Fondamenti Logici dell'Informatica

Corso di Laurea Magistrale in Informatica

Logica e Verifica

Fabio Zanasi

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Anno Accademico 2022-2023

La Verifica di Programmi e Sistemi — I

- In condizioni *normali*, i programmi e i sistemi hardware concorrenti e distribuiti *non* sono sottoposti a verifica, ma solo a testing.

La Verifica di Programmi e Sistemi — I

- In condizioni *normali*, i programmi e i sistemi hardware concorrenti e distribuiti *non* sono sottoposti a verifica, ma solo a testing.
- Quando, però, il sistema in questione è *mission critical*, ossia le conseguenze di un suo eventuale malfunzionamento risulterebbero devastanti, si può ricorrere ad una parziale o totale verifica.
- Esempi di sistemi mission critical:
  - **Avionica**.
  - **Centrali Nucleari**.
  - **Transazioni Bancarie**.

La Verifica di Programmi e Sistemi — I

- In condizioni *normali*, i programmi e i sistemi hardware concorrenti e distribuiti *non* sono sottoposti a verifica, ma solo a testing.
- Quando, però, il sistema in questione è *mission critical*, ossia le conseguenze di un suo eventuale malfunzionamento risulterebbero devastanti, si può ricorrere ad una parziale o totale verifica.
- Esempi di sistemi mission critical:
  - **Avionica**.
  - **Centrali Nucleari**.
  - **Transazioni Bancarie**.
- Ma cosa significa verificare la correttezza di un sistema?

La Verifica di Programmi e Sistemi — I

- In condizioni *normali*, i programmi e i sistemi hardware concorrenti e distribuiti *non* sono sottoposti a verifica, ma solo a testing.
- Quando, però, il sistema in questione è *mission critical*, ossia le conseguenze di un suo eventuale malfunzionamento risulterebbero devastanti, si può ricorrere ad una parziale o totale verifica.
- Esempi di sistemi mission critical:
  - **Avionica**.
  - **Centrali Nucleari**.
  - **Transazioni Bancarie**.
- Ma cosa significa verificare la correttezza di un sistema?
- Data la descrizione di un sistema $\mathcal{S}$ e una proprietà P che descriva il comportamento atteso, occorre **verificare** che $\mathcal{S}$ effettivamente soddisfi P.

La Verifica di Programmi e Sistemi — II

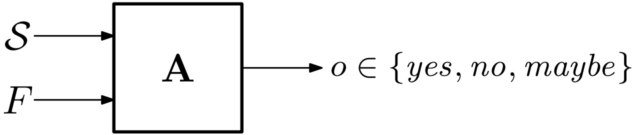
- L'informatica si è interessata fin dalla sua nascita al problema della verifica di correttezza di programmi e sistemi.

La Verifica di Programmi e Sistemi — II

- L'informatica si è interessata fin dalla sua nascita al problema della verifica di correttezza di programmi e sistemi.
- Cosa succede se si insiste sul fatto che la verifica debba essere fatta in modo *automatico*?

La Verifica di Programmi e Sistemi — II

- L'informatica si è interessata fin dalla sua nascita al problema della verifica di correttezza di programmi e sistemi.
- Cosa succede se si insiste sul fatto che la verifica debba essere fatta in modo *automatico*?
- Quasi sempre lo scenario è rappresentabile come segue:



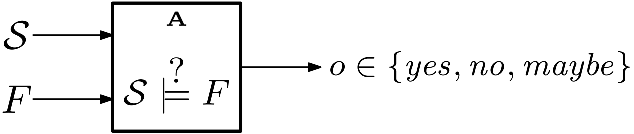
- Il fatto che tra le possibili risposte di **A** ci sia anche *maybe* dipende dal fatto che il problema di verificare se $\mathcal{S}$ soddisfa P è spesso indecidibile.
  - In tal caso non si può fare altro che risolvere un'approssimazione del problema.

Il Model Checking

- Nel model checking, si fa verifica considerando la proprietà P come una formula di una opportuna logica e il sistema come un'interpretazione per essa.

