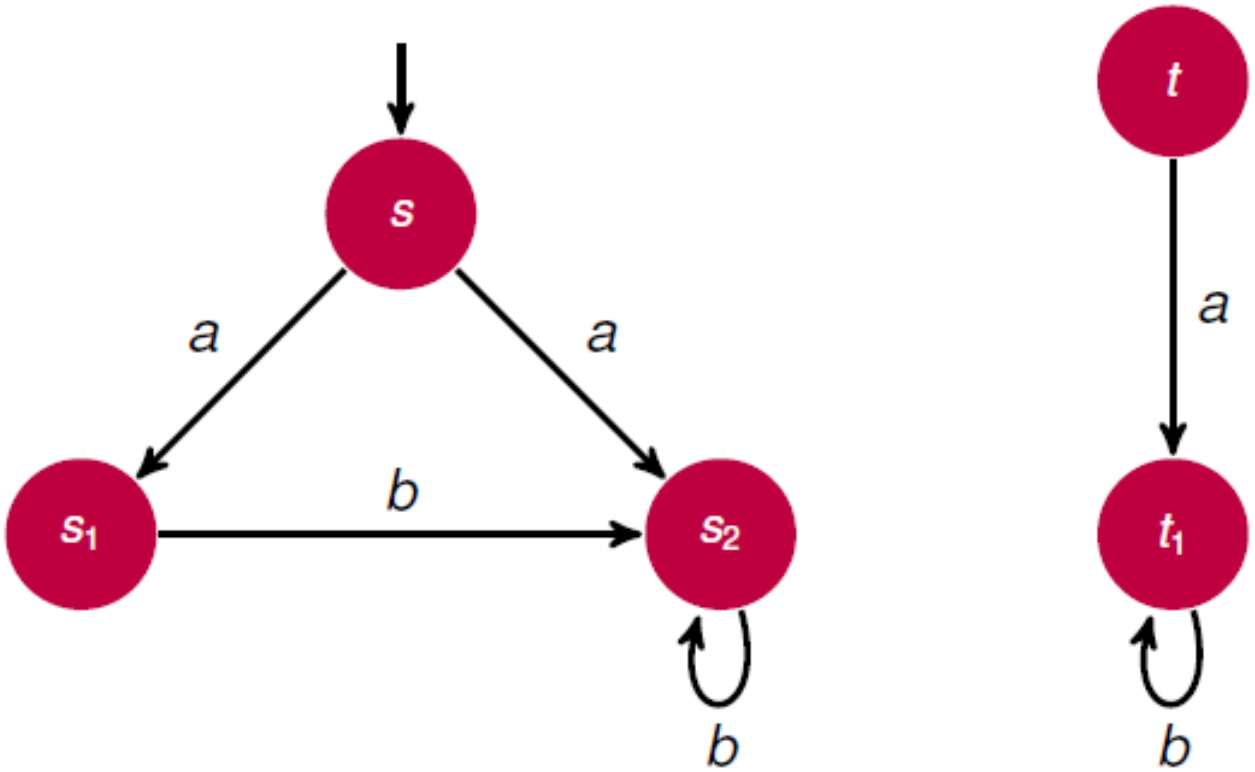
Esempi su LTS



- $\langle \cdot a \cdot \rangle \{s1, t1\} =$

Hennesy-Milner logic

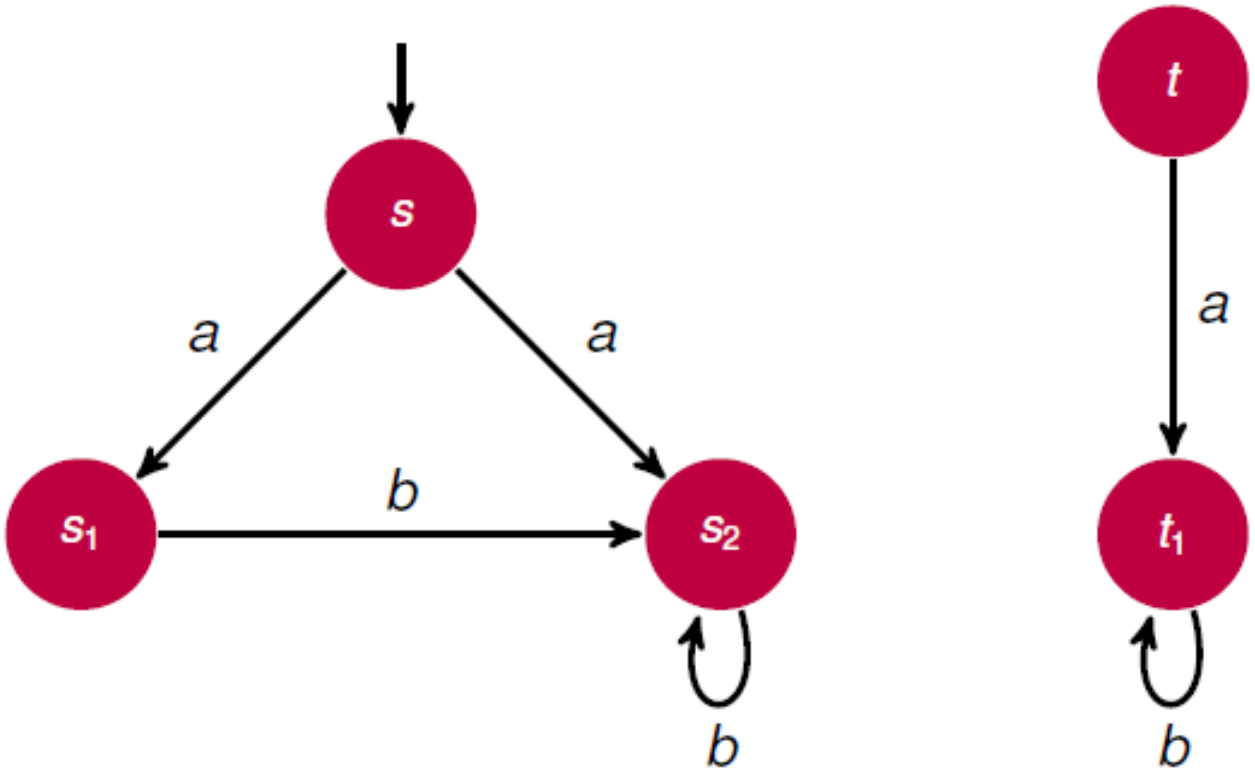
Esempi su LTS



- $\langle \cdot a \cdot \rangle \{s1, t1\} = \{s, t\}$

Hennesy-Milner logic

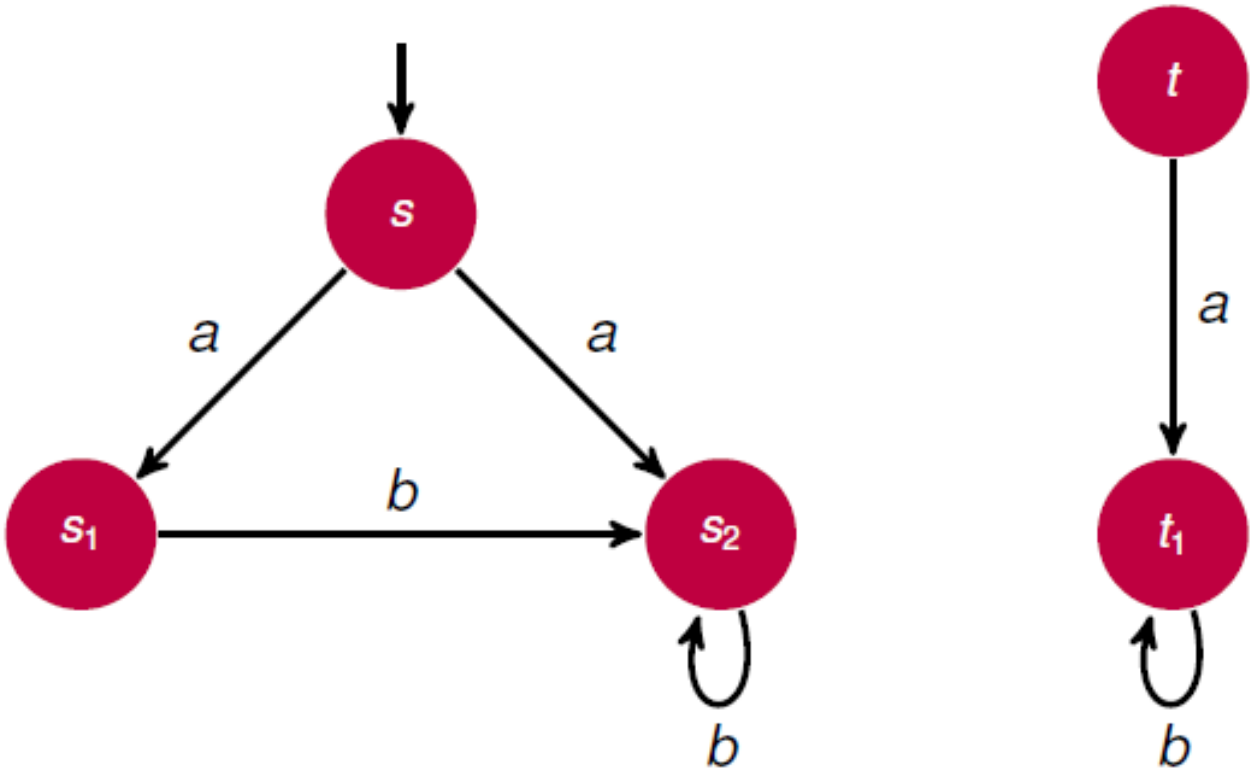
Esempi su LTS



- $\langle \cdot a \cdot \rangle \{s1, t1\} = \{s, t\}$
- $[\cdot a \cdot] \{s1, t1\} =$

