Laurea Magistrale in Informatica A.A. 2020/2021
Università degli Studi di Milano-Bicocca

Appunti Modelli della Concorrenza

Marta Pelusi
@Marta629

Indice

1 Introduzione

Questo capitolo è pensato per inquadrare in modo generale gli argomenti trattati in questo corso.
Si parte dalla **logica di Hoare**, introdotta dal matematico Hoare, al fine di dimostrare la correttezza di programmi sequenziali tramite le triple di Hoare, espresse nella forma $\{p\}P\{q\}$. p, q sono rispettivamente la precondizione e la postcondizione del programma P. In poche parole ci si sta chiedendo se, partendo da uno stato della memoria in cui vale p, eseguendo i comandi in P, si arriva ad uno stato della memoria (diverso da quello di partenza a causa dell'esecuzione di P) in cui vale q.
Negli anni 1978-1980, quasi 30'anni dopo la nascità dei sistemi di transazioni etichettati - LTS anche meglio conosciuti come automi a stati finiti, Hoare e Milner avevano iniziato a studiare un nuovo paradigma di programmazione chiamato Comunicationg Sequential Processing - CSP introducendo la nozione di paradigma di processo, ovvero:

- no memoria condivisa
- insieme di processi dove ogni processo ha memoria privata e comportamento autonomo
- interazione tra processi tramite lo scambio di messaggi *hand shaking*

Milner, successivamente, introdurrà il λ-calcolo con l'obiettivo di passare dallo studio di programmi sequenziali a programmi/sistemi concorrenti. È questo lo scopo del **Calcolus of Comunicating Systems - CCS**, ovvero lo studio di un sistema costituito da componenti/processi che comunicano tra loro tramite scambio di messaggi in modo sincrono. Una particolarità di questi sistemi è che, oltre che interagire tra loro, possono anche interagire con l'ambiente esterno. Si va quindi a studiare lo stato globale del sistema. Un esempio di sistema concorrente può essere una cellula umana, dentro la quale vengono eseguiti dei processi in modo asincrono (ogni componente lavora per i fatti suoi), oppure un'orchestra, nella quale tutti suonano simultaneamente, quindi in modo sincrono.
Un concetto fondamentale introdotto per i sistemi CCS è il concetto di **Bisimulazione**. Questo è stato introdotto a seguito dell'esempio della macchinetta del caffè:

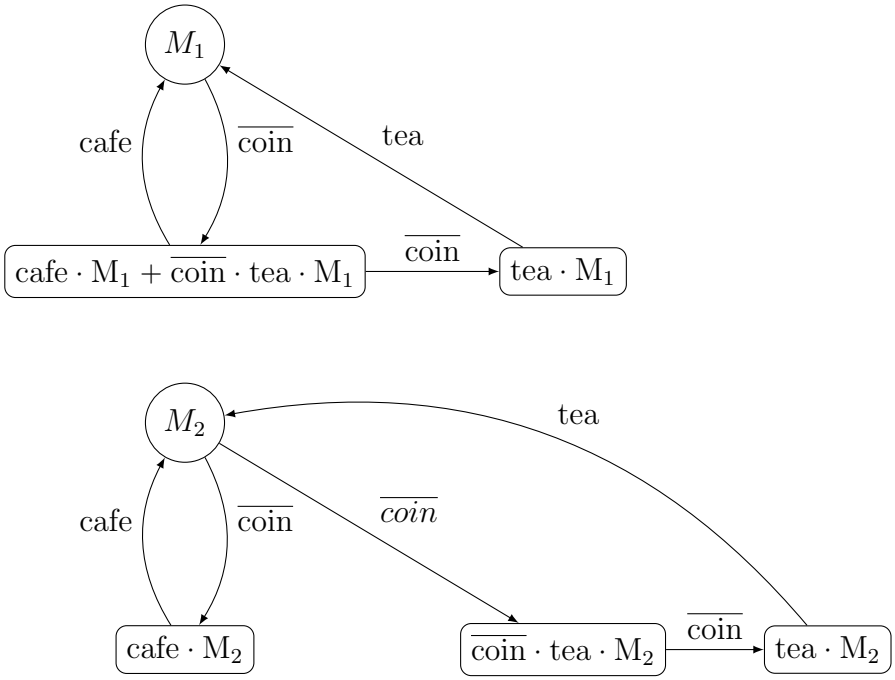


Figura 1: Esempio macchinetta del caffè.