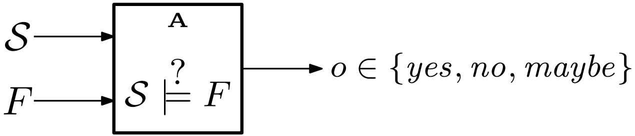
Il Model Checking

- Nel model checking, si fa verifica considerando la proprietà P come una formula di una opportuna logica e il sistema come un'interpretazione per essa.
- Graficamente:



Il Model Checking

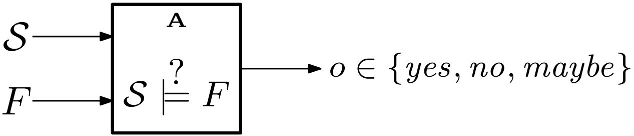
- Nel model checking, si fa verifica considerando la proprietà P come una formula di una opportuna logica e il sistema come un'interpretazione per essa.
- Graficamente:



- Bisogna però capire:
  - In che senso un *sistema* o *programma* possa essere visto come un'interpretazione.

Il Model Checking

- Nel model checking, si fa verifica considerando la proprietà P come una formula di una opportuna logica e il sistema come un'interpretazione per essa.
- Graficamente:



- Bisogna però capire:
  - In che senso un *sistema* o *programma* possa essere visto come un'interpretazione.
  - Quale sia la *logica* adatta a specificare le proprietà d'interesse.

Le Strutture di Kripke

- Supponiamo che AP sia un insieme di proposizioni atomiche, che catturino le proprietà di interesse di uno stato, magari astraendo sullo stato stesso.

Le Strutture di Kripke

- Supponiamo che AP sia un insieme di proposizioni atomiche, che catturino le proprietà di interesse di uno stato, magari astraendo sullo stato stesso.
- Una **struttura di Kripke** su AP è una quadrupla $\mathcal{M} = (S, S_0, R, L)$ dove:
  - S è un insieme di *stati*.
  - $S_0 \subseteq S$ è l'insieme degli *stati iniziali*.
  - $R \subseteq S \times S$ è la *relazione di transizione*, che supponiamo totale: per ogni $s \in S$ esiste $t \in S$ con $(s, t) \in R$.
  - $L : S \to \mathbf{P}(AP)$ è una *funzione di etichettatura*.

Le Strutture di Kripke

- Supponiamo che AP sia un insieme di proposizioni atomiche, che catturino le proprietà di interesse di uno stato, magari astraendo sullo stato stesso.
- Una **struttura di Kripke** su AP è una quadrupla $\mathcal{M} = (S, S_0, R, L)$ dove:
  - S è un insieme di *stati*.
  - $S_0 \subseteq S$ è l'insieme degli *stati iniziali*.
  - $R \subseteq S \times S$ è la *relazione di transizione*, che supponiamo totale: per ogni $s \in S$ esiste $t \in S$ con $(s, t) \in R$.
  - $L : S \to \mathbf{P}(AP)$ è una *funzione di etichettatura*.
- L'insieme degli stati si suppone spesso essere finito, questo per garantire la decidibilità.
- La funzione di etichettatura ha il ruolo di dire quale proposizione atomiche valgono in ogni stato.

Programmi Concorrenti come Strutture

- Consideriamo il seguente *programma concorrente*:

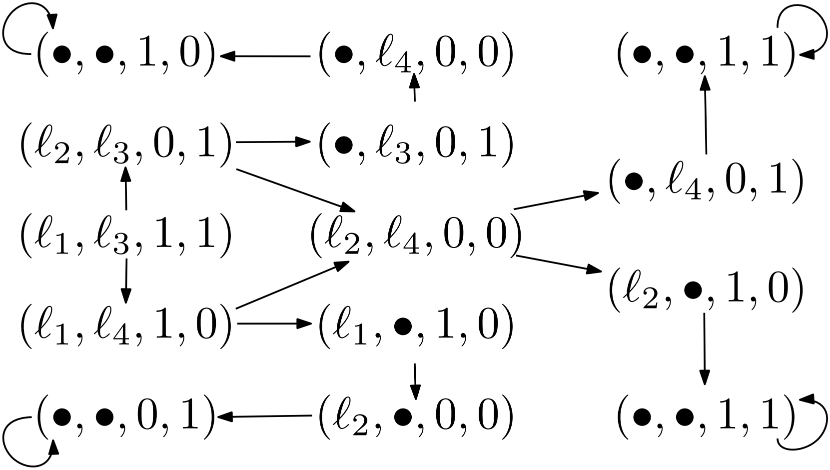
$$(\ell_1 : \ \mathtt{x} \leftarrow 0 \ ; \ \ell_2 : \ \mathtt{y} \leftarrow 1) \, || \, (\ell_3 : \ \mathtt{y} \leftarrow 0 \ ; \ \ell_4 : \ \mathtt{x} \leftarrow 1)$$

- L'insieme degli *stati* di questo programma può essere visto come l'insieme

$$\{\ell_1, \ell_2, \bullet\} \times \{\ell_3, \ell_4, \bullet\} \times \{0, 1\} \times \{0, 1\}$$

  mentre l'unico stato iniziale potrebbe essere $(\ell_1, \ell_3, 1, 1)$.
- La relazione di transizione corrisponde a quella intuitiva, ed è costruita usando il principio dell'interleaving.

Programmi Concorrenti come Strutture



Programmi Concorrenti come Strutture

- L'insieme AP potrebbe *per esempio* contenere le seguenti proposizioni:
  - Una proposizione chiamata `null`, che modella il fatto che entrambe le variabili x e y valgono 0.
  - Una proposizione chiamata `stop`, che modella la terminazione del programma.

Programmi Concorrenti come Strutture

- L'insieme AP potrebbe *per esempio* contenere le seguenti proposizioni:
  - Una proposizione chiamata `null`, che modella il fatto che entrambe le variabili x e y valgono 0.
  - Una proposizione chiamata `stop`, che modella la terminazione del programma.
- Fromalmente,
  - `null` $\in L(S)$ se e solo se le ultime due componenti di S sono entrambe 0.
  - `stop` $\in L(S)$ se e solo se le prime due componenti di S sono entrambe $\bullet$.

Programmi Concorrenti come Strutture

- L'insieme AP potrebbe *per esempio* contenere le seguenti proposizioni:
  - Una proposizione chiamata `null`, che modella il fatto che entrambe le variabili x e y valgono 0.
  - Una proposizione chiamata `stop`, che modella la terminazione del programma.
- Fromalmente,
  - $\mathtt{null} \in L(S)$ se e solo se le ultime due componenti di S sono entrambe 0.
  - $\mathtt{stop} \in L(S)$ se e solo se le prime due componenti di S sono entrambe $\bullet$.
- Ciò non toglie che AP possa contenere anche tante altre proposizioni atomiche, come per esempio $\mathtt{x} = 0$, $\mathtt{y} = 1$, ℓ_1, il cui significato è intuitivo.

Come Speficare Proprietà di Sistemi?

- Un'idea ovvia e anche abbastanza interessante consiste nell'utilizzare come linguaggio in cui specificare le proprietà dei sistemi nient'altro che la **logica proposizionale** in cui le proposizioni atomiche sono gli atomi.

Come Speficare Proprietà di Sistemi?

- Un'idea ovvia e anche abbastanza interessante consiste nell'utilizzare come linguaggio in cui specificare le proprietà dei sistemi nient'altro che la **logica proposizionale** in cui le proposizioni atomiche sono gli atomi.
- In questo modo possiamo parlare del sistema in senso statico, ossia della struttura di Kripke *e di un suo stato*.

Come Speficare Proprietà di Sistemi?