Hennesy-Milner logic

Esempi su LTS

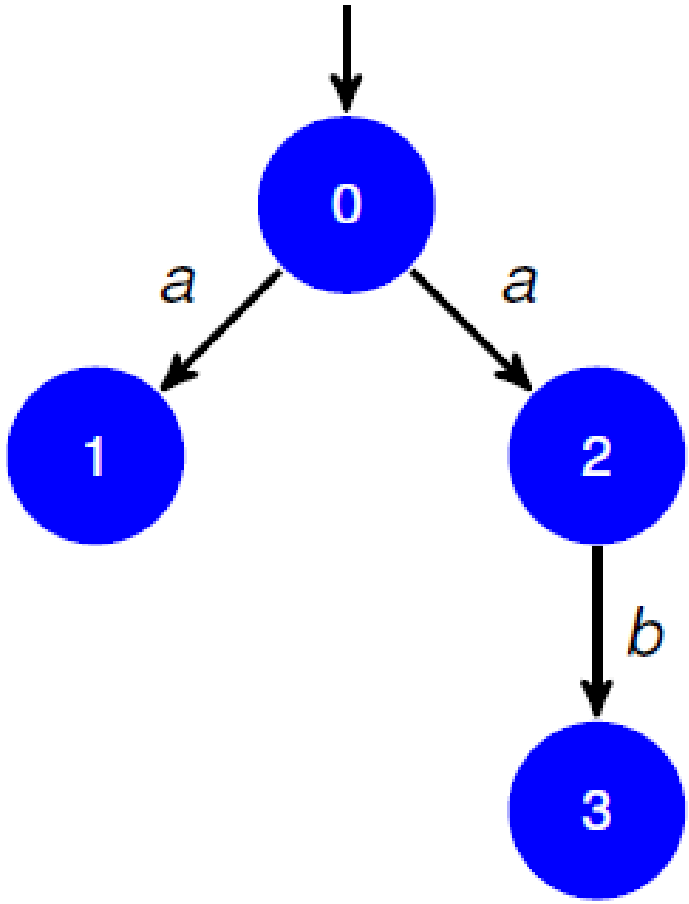


- $\langle \cdot a \cdot \rangle \{s1, t1\} = \{s, t\}$
- $[\cdot a \cdot] \{s1, t1\} = \{s1, s2, t, t1\}$

Hennesy-Milner logic

Esempi su LTS

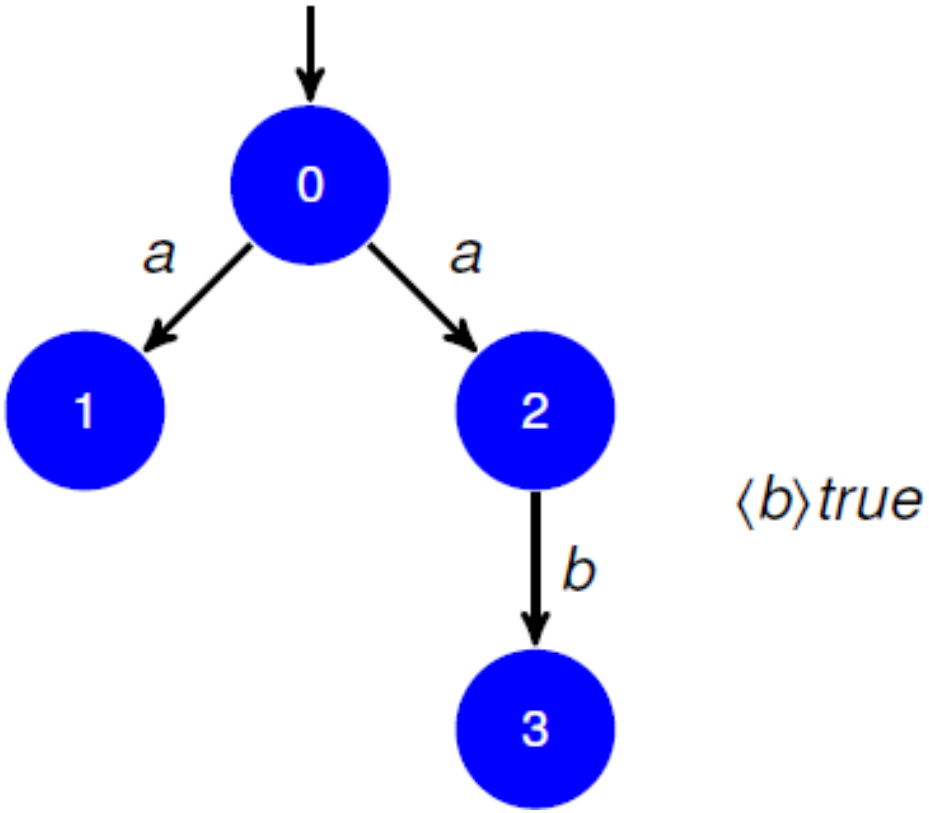
- Possibile effettuare una a-transizione ed è possibile fare una b-transizione
  - $\langle a \rangle \langle b \rangle true$



Hennesy-Milner logic

Esempi su LTS

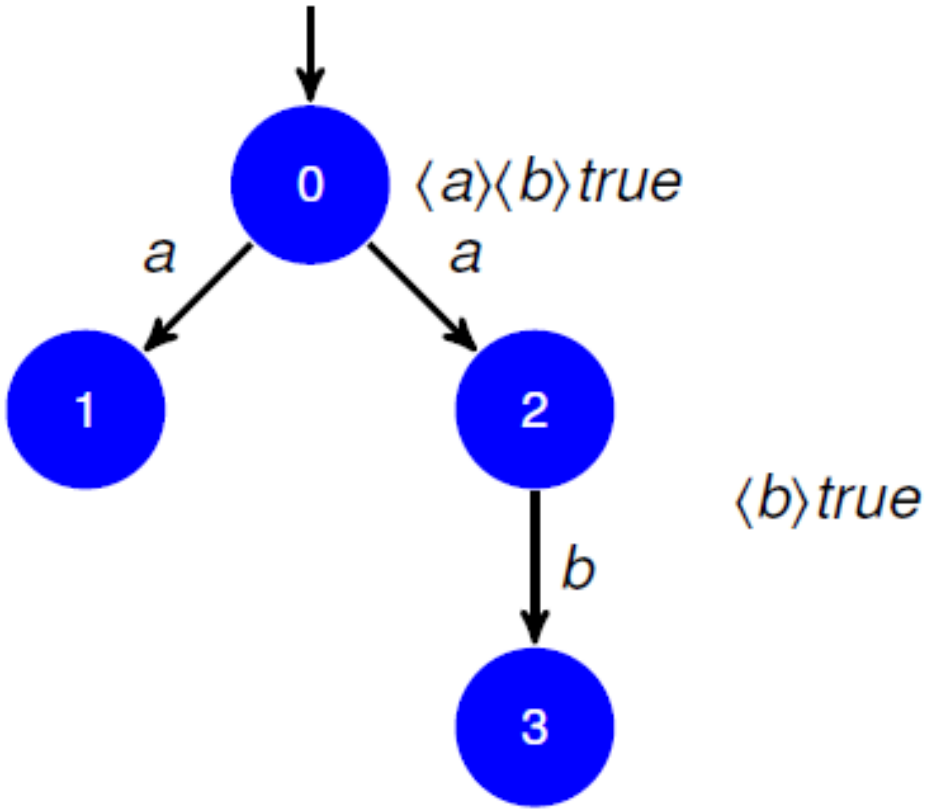
- Possibile effettuare una a-transizione ed è possibile fare una b-transizione
  - $\langle a \rangle \langle b \rangle true$



Hennesy-Milner logic

Esempi su LTS

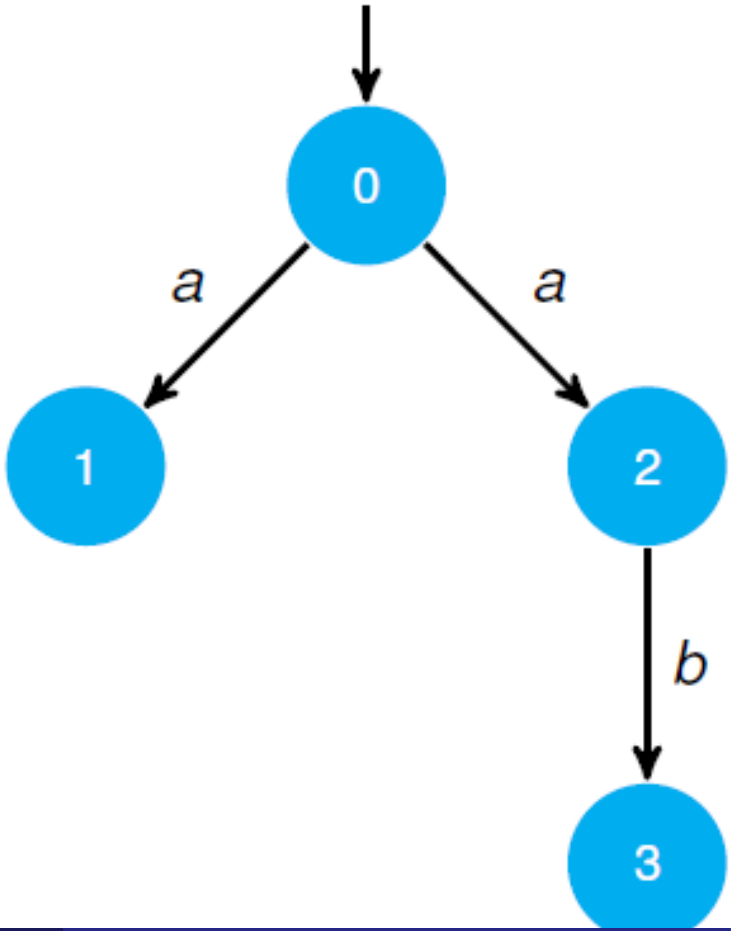
- Possibile effettuare una a-transizione ed è possibile fare una b-transizione
  - $\langle a \rangle \langle b \rangle true$