La cosa che differisce M_1 da M_2 è che in M_2 è stato separato lo stato con la scelta $(+)$ presente in M_1 in due stati differenti. I processi M_1, M_2 sono equivalenti rispetto alle tracce. A questo punto ci si chiede se i due sistemi hanno effettivamente lo stesso comportamento, ovvero ci si sta chiedendo se i due sistemi sono bisimili. La risposta a questa domanda è no, in quanto, facendo $\xrightarrow{\overline{\text{coin}}}$ da M_1 si può decidere se prendere subito il caffè inserendo una moneta. In M_2, invece, non si può imporre di prendere subito il caffè in quanto, facendo $\xrightarrow{\overline{\text{coin}}}$, si può capitare nello stato $\overline{\text{coin}} \cdot \text{tead} \cdot \text{M}_2$ che aspetta che venga inserita un'altra moneta per poi prendere il te. In questo caso la persona che usufruisce della macchinetta aspetta che le venga restituito il caffè mentre la macchinetta aspetta che venga inserita un'altra moneta. Questa situazione di attesa "infinita" viene chiamata deadlock. È per questo motivo, dunque, che viene introdotto il concetto di bisimulazione.
Nel 1962 Petri nella sua tesi di laurea scrisse a proposito della Teoria generale delle Reti di Petri. Petri criticava gli automi a stati finiti in quanto sosteneva che:

- nei sistemi distribuiti lo stato globale non è osservabile
- non esiste un sistema temporale unico
- la simulazione sequenziale non deterministica è una forzatura

Sulla base di questo introduce le **Reti di Petri**, che sono dei sistemi sincroni che si focalizzano sullo studio di stati locali, ovvero su diverse configurazioni del sistema partendo da una configurazione iniziale.
Alcuni anni dopo si volle fare un passo avanti: studiare programmi/sistemi che

non presentavano uno stato iniziale e/o finale. Si voleva, quindi, andare a studiare la correttezza di sistemi reattivi, che sono sistemi:

- concorrenti
- distribuiti
- asincroni

Questo non era possibile farlo tramite la logica di Hoare in quanto essa studiava programmi sequenziali e non concorrenti; non era possibile farlo nemmeno utilizzando le Reti di Petri perchè non era fornito uno stato/configurazione iniziale. Dunque, sono qui introdotti il **Model Checking e le Logiche Temporali**: formule formali che fanno riferimento a stati futuri del sistema e non al singolo stato della memoria. Il sistema, in questo senso, è rappresentato tramite il Modello di Kripke.

2 Dimostrazioni di correttezza di programmi

Definizione 1. *La **precondizione** è una formula che esprime ciò che si suppone vero all'inizio dell'esecuzione di un programma.*

Definizione 2. *La **postcondizione** è una formula che esprime ciò che ci si aspetta che sia vero al termine dell'esecuzione di un programma.*

Definizione 3. *Lo **stato della memoria** è una funzione $s : V \to \mathbb{Z}$ che associa ad ogni variabile $v \in V$ del programma il suo valore in un dato istante.*

Fissato lo stato della memoria s e una formula ϕ si può verificare se una formula è vera in quel dato stato.

Definizione 4. *Una **formula invariante** è una formula che se è vera all'inizio di un'iterazione è vera anche alla fine della stessa iterazione.*

2.1 Logica di Hoare

Le **triple di Hoare** sono così rappresentate: $\alpha\ P\ \beta$.
P è un comando, ovvero una seingola istruzione o un gruppo di istruzioni strutturate.