- Un'idea ovvia e anche abbastanza interessante consiste nell'utilizzare come linguaggio in cui specificare le proprietà dei sistemi nient'altro che la **logica proposizionale** in cui le proposizioni atomiche sono gli atomi.
- In questo modo possiamo parlare del sistema in senso statico, ossia della struttura di Kripke *e di un suo stato*.
- Ad esempio potremmo concludere che:

$$\mathcal{M}, (\ell_1, \ell_3, 0, 0) \models \texttt{null} \wedge \neg\texttt{stop}$$

$$\mathcal{M}, (\bullet, \bullet, 1, 1) \models \texttt{null} \rightarrow \texttt{stop}$$

- Al limite, si potrebbe per esempio scrivere $\mathcal{M} \models F$ se solo se $\mathcal{M}, S \models F$ per ogni $S \in S_0$.
- Manca però *completamente* l'aspetto dinamico. Come esprimere proprietà relative all'**evoluzione** del sistema?
  - Ad esempio: **reachability**, **safety**, etc.

Logiche Temporali

- Le logiche temporali possono essere viste come estensioni della logica proposizionale ottenute dotando quest'ultima di **operatori modali** che permettano di esprimere:
  - In che senso una formula vale *nel futuro*.
  - Il fatto che certe formule valgano *in alcune* esecuzioni nondeterministiche, oppure *in tutte*.
- **Operatori Temporali**
  - Potremmo per esempio voler dire che una certa formula F *vale ora e rimane valida nel futuro*, e in tal caso scriviamo $(\mathsf{G}\ F)$.
  - Altra cosa è dire che una certa formula F vale *dopo la prossima* transizione di stato, formalmente $(\mathsf{X}\ F)$.
  - Infine, potremmo voler dire che una formula F vale *in un certo istante inefinito del futuro*, formalmente $(\mathsf{F}\ F)$.
- **Quantificatori sui Cammini**
  - Potremmo voler dire che una formula F vale *indipendentemente* dal nondeterminismo, oppure che vale per *almeno una* scelta nondeterministica. Scriveremo, rispettivamente, $\mathsf{A}\ F$ e $\mathsf{E}\ F$.

La Logica Temporale CTL*: Sintassi

- Gli operatori temporali producono formule da valutare su *cammini* di esecuzione, mentre i quantificatori sui cammini vengono valutati su *stati*.

La Logica Temporale CTL^*: Sintassi

- Gli operatori temporali producono formule da valutare su *cammini* di esecuzione, mentre i quantificatori sui cammini vengono valutati su *stati*.
- **Formule di Stato** su AP.

$$F_S, G_S ::= P \;\big|\; F_S \wedge G_S \;\big|\; F_S \vee G_S \;\big|\; \neg F_S \;\big|\; \mathsf{E}\, F_P \;\big|\; \mathsf{A}\, F_P.$$

dove P è una proposizione atomica in AP.

- **Formule di Cammino** su AP.

$$F_P, G_P ::= F_S \;\big|\; F_P \wedge G_P \;\big|\; F_P \vee G_P \;\big|\; \neg G_P \;\big|\; \mathsf{X}\, F_P \;\big|$$
$$\mathsf{F}\, F_P \;\big|\; \mathsf{G}\, F_P \;\big|\; F_P\, \mathsf{U}\, G_P \;\big|\; F_P\, \mathsf{R}\, G_P$$

La Logica Temporale CTL^*: Semantica

- Un **cammino** π in un struttura di Kripke $\mathcal{M} = (S, S_0, R, L)$ è una sequenza infinita di stati $s_0 s_1 s_2 \ldots \in S^\omega$ tale che $(s_n, s_{n+1}) \in R$ per ogni naturale n.
- Dato un cammino π e un naturale n, indicheremo con π^n l'n-esimo suffisso di π, anch'esso cammino.

La Logica Temporale CTL^*: Semantica

- Un **cammino** π in un struttura di Kripke $\mathcal{M} = (S, S_0, R, L)$ è una sequenza infinita di stati $s_0 s_1 s_2 \ldots \in S^\omega$ tale che $(s_n, s_{n+1}) \in R$ per ogni naturale n.
- Dato un cammino π e un naturale n, indicheremo con π^n l'n-esimo suffisso di π, anch'esso cammino.
- Una formula di stato F_S su AP è vera in una struttura di Kripke $\mathcal{M}$ su AP e in uno stato s di $\mathcal{M}$. In tal caso scriveremo
$$\mathcal{M}, s \models F_S.$$