Hennesy-Milner logic

Esempi su LTS

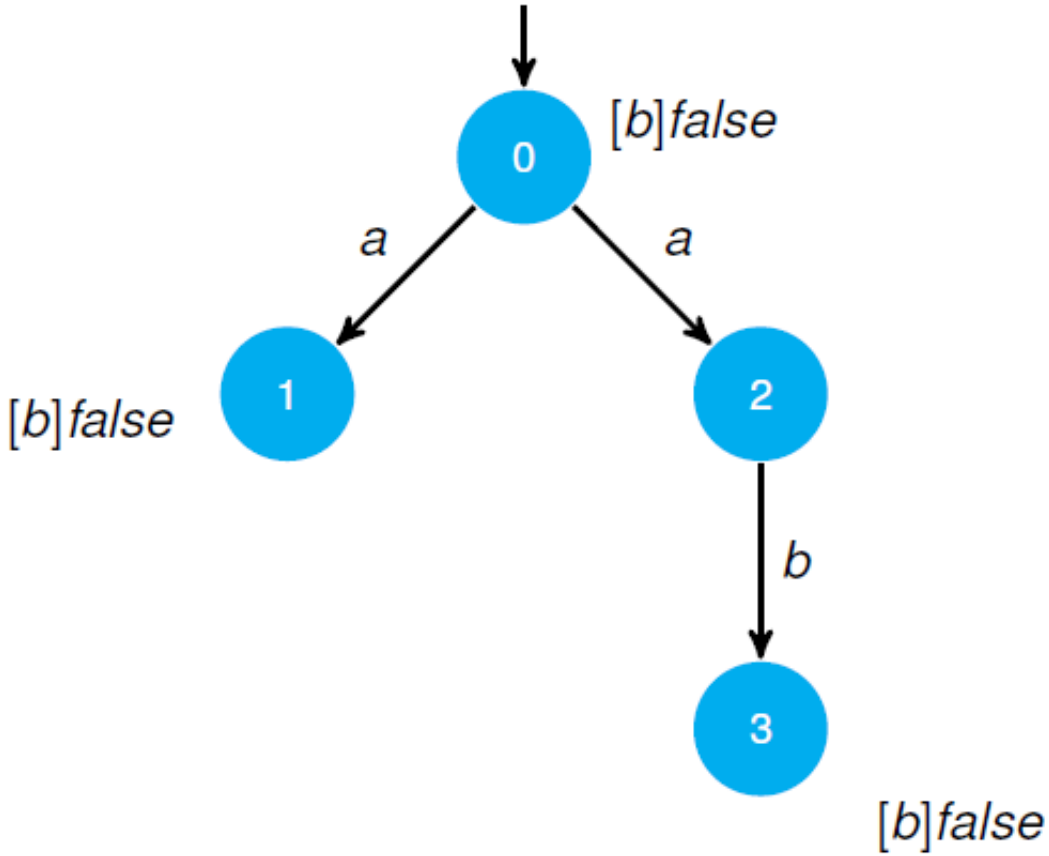
- Possibile effettuare una a-transizione ed impossibile fare un b-transizione
  - $\langle a \rangle [b] false$



Hennesy-Milner logic

Esempi su LTS

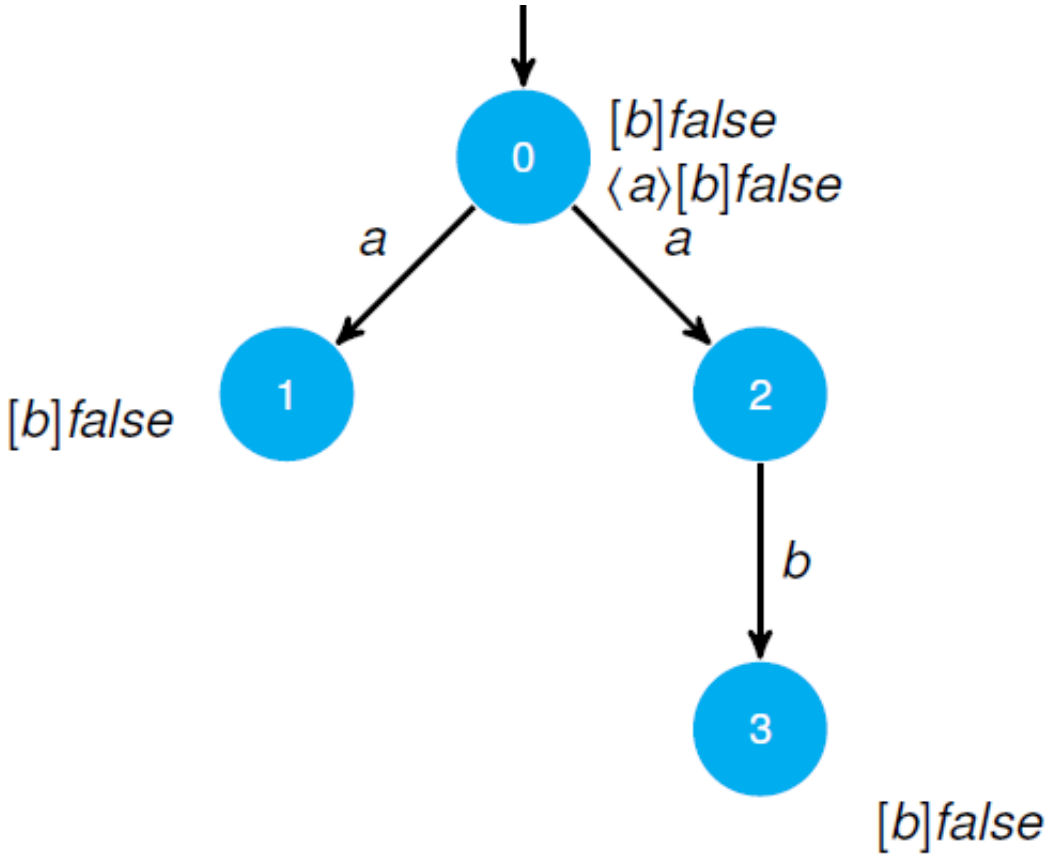
- Possibile effettuare una a-transizione ed impossibile fare un b-transizione
  - $\langle a \rangle [b] false$



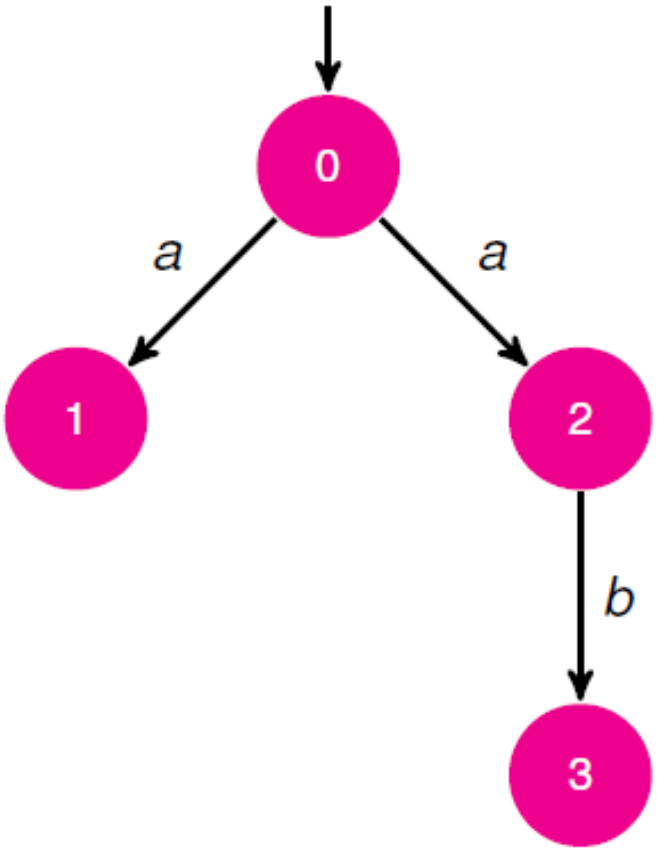
Hennesy-Milner logic

Esempi su LTS

- Possibile effettuare una a-transizione ed impossibile fare un b-transizione
  - $\langle a \rangle [b] false$



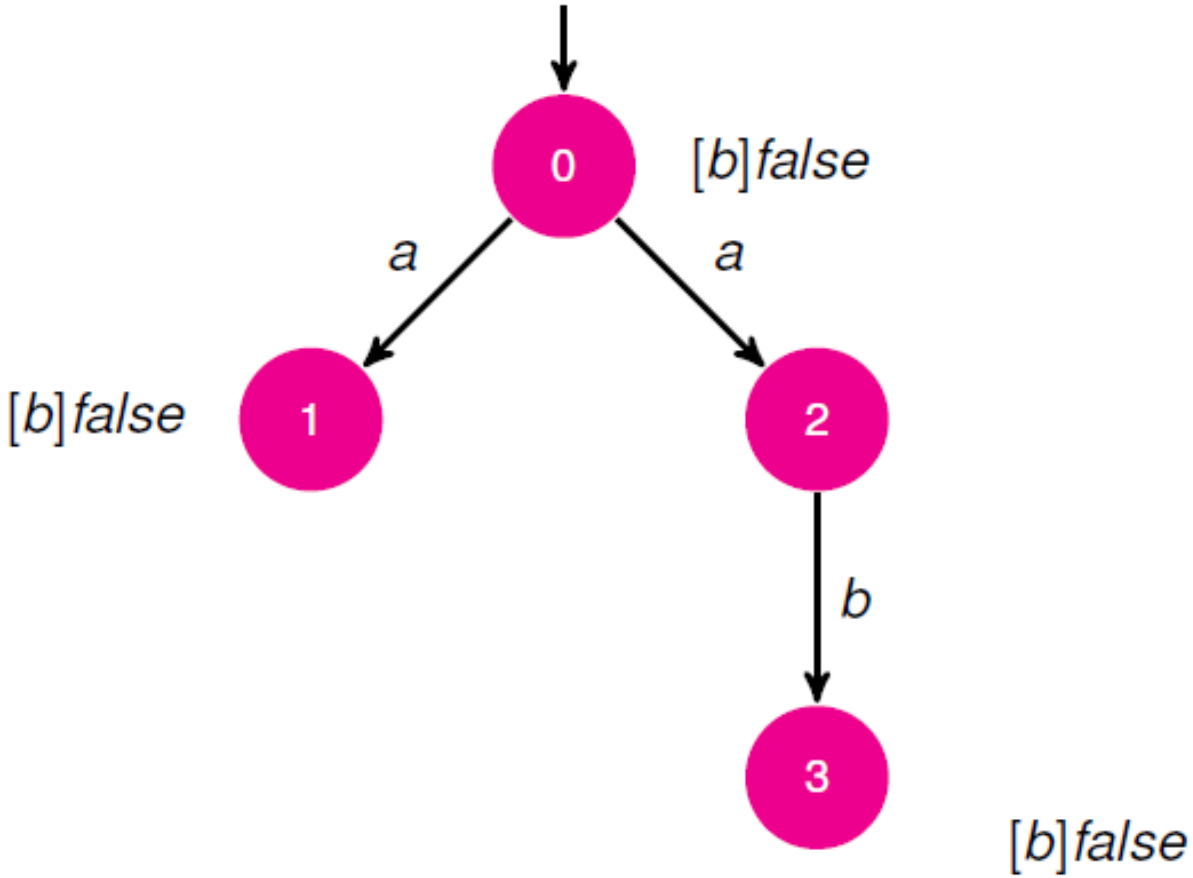
- Non possiamo mai effettuare una a-transizione seguita da una b-transizione
  - $[a][b]false$