- C ::= indica le produzioni, ovvero i modi con cui si può costruire un comando:
  - x ::= E con x assegnamento e E espressione
  - $C; C$ combinazione di due comandi in sequenza
  - $if\ B\ then\ C\ else\ D\ endif$ è un'istruzione di scelta con B espressione booleana
  - $while\ B\ do\ C\ endwhile$ è un'istruzione iterativa

  - *skip* è un'istruzione speciale fittizia che non fa niente

- B ::= espressione booleana:
  - $true$
  - $false$
  - $not\ B$
  - $B\ and\ B$
  - $B\ or\ B$
  - $E < E$ oppure $E = E$

Regole di derivazione

Le regole di derivazione sono delle regole in cui date delle precondizioni/premesse $\alpha_1, \alpha_2, ..., \alpha_k$ si arriva ad una postcondizione/conclusione α.
Questo è così indicato: $\frac{\alpha_1,\alpha_2,...,\alpha_k}{\alpha}$

- *skip* non cambia lo stato della memoria: $\frac{}{\{p\}\ skip\ \{p\}}$
- regola di conseguenza/regola dell'implicaizone: $\frac{p \to p'\ \{p'\}C\{q\}}{\{p\}C\{q\}}$, $\frac{\{p'\}C\{q\}\ q' \to q}{\{p\}C\{q\}}$
- sequenza: $\frac{\{p\}C_1\{q\}\ \{q\}C_2\{r\}}{\{p\}C_1;C_2\{r\}}$
- assegnamento: $\frac{}{p[E/x]\ x:=\ E\{p\}}$, con E espressione numerica e $p[E/x]$ che significa: cercare nella formula p tutte le occorrenze di x e sostituirle con l'espressione E, ovvero E/x indica una sostituzione
- istruzioni di scelta: si hanno due premesse: $\{p \wedge B\}C\{q\}$ dove B è vera e $\{p \wedge \neg B\}D\{q\}$ dove B è falsa, quindi $\frac{\{p \wedge B\}C\{q\}\ \{p \wedge \neg B\}D\{q\}}{\{p\}\ if\ B\ then\ C\ else\ D\ endif\ \{q\}}$
- istruzioni iterative: $\frac{\{i \wedge B\}C\{i\}}{\{i\}W\{i \wedge \neg B\}}$

2.1.1 Correttezza parziale

Per le **istruzioni iterative** può essere che il ciclo non termini mai andando in un loop infinito. Quindi, per comodità, va fatta una distinzione tra correttezza parziale e correttezza totale.
Supponiamo, dunque, che il ciclo termini sempre trovandoci nel caso della correttezza parziale, ovvero il caso in cui vale $W\{q \wedge \neg B\}$.
Supponendo di avere la tripla derivata $\{i \wedge B\}C\{i\}$, si può notare che i vale sia prima dell'esecuzione del comando C, sia dopo. Questo viene chiamato **invariante di ciclo**.
La regola di derivazione per istruzioni iterative, considerando la correttezza parziale, è dunque la seguente: $\frac{\{inv \wedge B\}C\{inv\}}{\{inv\}\ while\ B\ do\ C\ endwhile\ \{i \wedge \neg B\}}$, dove inv è l'invariante di ciclo.

2.1.2 Correttezza totale

Tra la correttezza parziale e la correttezza tottale c'è una sottilissima differenza:
Correttezza parziale: se si esegue W a partire da uno stato in cui vale p e l'esecuzione termina, nello stato finale vale q.
Correttezza totale: se si esegue W a partire da uno stato in cui vale p, l'esecuzione termina e nello stato finale vale q.
In soldoni: la correttezza parziale suppone a priori che l'esecuzione del ciclo termini, mentre la correttezza totale mira a dimostrare che l'esecuzione del ciclo termina.

Tecnica di dimostrazione per la correttezza totale: supponendo che E sia un'espressione aritmetica nella quale compaiono le variabili del programma, costanti numeriche e operazioni aritmetiche e che inv sia un invariante di ciclo per W scelto in modo tale che:

1. $inv \Rightarrow E \geqslant 0$
2. $\vdash_{tot} \ \{inv \wedge B \wedge E = k\}C\{inv \wedge E < k\}$ supponendo che il valore in E decresce ogni volta che si esegue C, ma rimanendo sempre positivo

si ha che:

- B vale, quindi si può applicare la condizione 2 per concludere che una singola esecuzione del comando C porterà ad uno stato in cui vale inv e il valore di E è diminuito. In questo nuovo stato:
  - B vale [...]
  - B non vale, quindi termino l'esecuzione
- B non vale, quindi l'intero comando iterativo non viene eseguito e quindi termina

Nota: nello stato iniziale si ha $E = k$ ed eseguendo il comando C questo valore diminuisce restando positivo e raggiungerà un valore minimo. Di conseguenza o B diventa false oppure il valore di E decrementerà arrivando ad un valore negativo, e questo rappresenta una contraddizione quindi l'esecuzione, in entrambi i casi, terminerà.