La Logica Temporale CTL^*: Semantica

- Un **cammino** π in un struttura di Kripke $\mathcal{M} = (S, S_0, R, L)$ è una sequenza infinita di stati $s_0 s_1 s_2 \ldots \in S^\omega$ tale che $(s_n, s_{n+1}) \in R$ per ogni naturale n.
- Dato un cammino π e un naturale n, indicheremo con π^n l'n-esimo suffisso di π, anch'esso cammino.
- Una formula di stato F_S su AP è vera in una struttura di Kripke $\mathcal{M}$ su AP e in uno stato s di $\mathcal{M}$. In tal caso scriveremo
$$\mathcal{M}, s \models F_S.$$
- Una formula di cammino F_P su AP è vera in una struttura di Kripke $\mathcal{M}$ su AP e in un cammino π in $\mathcal{M}$. In tal caso scriveremo
$$\mathcal{M}, \pi \models F_P.$$

CTL*: Semantica delle Formule di Stato

- I connettivi $\neg$, $\wedge$ e $\vee$ sono interpretati in modo standard.
- Le proposizioni atomiche si interpretano facendo riferimento alla struttura di Kripke:

  $$(S, S_0, R, L), s \models P$$

  se e solo se $P \in L(s)$.
- I quantificatori sui cammini fanno ovviamente riferimento alla semantica delle formule di cammino:

  $$\mathcal{M}, s \models (\mathsf{E}\, F_P) \text{ sse } \mathcal{M}, \pi \models F_P$$
  $$\text{per almeno un cammino } \pi \text{ che inizi in } s;$$
  $$\mathcal{M}, s \models (\mathsf{A}\, F_P) \text{ sse } \mathcal{M}, \pi \models F_P$$
  $$\text{per tutti i cammini } \pi \text{ che iniziano in } s.$$

CTL*: Semantica delle Formule di Cammino

- I connettivi $\neg$, $\wedge$ e $\vee$ sono interpretati in modo standard.
- Le formule di stato si valutano nel primo stato del cammino:

$$\mathcal{M}, \pi \models F_S \text{ sse } \mathcal{M}, s \models F_S.$$

CTL*: Semantica delle Formule di Cammino

- I connettivi $\neg$, $\wedge$ e $\vee$ sono interpretati in modo standard.
- Le formule di stato si valutano nel primo stato del cammino:

$$\mathcal{M}, \pi \models F_S \text{ sse } \mathcal{M}, s \models F_S.$$

- I quantificatori sui cammini fanno ovviamente riferimento alla semantica delle formule di cammino:

$\mathcal{M}, \pi \models (\mathsf{X}\ F_P)$ sse $\mathcal{M}, \pi^1 \models F_P$

$\mathcal{M}, \pi \models (\mathsf{F}\ F_P)$ sse $\mathcal{M}, \pi^i \models F_P$ per almeno un i;

$\mathcal{M}, \pi \models (\mathsf{G}\ F_P)$ sse $\mathcal{M}, \pi^i \models F_P$ per tutti gli i;

$\mathcal{M}, \pi \models (F_P\ \mathsf{U}\ G_P)$ sse esiste k naturale

con $\mathcal{M}, \pi^k \models G_P$ e $\mathcal{M}, \pi^j \models F_P$ per ogni $0 \leq j < k$;

$\mathcal{M}, \pi \models (F_P\ \mathsf{R}\ G_P)$ sse per ogni j naturale,

se per ogni $i < j$, $\mathcal{M}, \pi^i \not\models F_P$, allora $\mathcal{M}, \pi^j \models G_P$.