Hennesy-Milner logic

Esempi su LTS

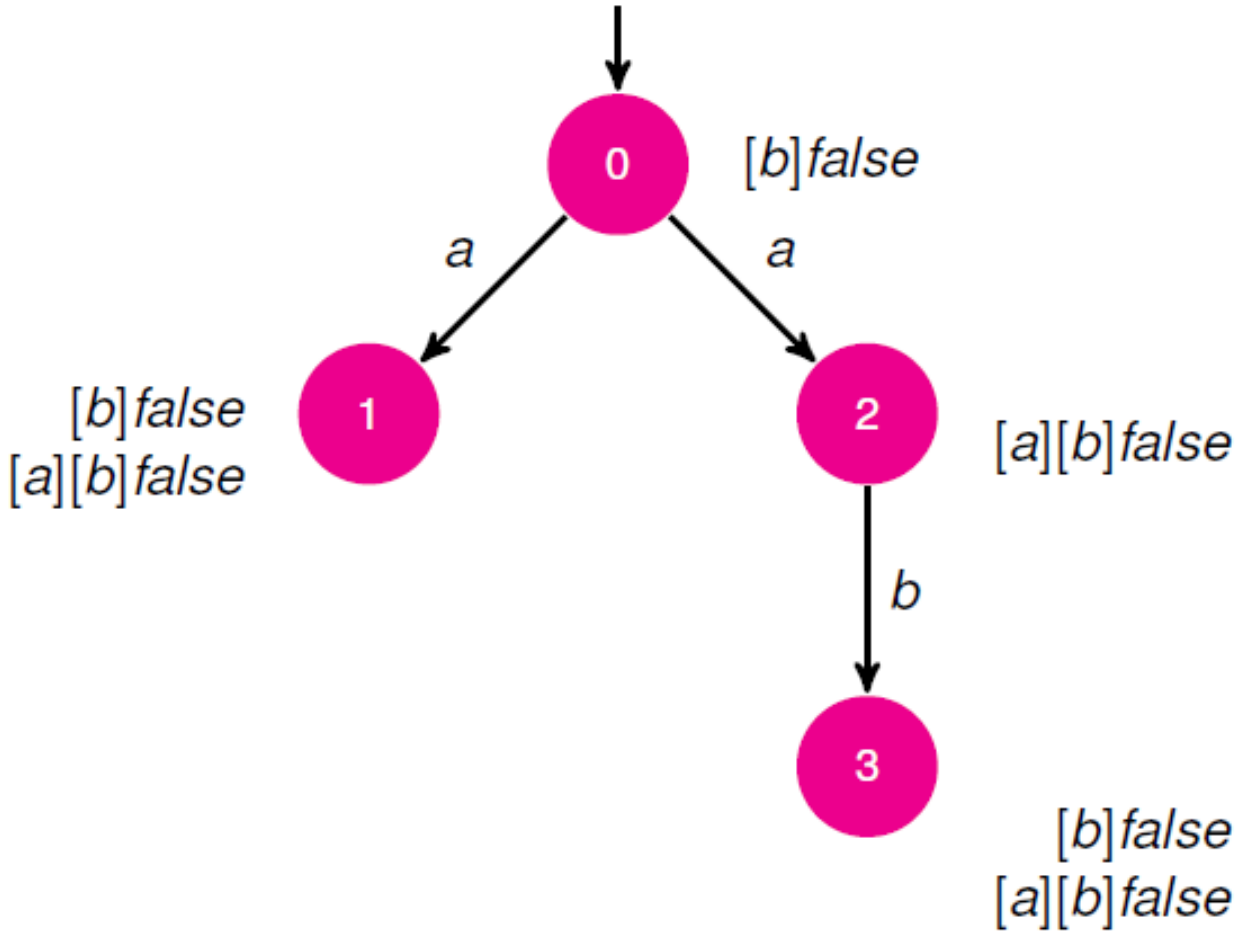
- Non possiamo mai effettuare una a-transizione seguita da una b-transizione
  - $[a][b]false$



Hennesy-Milner logic

Esempi su LTS

- Non possiamo mai effettuare una a-transizione seguita da una b-transizione
  - $[a][b]false$



Hennesy-Milner logic

Esempi su proprietà

- Jhon non vuole bere caffè adesso

Hennesy-Milner logic

Esempi su proprietà

- Jhon non vuole bere caffè adesso
  - $\neg\langle caffe\rangle true$ oppure $[caffe]false$

Hennesy-Milner logic

Esempi su proprietà

- Jhon non vuole bere caffè adesso
  - $\neg\langle caffe\rangle true$ oppure $[caffe]false$
- Jhon non beve mai birra

Hennesy-Milner logic

Esempi su proprietà

- Jhon non vuole bere caffè adesso
  - $\neg\langle caffe\rangle true$ oppure $[caffe]false$
- Jhon non beve mai birra
  - ?

Hennesy-Milner logic

Esempi su proprietà

- Jhon non vuole bere caffè adesso
  - $\neg\langle caffe\rangle true$ oppure $[caffe]false$
- Jhon non beve mai birra
  - ?
- Jhon produrrà sempre una pubblicazione immediatamente dopo aver bevuto due caffè di fila

Hennesy-Milner logic

Esempi su proprietà

- Jhon non vuole bere caffè adesso
  - $\neg\langle caffe\rangle true$ oppure $[caffe]false$
- Jhon non beve mai birra
  - ?
- Jhon produrrà sempre una pubblicazione immediatamente dopo aver bevuto due caffè di fila
  - $[caffe][caffe](\langle pub\rangle true \wedge [Act \setminus \{pub\}]false)$

Hennesy-Milner logic

Esempi su proprietà

- Jhon non vuole bere caffè adesso
  - $\neg\langle caffe\rangle true$ oppure $[caffe]false$
- Jhon non beve mai birra
  - ?
- Jhon produrrà sempre una pubblicazione immediatamente dopo aver bevuto due caffè di fila
  - $[caffe][caffe](\langle pub\rangle true \wedge [Act \setminus \{pub\}]false)$
- Jhon produce sempre una pubblicazione dopo aver bevuto caffè

Hennesy-Milner logic

Esempi su proprietà

- Jhon non vuole bere caffè adesso
  - $\neg\langle caffe\rangle true$ oppure $[caffe]false$
- Jhon non beve mai birra
  - ?
- Jhon produrrà sempre una pubblicazione immediatamente dopo aver bevuto due caffè di fila
  - $[caffe][caffe](\langle pub\rangle true \wedge [Act \setminus \{pub\}]false)$
- Jhon produce sempre una pubblicazione dopo aver bevuto caffè
  - ?

Hennesy-Milner logic

- Il potere espressivo di HML in questa forma è debole: dato una formula HML possiamo creare statements su numero *finito* di passi nel futuro. HML è stato introdotto non come un linguaggio per esprimere proprietà, ma per comprendere se due processi sono equivalenti:

due processi sono equivalenti sse soddisfanno la stessa HML formula.

Hennesy-Milner logic

- Il potere espressivo di HML in questa forma è debole: dato una formula HML possiamo creare statements su numero *finito* di passi nel futuro. HML è stato introdotto non come un linguaggio per esprimere proprietà, ma per comprendere se due processi sono equivalenti:

due processi sono equivalenti sse soddisfanno la stessa HML formula.

- Per ottenere più potenza espressiva abbiamo bisogno di una logica più forte e che ci permetta di esprimere proprietà durevoli nel tempo.

Hennesy-Milner logic

- Il potere espressivo di HML in questa forma è debole: dato una formula HML possiamo creare statements su numero *finito* di passi nel futuro. HML è stato introdotto non come un linguaggio per esprimere proprietà, ma per comprendere se due processi sono equivalenti:

due processi sono equivalenti sse soddisfanno la stessa HML formula.

- Per ottenere più potenza espressiva abbiamo bisogno di una logica più forte e che ci permetta di esprimere proprietà durevoli nel tempo.

- Nel 1983, Dezter Kozen ha pubblicato uno studio che combina la logica HM con gli operatori fixed point per fornire una forma di ricorsione alla logica.

Fixed Point Logic

- La caratteristica principale del μ-caluclus è l'uso degli operatori fixed point.

Fixed Point Logic

- La caratteristica principale del μ-caluclus è l'uso degli operatori fixed point.
- Permette le definizione induttive, come un descrittore di funzioni ricorsive primitive.

Fixed Point Logic

- La caratteristica principale del μ-caluclus è l'uso degli operatori fixed point.
- Permette le definizione induttive, come un descrittore di funzioni ricorsive primitive.
- E' usato per descrivere proprietà di transizioni di sistemi e per verificare queste proprietà

Fixed Point Logics

- Sia $\mathfrak{S}$ l'insieme degli stati di un sistema.