Proprietà generali

Correttezza: tutto ciò che si può derivare attraverso l'apparato deduttivo è effettivamente vero, ovvero $\vdash \Rightarrow \models$
Completezza: l'apparato deduttivo è in grado di derivare tutte le formule/triple vere, ovvero $\models \Rightarrow \vdash$

3 Logica proposizionale e insiemi di stati

Relazione tra gli insiemi di stati associati alle formule:

- $m(\neg p) = \Sigma \setminus m(p)$ complemento insiemistico di p
- $m(p \vee q) = m(p) \cup m(q)$
- $m(p \wedge q) = m(p) \cap m(q)$
- operatore logico
  - $p \Rightarrow q \equiv \neg p \vee q$
  - $m(p \Rightarrow q) = m(\neg p) \cup m(q)$
- relazione tra formule
  - se $p \Rightarrow q$ allora $m(p) \subseteq m(q)$, quindi q è più debole di p

3.1 Scelta della precondizione migliore

Vengono definite una serie di notazioni:

- V insieme delle variabili di un comando C per cui si vuole trovare la precondizione migliore
- $\Sigma = \{\sigma \ tc: \ \sigma : V \to \mathbb{Z}\}$ insieme degli stati della memoria
- Π insieme delle formule in V
- $\sigma \vDash p$, ovvero p è vera in σ, con $\sigma \in \Sigma, \ p \in \Pi$, e si ha $\vDash \subseteq \Sigma \times \Pi$
- $t(\sigma) = \{p \in \Pi \ tc: \ \sigma \vDash p\}$ funzione che indica l'insieme delle formule vere in σ
- $m(p) = \{\sigma \in \Sigma \ tc: \ \sigma \vDash p\}$ funzione che indica l'insieme di stati che soddisfano la formula p
- se $S \subseteq \Sigma$ sottoinsieme di stati, $F \subseteq \Pi$ sottoinsieme di formule, allora
  - $t(S) = \{p \in \Pi \ tc: \ \forall s \in S: \ s \vDash p\} = \bigcap_{s \in S} t(s)$
  - $m(F) = \{s \in \Sigma \ tc: \ \forall p \in F: \ s \vDash p\} = \bigcap_{p \in F} m(p)$

Si vuole cercare la precondizione più debole p tale che $\vDash \{p\}C\{q\}$, dove p corrisponde al più grande insieme di stati tale che la tripla è valida, ovvero p determina l'insieme di tutti gli stati che come stati iniziali garantiscono il raggiungimento di q all'esecuzione di C.
Questa precondizione più debole è chiamata **weakest precondition** per $C\{q\}$ ed è indicata con $wp(C, q)$.

Teorema 1. $\vDash \{p\}C\{q\}$ *SSE* $p \Rightarrow wp(C, q)$*, cioè se e solo se* p *è più forte di* $wp(C, q)$ *che rappresenta l'insieme di tutti gli stati che partendo da* p *ed eseguendo* C *raggiungono* q*.*

Regole di calcolo

- assegnamento: $\{q[E/x]\}\ x := \ E\{q\}$, dove $q[E/x]$ è la precondizione più debole e si sostituisce in q ogni occorrenza di x con l'espressione E
- sequenza: $wp((C_1; C_2), q) = wp(C_1,\ wp(C_2, q))$
- scelta: $C:\ if\ B\ then\ C_1\ else\ C_2\ endif$

$$wp(P, q) \equiv (B \wedge wp(C, q))\ \vee\ (\neg B \wedge wp(D, q))$$

- iterazione: $while\ B\ do\ C\ endwhile$

$$wp(W, q) \equiv (\neg B \wedge q) \vee (B \wedge wp((C; wp(W, q)))$$

Nota: non è sempre possibile stabilire qual è la precondizione più debole.