Due Frammenti di CTL^*

- **La Logica** CTL.
  - Ogni operatore temporale *deve essere* immediatamente preceduto da un quantificatore sui cammini.
  - In altre parole, la grammatica per le formule di cammino diventa molto più semplice:

$$F_P, G_P ::= \mathsf{X}\, F_S \;\Big|\; \mathsf{F}\, F_S \;\Big|\; \mathsf{G}\, F_S \;\Big|\; F_S \,\mathsf{U}\, G_S \;\Big|\; F_S \,\mathsf{R}\, G_S$$

Due Frammenti di CTL^*

- **La Logica** CTL.
  - Ogni operatore temporale *deve essere* immediatamente preceduto da un quantificatore sui cammini.
  - In altre parole, la grammatica per le formule di cammino diventa molto più semplice:

$$F_P, G_P ::= \mathsf{X}\ F_S \ \Big|\ \mathsf{F}\ F_S \ \Big|\ \mathsf{G}\ F_S \ \Big|\ F_S\ \mathsf{U}\ G_S \ \Big|\ F_S\ \mathsf{R}\ G_S$$

- **La Logica** LTL.
  - Le uniche formule considerate sono le formule di cammino, che però vengono implicitamente quantificate con il quantificatore $\mathsf{A}\ \cdot$.
  - In altre parole, le formule diventano:

$$F_P, G_P ::= P \ \Big|\ F_P \wedge G_P \ \Big|\ F_P \vee G_P \ \Big|\ \neg F_P \ \Big|\ \mathsf{X}\ F_P \ \Big|$$
$$\mathsf{F}\ F_P \ \Big|\ \mathsf{G}\ F_P \ \Big|\ F_P\ \mathsf{U}\ G_P \ \Big|\ F_P\ \mathsf{R}\ G_P$$

Il Problema del Model Checking

- **Model Checking Universale**.
  - Data una struttura di Kripke $\mathcal{M} = (S, S_0, R, L)$ e una formula di stato F_S, determinare se $\mathcal{M}, s \models F_S$ *per ogni* $s \in S_0$.

Il Problema del Model Checking

- **Model Checking Universale**.
  - Data una struttura di Kripke $\mathcal{M} = (S, S_0, R, L)$ e una formula di stato F_S, determinare se $\mathcal{M}, s \models F_S$ *per ogni* $s \in S_0$.
- **Model Checking Esistenziale**
  - Data una struttura di Kripke $\mathcal{M} = (S, S_0, R, L)$ e una formula di stato F_S, determinare se *esiste* $s \in S_0$ con $\mathcal{M}, s \models F_S$.

Il Problema del Model Checking

- **Model Checking Universale**.
  - Data una struttura di Kripke $\mathcal{M} = (S, S_0, R, L)$ e una formula di stato F_S, determinare se $\mathcal{M}, s \models F_S$ *per ogni* $s \in S_0$.
- **Model Checking Esistenziale**
  - Data una struttura di Kripke $\mathcal{M} = (S, S_0, R, L)$ e una formula di stato F_S, determinare se *esiste* $s \in S_0$ con $\mathcal{M}, s \models F_S$.

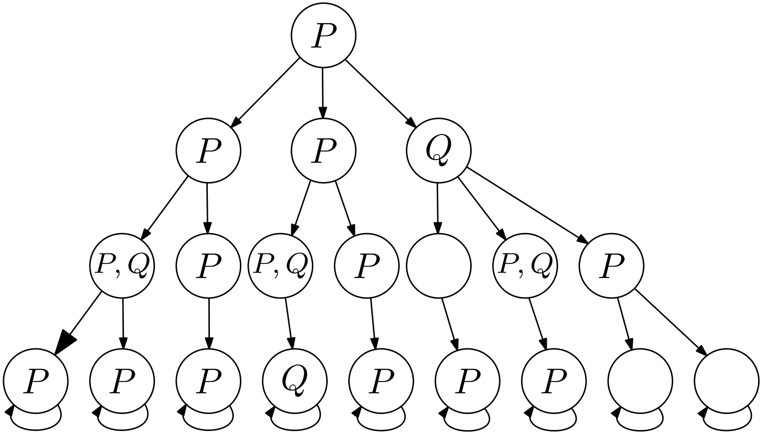
Teorema

I problemi del model checking universale e esistenziale sono PSPACE-completi per CTL^*.