Fixed Point Logics

- Sia $\mathfrak{S}$ l'insieme degli stati di un sistema.
- Definiamo $\wp\mathfrak{S} = \{s \in \mathfrak{S} | s \models \phi\}$.

Fixed Point Logics

- Sia $\mathfrak{S}$ l'insieme degli stati di un sistema.
- Definiamo $\wp\mathfrak{S} = \{s \in \mathfrak{S} | s \models \phi\}$.
- Se noi permettiamo alla logica di contenere variabili con interpretazione che vanno su $\wp\mathfrak{S}$ allora possiamo vedere la semantica di una formula con una variabile libera $\varphi(X)$ come una funzione $f : \wp\mathfrak{S} \Rightarrow \wp\mathfrak{S}$.

Fixed Point Logics

- Sia $\mathfrak{S}$ l'insieme degli stati di un sistema.
- Definiamo $\wp\mathfrak{S} = \{s \in \mathfrak{S} | s \models \phi\}$.
- Se noi permettiamo alla logica di contenere variabili con interpretazione che vanno su $\wp\mathfrak{S}$ allora possiamo vedere la semantica di una formula con una variabile libera $\varphi(X)$ come una funzione $f : \wp\mathfrak{S} \Rightarrow \wp\mathfrak{S}$.
- Se $f(S) \subseteq f(S')$ con $S \subseteq S' \subseteq \mathfrak{S}$ allora f è monotona.

Fixed Point Logics

- Sia $\mathfrak{S}$ l'insieme degli stati di un sistema.
- Definiamo $\wp\mathfrak{S} = \{s \in \mathfrak{S} | s \models \phi\}$.
- Se noi permettiamo alla logica di contenere variabili con interpretazione che vanno su $\wp\mathfrak{S}$ allora possiamo vedere la semantica di una formula con una variabile libera $\varphi(X)$ come una funzione $f : \wp\mathfrak{S} \Rightarrow \wp\mathfrak{S}$.
- Se $f(S) \subseteq f(S')$ con $S \subseteq S' \subseteq \mathfrak{S}$ allora f è monotona.
- Se $f(S) = S$, S è un punto fisso di f.

Fixed Point Logics

Se vediamo $(\wp\mathfrak{S}, \subseteq)$ come un reticolo completo, e f è monotona, allora:

Theorem (Knaster–Tarski)

Sia $f : L \Rightarrow L$ una funzione monotona su un reticolo completo L.
L'insieme dei punti fissi di f è anche'esso un reticolo completo.

Fixed Point Logics

Se vediamo $(\wp\mathfrak{S}, \subseteq)$ come un reticolo completo, e f è monotona, allora:

Theorem (Knaster–Tarski)

Sia $f : L \Rightarrow L$ una funzione monotona su un reticolo completo L. L'insieme dei punti fissi di f è anche'esso un reticolo completo.

- Quindi f ha un unico massimo e un unico minimo.

Fixed Point Logics

Se vediamo $(\wp\mathfrak{S}, \subseteq)$ come un reticolo completo, e f è monotona, allora:

Theorem (Knaster–Tarski)

Sia $f : L \Rightarrow L$ una funzione monotona su un reticolo completo L. L'insieme dei punti fissi di f è anche'esso un reticolo completo.

- Quindi f ha un unico massimo e un unico minimo.
- Il massimo fixed-point è l'unione dei post-fixed points
  - $\bigcup\{S \subset \mathfrak{S} | S \subseteq f(S)\}$

Fixed Point Logics

Se vediamo $(\wp\mathfrak{S}, \subseteq)$ come un reticolo completo, e f è monotona, allora:

Theorem (Knaster–Tarski)

Sia $f : L \Rightarrow L$ una funzione monotona su un reticolo completo L. L'insieme dei punti fissi di f è anche'esso un reticolo completo.

- Quindi f ha un unico massimo e un unico minimo.
- Il massimo fixed-point è l'unione dei post-fixed points
  - $\bigcup\{S \subset \mathfrak{S} | S \subseteq f(S)\}$
- Il minimo fixed-point è l'intersezione dei pre-fixed points
  - $\bigcap\{S \subset \mathfrak{S} | f(S) \subseteq S\}$

HML con ricorsione

Possiamo estendere HML con gli operatori fixed point per creare ricorsione.
Quindi presentiamo la logica di HM con un'unica variabile ricorsiva (1HML).
Massimo operatore fixed point μ minimo operatore fixed point e ν massimo operatore fixed point. $\mu X.\varphi(X) \quad , \quad \nu X.\varphi(X)$
Sintassi alternativa:
$X \stackrel{min}{=} Fx \quad X \stackrel{max}{=} Fx$

HML con ricorsione

Esempi

$X \overset{min}{=} X$

Qualsiasi insieme di stati S soddisfa X=X.

Il più piccolo insieme è $\emptyset$.

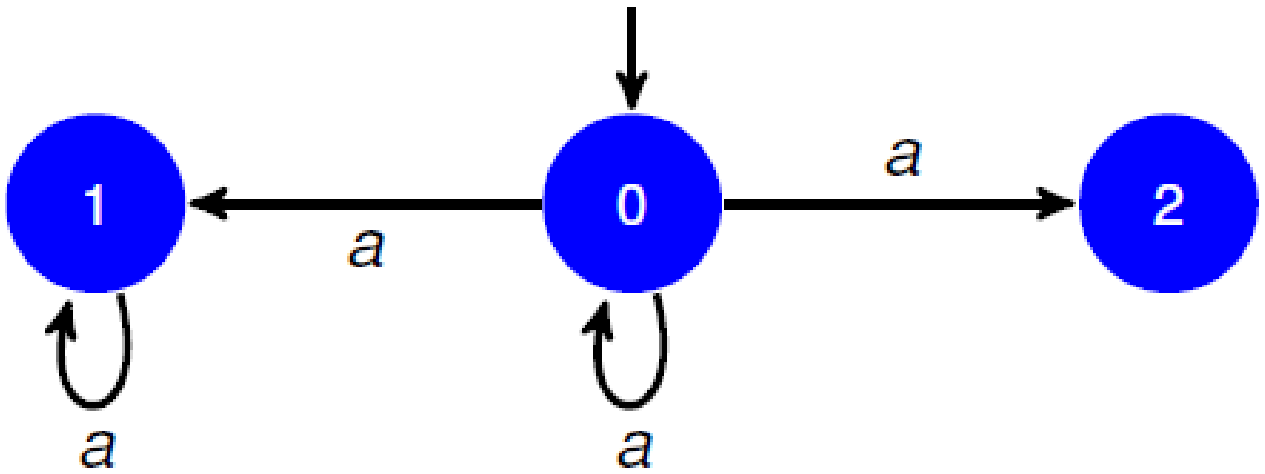
$X \overset{max}{=} X$

Qualsiasi insieme di stati S soddisfa X=X.

Il più grande insieme è S.

Eventualmente *a* sarà disabilitata:

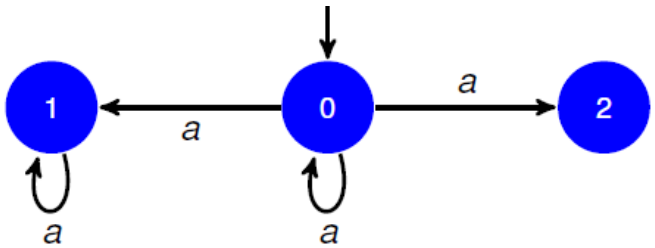
$$X \stackrel{?}{=} [a]false \vee \langle Act \rangle X$$



HML con ricorsione

Esempi su LTS

$X \stackrel{?}{=} [a]false \vee \langle Act \rangle X$

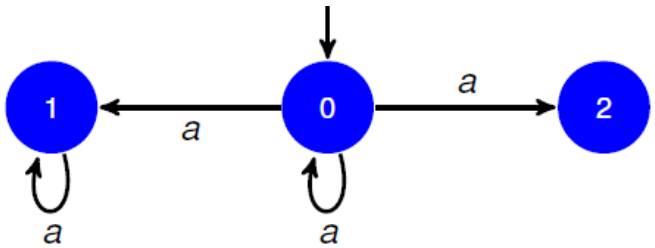


$[a]false \vee \langle Act \rangle \emptyset =$

HML con ricorsione

Esempi su LTS

$$X \stackrel{?}{=} [a]false \vee \langle Act \rangle X$$

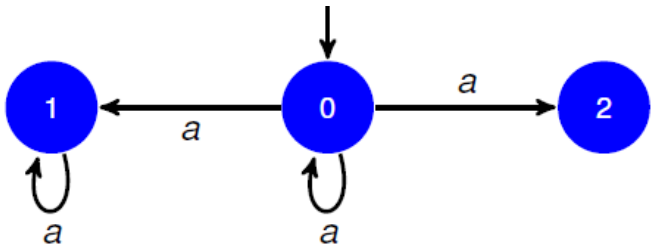


$$\overbrace{[a]false}^{\{2\}} \vee \overbrace{\langle Act \rangle \emptyset}^{\emptyset} = \{2\}$$

HML con ricorsione

Esempi su LTS

$$X \stackrel{?}{=} [a]false \vee \langle Act \rangle X$$

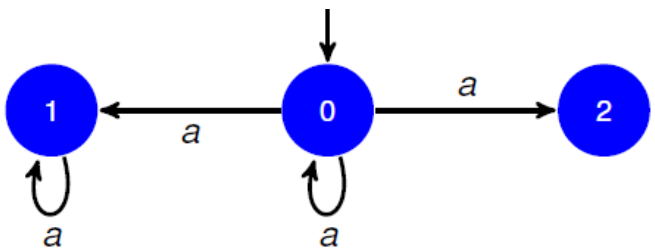


$$[a]false \vee \langle Act \rangle \emptyset = \{2\}$$
$$[a]false \vee \langle Act \rangle \{2\} =$$