4 Calcolo di sistemi comunicanti - CCS

4.1 Sistemi di transizioni etichettati - LTS

Definizione 5. *Un **sistema di transizioni etichettato** è definito come $LTS = (S, Atc, T, s_0)$, dove:*

- *S insieme di stati*
- *Act insieme di azioni - transizioni tra stati*
- *$T = \{(s, a, s')\ tc:\ s, s' \in S \wedge a \in Act\},\ T \subseteq S \times Act \times S$*
- *s_0 stato iniziale*

Definizione 6. *Un'**azione** $s \xrightarrow{a} s'$ può essere estesa ad $w \in Act$, quindi $s \xrightarrow{w} s'$ SSE:*

- *se $w = \varepsilon \Rightarrow s = s'$*
- *se $w = a \cdot x \Rightarrow$ se $s \xrightarrow{a} s'' \xrightarrow{x} s'$ con $a \in Act,\ x \in Act^*$*

Si assume che:

- $s \longrightarrow s' \Leftrightarrow\ \exists\ a \in Act\ tc:\ s \xrightarrow{a} s' \in T \Longrightarrow \bigcup_{a \in Act} \xrightarrow{a} \subseteq S \times S \qquad (\rightarrow)$
- $s \longrightarrow^* s' \Leftrightarrow\ \exists\ a \in Act^*\ tc:\ s \xrightarrow{w} s' \in T \Longrightarrow \bigcup_{w \in Act^*} \xrightarrow{a} \subseteq S \times S \qquad (\rightarrow^*)$

4.1.1 Proprietà delle relazioni

Definizione 7. *Una **relazione di equivalenza** è una relazione $R \subseteq X \times X$ binaria su X tale che valgono le seguenti proprietà:*

- *R è riflessiva $\Leftrightarrow \forall x \in X,\ (x,x) \in R$ oppure xRx, ogni elemento è in relazione con se stesso*
- *R è simmetrica $Leftrightarrow(x,y) \in R \Rightarrow (y,x) \in R,\ \forall x,y \in X$*
- *R è transitiva se $(x,y) \in R \wedge (y,z) \in R \Rightarrow (x,z) \in R$*

Siano $R, R', R'' \subseteq X \times X$ relazioni binarie su X. R' è **chiusura** riflessiva/simmetrica/transitiva di R SSE:

1. $R \subseteq R'$
2. R' è riflessiva/simmetrica/transitiva (rispettivamente)
3. R' è la più piccola relazione che soddisfa 1. e 2., cioè $\forall R''$ se $R \subseteq R'' \wedge R''$ è riflessiva/simmetrica/transitiva (rispettivamente) allora $R' \subseteq R''$

4.2 CSS puro

Definizione 8. *Un **CCS** è definito come $(K, A, \bar{A})$, dove:*

- *K insieme di nomi di processi*
- *A insieme di nomi di azioni*
- *$\bar{A} = \{\bar{a}\ tc:\ a \in A\}$ insieme di nomi di coazioni, e $\forall a \in A\ \exists\ \bar{a} \in \bar{A}$ sapendo che $Act = A \cup \bar{A} \cup \{\tau\},\ \tau \notin A$*

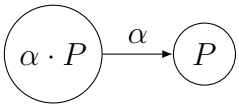
I **processi CCS** (P_{CCS}) sono espressioni con linguaggi CCS.

Un **sistema CCS** è un insieme di processi $p \in K$ specificati da espressioni CCS.

4.2.1 Operazioni del CCS

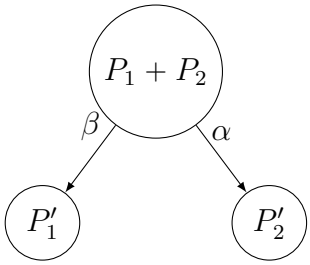
Un processo CCS può essere **nullo** NIL:

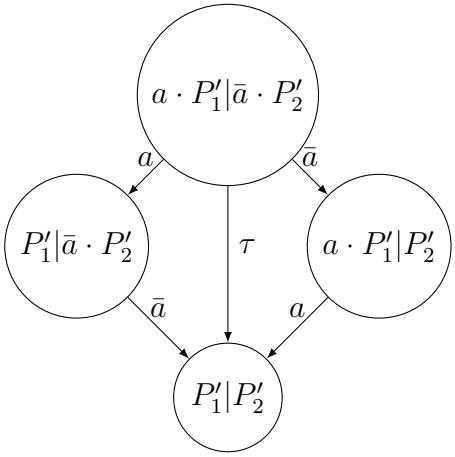
Un processo CCS può essere **prefisso** $\alpha \cdot P$:



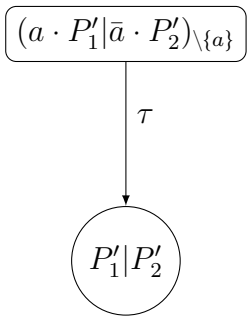
Un processo CCS può essere somma $P_1 + P_2$. Questo rappresenta una **scelta**:

Definiamo $P_1|P_2$ **composizione parallela** di processi:





Sia $L \subseteq A$, $Act = A \cup \bar{A} \cup \{\tau\}$, $P \in$ processi CCS. $P_{\backslash L}$ è definito come **restrizione**, ovvero il processo P non può interagire con il suo ambiente con azioni in $L \cup \bar{L}$, ma le azioni $L \cup \bar{L}$ sono locali a P:



Si definisce $f : Act \to Act$ **funzione di rietichettatura** tale che
$f(\tau) = \tau \wedge f(\bar{a}) = \overline{f(a)}$.

La **precedenza rispetto gli operatori** è l'ordine in cui vanno eseguiti gli operatori è il seguente (da sinistra a destra): $\backslash L$, $[f]$, $\alpha \cdot P$, $|$, $+$.
A questo punto Milner si chiede se data un'implementazione, questa soddisfa una certa speficia, ovvero si pone il seguente problema:

implementazione $\models$ **specifica** ?

Prendiamo ad esempio l'implementazione
Uni = $(\text{coin} \cdot \overline{\text{cafe}} \cdot \text{M} | \overline{\text{lez}} \cdot \overline{\text{coin}} \cdot \text{cafe} \cdot \text{LP}_{\backslash \{coin, cafe\}}$ e la specifica $S = \overline{\text{lez}} \cdot \text{S}$ e ci chiediamo se possiamo vedere Uni come un'implementazione di S oppure no.
Requisiti:

- si vuole avere una **relazione di equivalenza** tra processi del CCS, ovvero una relazione $R \subseteq P_{CCS} \times P_{CCS}$ con R che è riflessiva, simmetrica e transitiva
- si vuole astrarre dagli stati e considerare come azioni fondamentali le azioni Act
- si vuole astrarre dalle sincronizzazioni interne, ovvero si vuole astrarre dalle τ
- si vuole astrarre rispetto al non determinismo
- R dev'essere una **congruenza** rispetto agli operatori del CCS

R relazione di equivalenza è una **congruenza** rispetto agli operatori del CCS SSE: $\forall p, q \in P_{CCS}$ e $\forall c[\cdot]$ contesto CCS e se pRq allora $c[p]\ R\ c[q]$

4.3 Equivalenza rispetto alle tracce

Data R una congruenza, se $M_1\ R\ M_2$ allora significa che $(M_1|LP)_{\setminus\ \{\text{coin,cafe}\}}\ R\ (M_2|LP)_{\setminus\ \{\text{coin,cafe}\}}$ quindi non mi accorgo della differenza di comportamento tra M_1 e M_2 perchè questi sono congruenti. Viene quindi introdotta la nozione di **equivalenza rispetto alle tracce**.

Definizione 9. *Sia $p \in P_{CCS}$ e si considerano le **tracce** di p:*
$tracce(p) = \{w \in Act^*\ tc:\ \exists\ p' \in P_{CCS}\ p \xrightarrow{w}\ p'\}$.