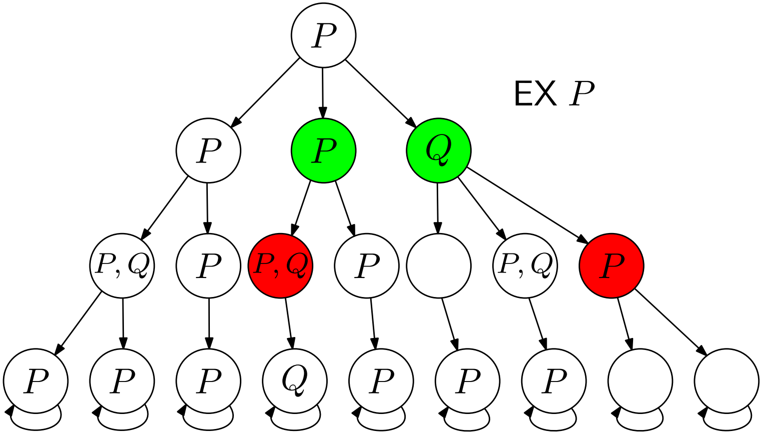
Teorema

I problemi del model checking universale e esistenziale per CTL *sono risolvibili in tempo polinomiale.*

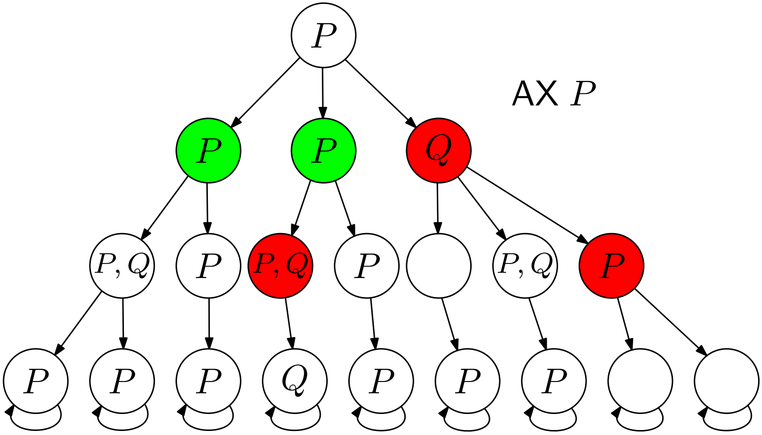
Esempi



Esempi

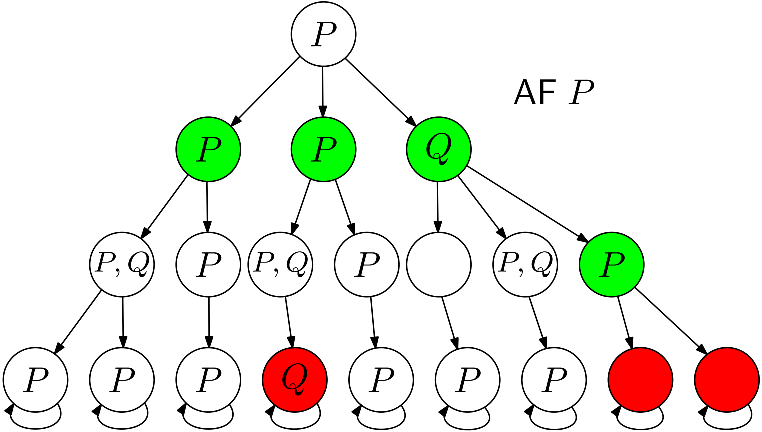


Esempi