HML con ricorsione

Esempi su LTS

$$X \stackrel{?}{=} [a]false \vee \langle Act \rangle X$$



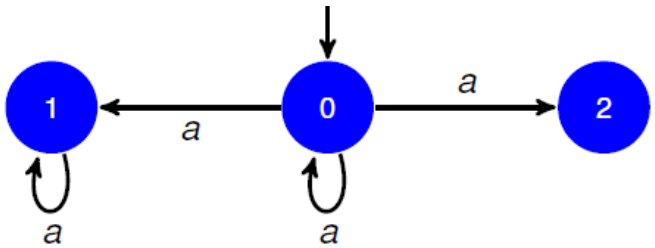
$$[a]false \vee \langle Act \rangle \emptyset = \{2\}$$

$$\overbrace{[a]false}^{\{2\}} \vee \overbrace{\langle Act \rangle \{2\}}^{\{0\}} = \{0, 2\}$$

HML con ricorsione

Esempi su LTS

$$X \stackrel{?}{=} [a]false \vee \langle Act \rangle X$$

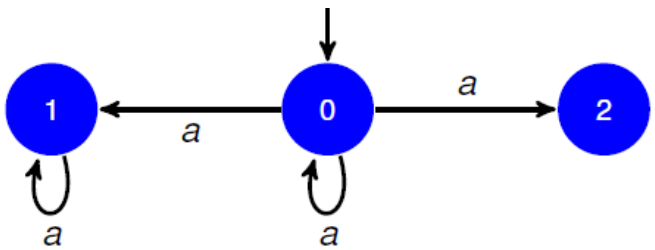


$$[a]false \vee \langle Act \rangle \emptyset = \{2\}$$
$$[a]false \vee \langle Act \rangle \{2\} = \{0, 2\}$$
$$[a]false \vee \langle Act \rangle \{0, 2\} =$$

HML con ricorsione

Esempi su LTS

$$X \stackrel{?}{=} [a]false \vee \langle Act \rangle X$$

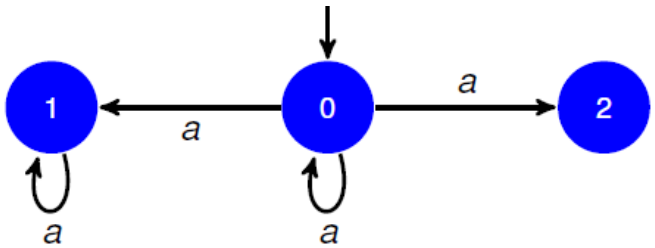


$$[a]false \vee \langle Act \rangle \emptyset = \{2\}$$
$$[a]false \vee \langle Act \rangle \{2\} = \{0, 2\}$$
$$\overbrace{[a]false}^{\{2\}} \vee \overbrace{\langle Act \rangle \{0, 2\}}^{\{0\}} = \{0, 2\}$$

HML con ricorsione

Esempi su LTS

$$X \stackrel{?}{=} [a]false \vee \langle Act \rangle X$$

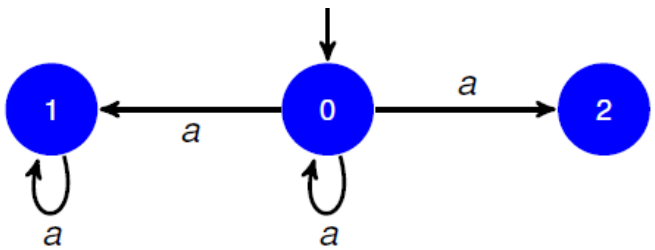


$$[a]false \vee \langle Act \rangle \emptyset = \{2\}$$
$$[a]false \vee \langle Act \rangle \{2\} = \{0, 2\}$$
$$[a]false \vee \langle Act \rangle \{0, 2\} = \{0, 2\}$$
$$[a]false \vee \langle Act \rangle \{0, 1, 2\} =$$

HML con ricorsione

Esempi su LTS

$$X \stackrel{?}{=} [a]false \vee \langle Act \rangle X$$

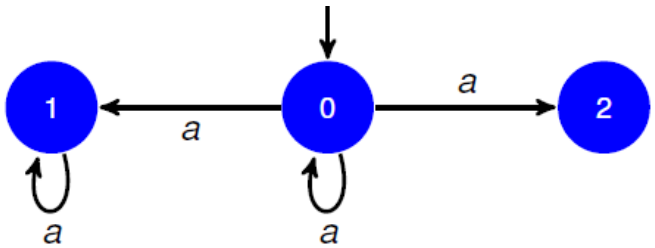


$$[a]false \vee \langle Act \rangle \emptyset = \{2\}$$
$$[a]false \vee \langle Act \rangle \{2\} = \{0, 2\}$$
$$[a]false \vee \langle Act \rangle \{0, 2\} = \{0, 2\}$$
$$\overbrace{[a]false}^{\{2\}} \vee \overbrace{\langle Act \rangle \{0, 1, 2\}}^{\{0,1\}} = \{0, 1, 2\}$$

HML con ricorsione

Esempi su LTS

$$X \stackrel{?}{=} [a]false \vee \langle Act \rangle X$$



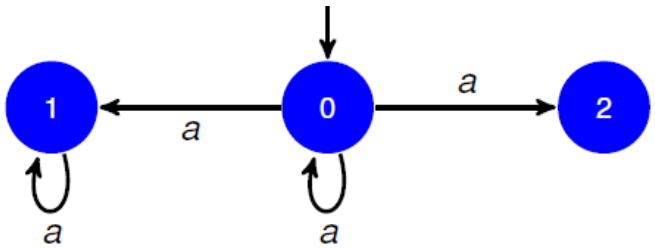
$$[a]false \vee \langle Act \rangle \emptyset = \{2\}$$
$$[a]false \vee \langle Act \rangle \{2\} = \{0, 2\}$$
$$[a]false \vee \langle Act \rangle \{0, 2\} = \{0, 2\}$$
$$[a]false \vee \langle Act \rangle \{0, 1, 2\} = \{0, 1, 2\}$$

I fixed point sono $\{0, 2\}$ e $\{0, 1, 2\}$.

HML con ricorsione

Esempi su LTS

$$X \stackrel{?}{=} [a]false \vee \langle Act \rangle X$$



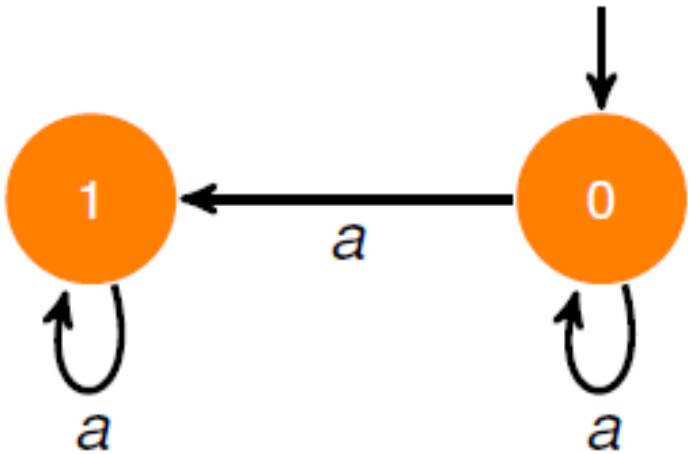
$$[a]false \vee \langle Act \rangle \emptyset = \{2\}$$
$$[a]false \vee \langle Act \rangle \{2\} = \{0, 2\}$$
$$[a]false \vee \langle Act \rangle \{0, 2\} = \{0, 2\}$$
$$[a]false \vee \langle Act \rangle \{0, 1, 2\} = \{0, 1, 2\}$$

I fixed point sono $\{0, 2\}$ e $\{0, 1, 2\}$.
La costruzione necessità del minimo fixed point.

$$X \stackrel{min}{=} [a]false \vee \langle Act \rangle X$$

In ogni stato raggiungibile una a-transizione è possibile:

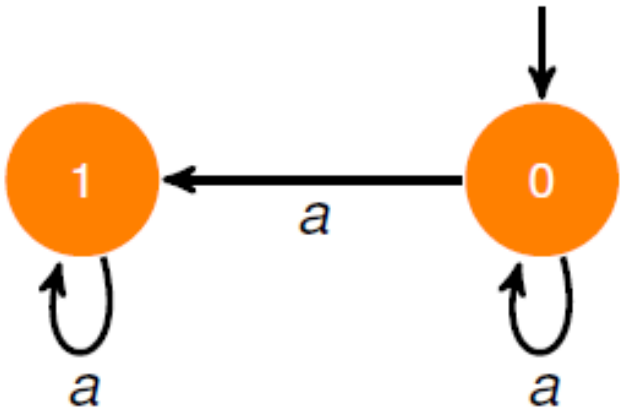
$$X \stackrel{?}{=} \langle a \rangle true \wedge [Act]X$$



HML con ricorsione

Esempi su LTS

$$X \stackrel{?}{=} \langle a \rangle true \wedge [Act]X$$

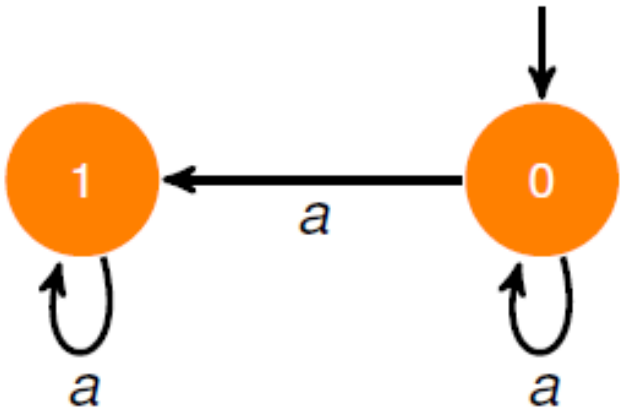


$$\langle a \rangle true \wedge [Act]\emptyset =$$

HML con ricorsione

Esempi su LTS

$$X \stackrel{?}{=} \langle a \rangle true \wedge [Act]X$$

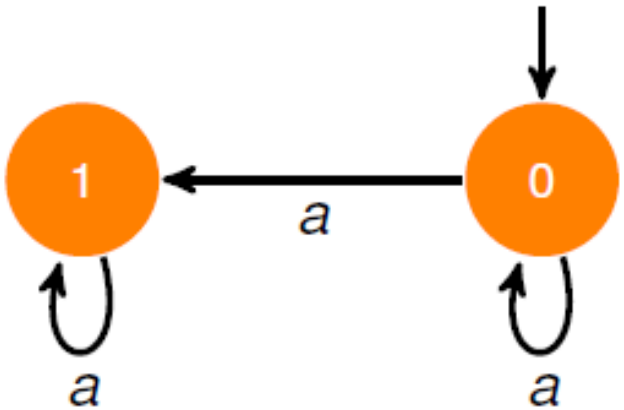


$$\overbrace{\langle a \rangle true}^{\{0,1\}} \wedge \overbrace{[Act]\emptyset}^{\emptyset} = \emptyset$$

HML con ricorsione

Esempi su LTS

$$X \stackrel{?}{=} \langle a \rangle true \wedge [Act]X$$

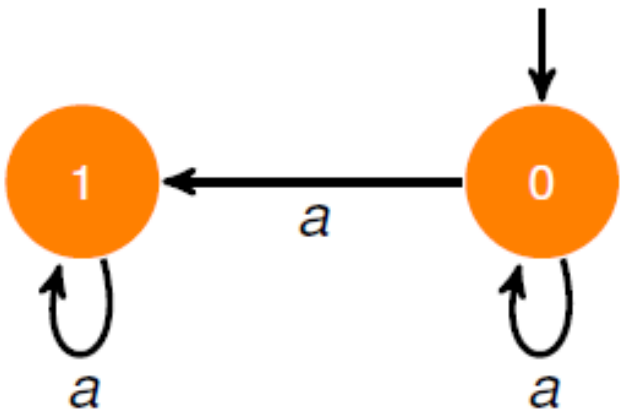


$$\langle a \rangle true \wedge [Act]\emptyset = \emptyset$$
$$\langle a \rangle true \wedge [Act]\{0,1\} =$$

HML con ricorsione

Esempi su LTS

$$X \stackrel{?}{=} \langle a \rangle true \wedge [Act]X$$



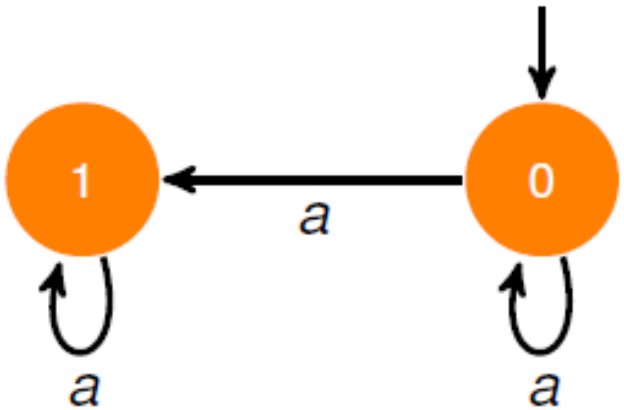
$$\overbrace{\langle a \rangle true}^{\{0,1\}} \wedge \overbrace{[Act]\emptyset}^{\{0,1\}} = \emptyset$$

$$\overbrace{\langle a \rangle true}^{\{0,1\}} \wedge \overbrace{[Act]\{0,1\}}^{\{0,1\}} = \{0,1\}$$

HML con ricorsione

Esempi su LTS

$$X \stackrel{?}{=} \langle a \rangle true \wedge [Act]X$$



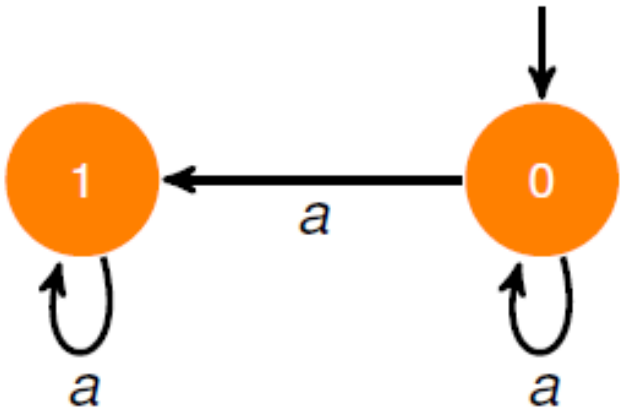
$$\langle a \rangle true \wedge [Act]\emptyset = \emptyset$$
$$\langle a \rangle true \wedge [Act]\{0,1\} = \{0,1\}$$

I fixed point sono $\emptyset$ e $\{0,1\}$.

HML con ricorsione

Esempi su LTS

$$X \stackrel{?}{=} \langle a \rangle true \wedge [Act]X$$



$$\langle a \rangle true \wedge [Act]\emptyset = \emptyset$$
$$\langle a \rangle true \wedge [Act]\{0,1\} = \{0,1\}$$

I fixed point sono $\emptyset$ e $\{0,1\}$.
La costruzione necessità del massimo fixed point.

$$X \stackrel{max}{=} [a]true \vee \langle Act \rangle X$$

HML con ricorsione

Esempi su proprietà

- Jhon non vuole bere caffè adesso
  - $\neg\langle caffe\rangle true$ oppure $[caffe]false$
- Jhon non beve mai birra
  - $X \stackrel{max}{=} [birra]false \wedge [Act]X$

HML con ricorsione

Esempi su proprietà

- Jhon non vuole bere caffè adesso
  - $\neg\langle caffe\rangle true$ oppure $[caffe]false$
- Jhon non beve mai birra
  - $X \stackrel{max}{=} [birra]false \wedge [Act]X$
- Jhon produrrà sempre una pubblicazione immediatamente dopo aver bevuto due caffè di fila
  - $[caffe][caffe](\langle pub\rangle true \wedge [Act \setminus \{pub\}]false)$
- Jhon produce sempre una pubblicazione dopo aver bevuto caffè
  - $X \stackrel{max}{=} [caffe](\langle pub\rangle true \wedge [Act \setminus \{pub\}]false) \wedge [Act]X$

μ-calculus

La logica μ-calculus descrive proprietà di un transition system.
Un transition system è una tupla:
$M = \langle S, \{R_a\}(a \in Act), P_i(i \in \mathbb{N})\rangle$

- S è l'insieme degli stati
- $\{R_a\} \subseteq SxS$ definisce transizioni per ogni azione $a \in Act$
- $\{P_i\} \subseteq S$ per ogni proposizione

Abbiamo bisogno anche di un insieme di variabili per la definizione della sintassi della logica
$Var = \{X, Y, Z...\}$

Le formule della logica sono costruite come segue:

- $p \quad and \ \neg p \quad \forall p \in P$
- $X \quad \forall X \in Var$
- $\varphi \wedge \psi; \varphi \wedge \psi;$ con $\quad \varphi, \psi$ formule
- $[a]\varphi; \quad \langle a \rangle \varphi; \quad (a \in A)$
- $\mu X.\varphi \quad and \quad \nu X.\varphi, \quad X \in Var \quad \varphi$ è una formula

Le formule della logica sono costruite come segue:

- $p \quad and \ \neg p \quad \forall p \in P$
- $X \quad \forall X \in Var$
- $\varphi \wedge \psi; \varphi \wedge \psi;$ con $\quad \varphi, \psi$ formule
- $[a]\varphi; \quad \langle a \rangle \varphi; \quad (a \in A)$
- $\mu X.\varphi \quad and \quad \nu X.\varphi, \quad X \in Var \quad \varphi$ è una formula

Esempio:
In qualche a-path ci sono infiniti stati in cui p è verificata
$\nu Y.\mu X.(p \wedge \langle a \rangle Y) \vee \langle a \rangle X)$

Il significato di una formula in un transition system è un insieme di stati che soddisfano una formula. Finchè la formula ha variabili libere, devono essere fissate prima di valutare la formula, e anche il significato della variabile sarà un insieme di stati.
$M = \langle S, \{R_a\}(a \in Act), P_i(i \in \mathbb{N})\rangle$

- $V : Var \rightarrow P(S)$
- $||X||_V^M = V(X)$ per ogni $X \in V$;
- $||p_i||_V^M = P_i \quad e \quad || \neg p_i||_V^M = S - P_i$ per ogni $p_i \in Prop$
- $||\varphi \vee \psi||_V^M = ||\varphi||_V^M \cup ||\psi||_V^M$
- $||\varphi \wedge \psi||_V^M = ||\varphi||_V^M \cap ||\psi||_V^M$

μ-calculus semantica

Semantica

La formula $\langle a \rangle \varphi$ è verificata in alcuni stati se ha una a-transizione in uscita in alcuni stati che soddisfanno φ.
Dualmente, la formula $[a]\varphi$ è verificata in alcuni stati se tutte le a-transizioni vanno in uno stato che soddisfa φ:

- $||\langle a \rangle \varphi||_V^M = \{s \in S : \exists s'.R_a(s, s') \wedge s' \in ||\varphi||_V^M\}$
- $||[a]\varphi||_V^M = \{s \in S : \forall s'.R_a(s, s') \Rightarrow s' \in ||\varphi||_V^M\}$

La formula $\langle a \rangle \varphi$ è verificata in alcuni stati se ha una a-transizione in uscita in alcuni stati che soddisfanno φ.
Dualmente, la formula $[a]\varphi$ è verificata in alcuni stati se tutte le a-transizioni vanno in uno stato che soddisfa φ:

- $||\langle a \rangle \varphi||_V^M = \{s \in S : \exists s'.R_a(s, s') \wedge s' \in ||\varphi||_V^M\}$
- $||[a]\varphi||_V^M = \{s \in S : \forall s'.R_a(s, s') \Rightarrow s' \in ||\varphi||_V^M\}$

$\alpha(X)$ contiene una variabile libera che puo' essere vista come un operatore su un insieme di stati che mappa un insieme S' alla semantica di α quando X è interpretato come S', in simboli

$$S' \mapsto [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}[S'/X]}$$

$$[\![\mu X.\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} = \bigcap \{S' \subseteq S : [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}[S'/X]} \subseteq S'\},$$

$$[\![\nu X.\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} = \bigcup \{S' \subseteq S : S' \subseteq [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}[S'/X]}\}.$$

μ-calculus semantica

Come definiamo il significato di $\nu Y.\mu X.(p \wedge \langle a \rangle Y) \vee \langle a \rangle X$?