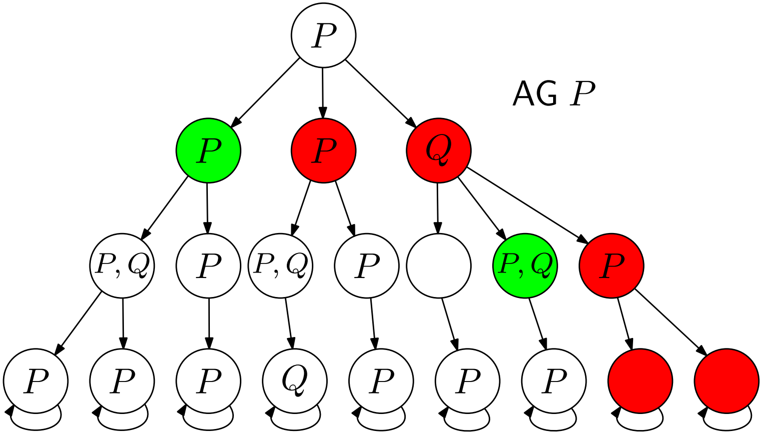
AX P

Esempi



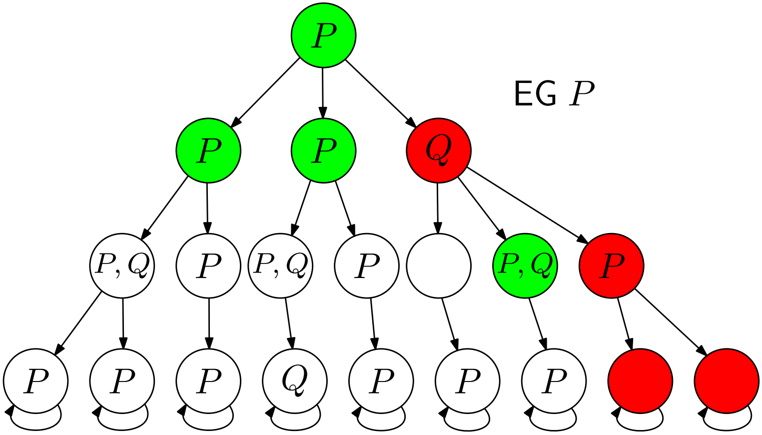
AF P

Esempi

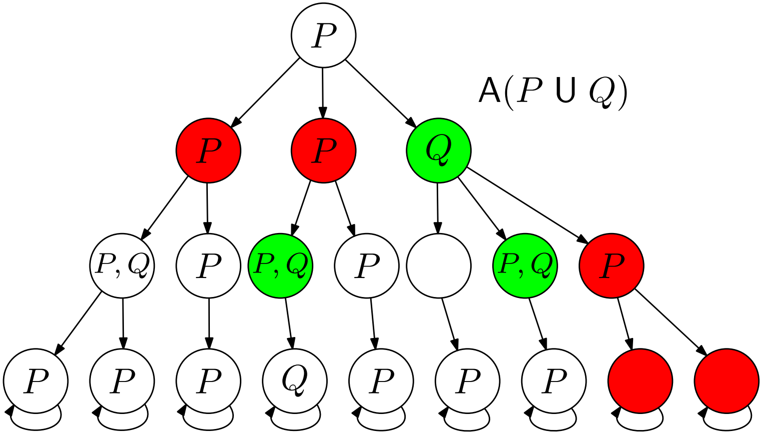


AG P

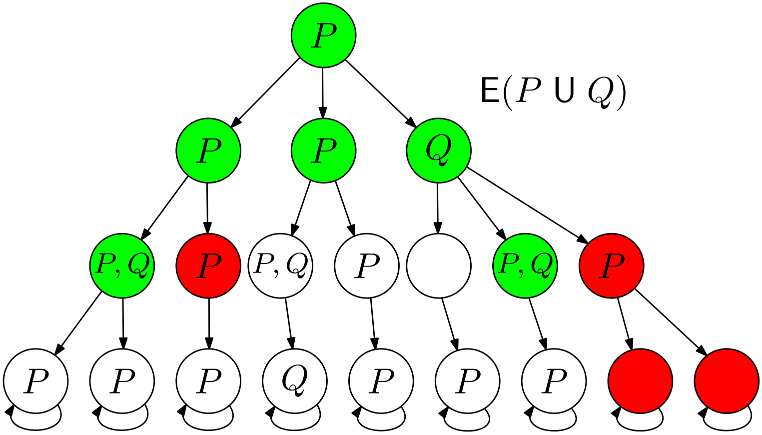
Esempi



Esempi



Esempi



$\mathsf{E}(P \mathbin{\mathsf{U}} Q)$