μ-calculus semantica

Come definiamo il significato di $\nu Y.\mu X.(p \wedge \langle a \rangle Y) \vee \langle a \rangle X$?
Informalmente:

- $\mu X.[a]X$
  - Tutte le sequenze di a-trasizioni sono finite.
- $\nu Y.\langle a \rangle Y$
  - C'è una infinita sequenza di a-trasizioni.

μ-calculus semantica

Come definiamo il significato di $\nu Y.\mu X.(p \wedge \langle a \rangle Y) \vee \langle a \rangle X$?
Informalmente:

- $\mu X.[a]X$
  - Tutte le sequenze di a-trasizioni sono finite.
- $\nu Y.\langle a \rangle Y$
  - C'è una infinita sequenza di a-trasizioni.
- $\nu Y.(p \wedge \langle a \rangle Y)$
  - C'è un'infinita sequenza di a-trasizioni e tutti gli stati in questa sequenza soddisfano p.
- $\mu X.(p \wedge \langle a \rangle X)$
  - C'è una sequenza di a-trasizioni che mi porta in uno stato in cui p è verificata
- $\nu Y.\mu X.(p \wedge \langle a \rangle Y) \vee \langle a \rangle X$
  - In qualche a-path ci sono infiniti stati in cui p è verificata

Utilizzando il teorema di Knaster-Tarski, definiamo formalmente un'approssimazione di una formula fixed point come segue:
$\mu^{\tau} X.\varphi(X) \quad e \quad \nu^{\tau} X.\varphi(X)$

- $\mu^0 X.\varphi(X) = \emptyset \iff X \overset{min}{=} X$
- $\nu^0 X.\varphi(X) = S \iff X \overset{max}{=} X$

$\mu^{\tau+1} X.\varphi(X) = \varphi(Z)$ dove Z è interpretata come $\mu^{\tau} X.\varphi(X)$

Esempio: Cerchiamo il significato della formula $\mu X.[a]X$

- $\mu^0 X.\varphi(X) = \emptyset$ *false*
- $\mu^1 X.\varphi(X) = [a]\emptyset$ stati con nessun a-path
- $\mu^2 X.\varphi(X) = [a][a]\emptyset$ stati con nessun aa-path
- ...
- $\mu^\omega X.\varphi(X) = \bigcup_{n<\omega} \mu^n$ gli stati per cui $\exists$ n. no $a^n - path$
- ...

Un esempio più complesso: in alcuni percorsi ci sono infiniti stati in cui p è verificata

$\nu Y.\mu X.\langle a\rangle((p \wedge Y) \vee X)$

$$
\begin{array}{lll}
\nu^0 & S & \\
\quad \mu^{0,0} & \quad \varnothing & \\
\quad \mu^{0,1} & \quad [\![\langle a\rangle((p \wedge S) \vee \mu^{0,0})]\!] = [\![\langle a\rangle p]\!] & \\
\quad \mu^{0,2} & \quad [\![\langle a\rangle((p \wedge S) \vee \mu^{0,1})]\!] = [\![\langle a\rangle(p \vee \langle a\rangle p)]\!] & \\
\quad \dots & & \\
\nu^1 = \mu^{0,\infty} & \langle a\rangle \text{eventually}(p) & \\
\quad \mu^{1,1} & \quad [\![\langle a\rangle((p \wedge \nu^1) \vee \varnothing)]\!] = [\![\langle a\rangle(p \wedge \nu^1)]\!] & \\
\quad \mu^{1,2} & \quad [\![\langle a\rangle((p \wedge \nu^1) \vee \mu^{1,1})]\!] = [\![\langle a\rangle\big((p \wedge \nu^1) \vee \langle a\rangle(p \wedge \nu^1)\big)]\!] & \\
\quad \dots & \dots & \\
\nu^2 = \mu^{1,\infty} & \text{eventually}(p \wedge \langle a\rangle \text{eventually}(p)) & \\
\quad \dots & \dots & \\
\nu^\infty & \text{infinitely often } p &
\end{array}
$$

Dobbiamo calcolare l'approssimazione della ν formula, e durante ogni step dobbiamo calcolare l'apporssimazione della μ formula relativa alla corrente ν approssimazione

Model Checking

Il problema del Model Checking per il μ-calculus è:

Data una formula φ, un transition system M, e un suo stato s, dobbiamo decidere se

$$M, s \models \varphi$$

Finchè M è finito c'è sicuramente un finito numero di insieme di stati di M e quindi la computazione terminerà.

Model Checking

Il problema del Model Checking per il μ-calculus è:

Data una formula φ, un transition system M, e un suo stato s, dobbiamo decidere se

$$M, s \models \varphi$$

Finchè M è finito c'è sicuramente un finito numero di insieme di stati di M e quindi la computazione terminerà.

Possiamo utilizzare il μ-calculus per il problema del Model Checking?

Model Checking

Possiamo usare la semantica di μ-calculus per giochi.

Game:

Un gioco è un grafo con una partizioni di nodi tra due giocatori, chiamati Eve e Adam, e un insieme definito di condizioni vincenti.

Model Checking

Possiamo usare la semantica di μ-calculus per giochi.
Game:
Un gioco è un grafo con una partizioni di nodi tra due giocatori, chiamati Eve e Adam, e un insieme definito di condizioni vincenti.

Supponiamo che vogliamo verificare che una formula è verificata in uno stato s di un transition system M, possiamo descrivere il processo di verifica come un gioco tra due giocatori, Eve e Adam.

- L'obiettivo di Eve è dimostrare che la formula è verificata.
- L'obiettivo di Adam è dimostrare il contrario.

Theorem (Risolvere Parity Games)

Ogni posizione di un gioco con un condizione di vicità paritaria, è vincente per uno dei due giocatori. Ovvero, un giocatore ha una strategia posizionale vincente per ogni sua posizione vincente. E' algoritmicamente decidibile chi è il vincitore da una data posizione in un gioco finito con una condizione di parità.

Model Checking

Possiamo dimostrare che i parity games si riducono al Model Checking.

Theorem

$\forall \alpha$ *sentenza, un transition system* M *e uno stato* s *:*
$M, s \models \alpha \iff$ *Eve ha una strategia vincente per la posizione* (s, α) *nel gioco* G

Model Checking

Possiamo dimostrare che i parity games si riducono al Model Checking.

Theorem

$\forall \alpha$ *sentenza, un transition system* M *e uno stato* s *:*
$M, s \models \alpha \iff$ *Eve ha una strategia vincente per la posizione* (s, α) *nel gioco* G

Esiste una riduzione lineare dal Model Checking alla decisione del vicintore in un gioco parità.

Model Checking

Possiamo dimostrare che i parity games si riducono al Model Checking.

Theorem

$\forall \alpha$ *sentenza, un transition system* M *e uno stato* s *:*
$M, s \models \alpha \iff$ *Eve ha una strategia vincente per la posizione* (s, α) *nel gioco* G

Esiste una riduzione lineare dal Model Checking alla decisione del vicintore in un gioco parità.
Risolvere un gioco è in NP, e in CO-NP.

Il μ-calculus è un frammento della logica monadica al second'ordine MSO che estende la logica modale (temporale) LM:

$$LM \subseteq \text{μ-calculus} \subseteq MSO$$

Il μ-calculus è un frammento della logica monadica al second'ordine MSO che estende la logica modale (temporale) LM:

$$LM \subseteq \text{μ-calculus} \subseteq MSO$$

Quindi copre molte program logic come PDL, CTL e CTL*.
E' possibile convertire facilmente formule PDL e CTL in formule μ-calculus.
E' invece più complicato convertire formule LTL in formule μ-calculus.

Model Checking

Il problema del model checking per il μ-calculus è decidbile e appartiene a NP e CO-NP ma anche a UP ∩ co-UP.
UP = la classe dei problemi che possono essere decisi da MdT non deterministiche in tempo polinomiale con la restrinzione che per ogni input c'è al massimo una computazione accettante della MdT.
Non si sa se è decidibile in tempo polinomiale.

Come ogni logica il μ-calculus ha delle limitazoni:

Come ogni logica il μ-calculus ha delle limitazoni:

- Non consente nessuna forma di eguaglianza
  - Non possiamo dire che una transizione ha un loop in se stesso o che una transizione su a e b va nello stesso stato

Come ogni logica il μ-calculus ha delle limitazoni:

- Non consente nessuna forma di eguaglianza
  - Non possiamo dire che una transizione ha un loop in se stesso o che una transizione su a e b va nello stesso stato
- Non è possibile avere un contatore
  - Non possiamo dire che c'è solo una a-transizione da uno stato

Come ogni logica il μ-calculus ha delle limitazoni:

- Non consente nessuna forma di eguaglianza
  - Non possiamo dire che una transizione ha un loop in se stesso o che una transizione su a e b va nello stesso stato
- Non è possibile avere un contatore
  - Non possiamo dire che c'è solo una a-transizione da uno stato
- Non ha quantificatori
  - Non possiamo dire una una formula è verificata in ogni stato

Come ogni logica il μ-calculus ha delle limitazoni:

- Non consente nessuna forma di eguaglianza
  - Non possiamo dire che una transizione ha un loop in se stesso o che una transizione su a e b va nello stesso stato
- Non è possibile avere un contatore
  - Non possiamo dire che c'è solo una a-transizione da uno stato
- Non ha quantificatori
  - Non possiamo dire una una formula è verificata in ogni stato

A suo favore possiamo dire che ha una formulazione semplice, un grande potere espressivo e ha proprietà algoritmiche che lo rendono una logica propositiva per la verifica di programmi.

References

Julian Bradeld and Igor Walukiewicz

The mu-calculus and model-checking

Jeroen J.A. Keiren

Modal μ-calculus

VU University Amsterdam

Julian Bradfield and Colin Stirling (2005)

Modal Mu-Calculi

Luca Aceto1 2 Anna Ingolfsdottir1 2 Kim G. Larsen1 Jiri Srba1

Reactive Systems: Modelling, Specification and Verification

The End