Definizione 10. *Dati $P_1, P_2 \in P_{CCS}$, P_1 è **equivalente rispetto alle tracce** a P_2 SSE: $tracce(P_1) = tracce(P_2)$, e si denota con $P_1 \sim^T P_2$.*

5 Bisimulazione

Definizione 11. *Considero una relazione binaria $R \subseteq P_{CCS} \times P_{CCS}$. R è una **relazione di bisimulazione forte** SSE: $\forall p, q \in P_{CCS}$ pRq vale che:*

1. $\forall \alpha \in Act$ *se* $p \xrightarrow{\alpha}\ p' \Rightarrow \exists\ q' \in P_{CCS}\ tc:\ q \xrightarrow{\alpha}\ q' \wedge p'Rq'$
2. $\forall \alpha \in Act$ *se* $q \xrightarrow{\alpha}\ q' \Rightarrow \exists\ p' \in p_{CCS}\ tc:\ p \xrightarrow{\alpha}\ p' \wedge p'Rq'$

Due processi p, q sono **fortemente bisimili** SSE $\exists\ R \subseteq P_{CCS} \times P_{CCS}$ relazione di bisimulazione forte tc: pRq, e si denota con $p \sim^{BIS} q$.

Teorema 2. *Presa $\sim^{BIS} \subseteq P_{CCS} \times P_{CCS}$ si può dimostrare che è riflessiva, simmetrica e transitiva, allora $\sim^{BIS}$ è una relazione di equivalenza.*

$p \sim^{BIS} q$ SSE $\forall \alpha \in Act$

- se $p \xrightarrow{\alpha}\ p' \Rightarrow \exists\ q'\ tc:\ q \xrightarrow{\alpha}\ q' \wedge p' \sim^{BIS} q'$
- se $q \xrightarrow{\alpha}\ q' \Rightarrow \exists\ p'\ tc:\ p \xrightarrow{\alpha}\ p' \wedge p' \sim^{BIS} q'$

Teorema 3. *Se* $p \sim^{BIS} q$ *allora* $p \sim^{T} q,\ \forall p, q \in P_{CCS}$.

La bisimulazione forte è anche una **congruenza** rispetto agli operatori del CCS.
Se $p, q \in P_{CCS}$ e $p \sim^{BIS} q$ allora valgono:

- $\alpha \cdot p \sim^{BIS} \alpha \cdot q,\ \forall \alpha \in Act$
- $p + r \sim^{BIS} q + r \wedge r + p \sim^{BIS} r + q,\ \forall r \in P_{CCS}$
- $p|r \sim^{BIS} q|r \wedge r|p \sim^{BIS} r|q,\ \forall r \in P_{CCS}$
- $p[f] \sim^{BIS} q[f],\ \forall f : Act \to Act\ tc:\ f(\tau) = \tau,\ f(\bar{A}) = \overline{f(A)}$ funzione di rietichettatura
- $p_{\backslash L} \sim^{BIS} q_{\backslash L},\ \forall L \subseteq A$

Definizione 12. $p \stackrel{\alpha}{\Longrightarrow} p'$ *è definita* ***relazione di transizione debole****, con* $\alpha \in Act$ *SSE:*

- *se* $\alpha = \tau$ *allora* $p \stackrel{\tau}{\longrightarrow}{}^{*} p'$, *con* $\to^{*}$ *che rappresenta una qualsiasi sequenza anche nulla di* τ
- *se* $\alpha \in A \cup \bar{A}$ *allora* $p \stackrel{\tau}{\longrightarrow}{}^{*} \stackrel{\alpha}{\longrightarrow} \stackrel{\tau}{\longrightarrow}{}^{*} p'$

La relazione di transizione debole può essere estesa anche a sequenze di Act. $\forall w \in Act^{*}\ p \stackrel{w}{\longrightarrow} p'$ SSE vale 1. oppure 2.:

1. $w = \varepsilon \vee w = \tau^{*}$ se $p \stackrel{\tau}{\longrightarrow}{}^{*} p'$
2. $w = a_1 ... a_n,\ a_i \in A \cup \bar{A},\ p \stackrel{a_1}{\Longrightarrow} ... \stackrel{a_n}{\Longrightarrow} p'$

Definizione 13. *Si possono dare due definizioni equivalenti di* ***equivalenza debole rispetto alle tracce****:*

- $P \approx^{T} Q$ *SSE* $tracce_{\Rightarrow}(P) = tracce_{\Rightarrow}(Q)$
- $\forall w \in (A \cup \bar{A})^{*}\ P \stackrel{w}{\Longrightarrow} \Leftrightarrow Q \stackrel{w}{\Longrightarrow}$

Definizione 14. $R \subseteq P_{CCS} \times P_{CCS}$ *è una* ***bisimulazione debole*** *SSE* $\forall p, q \in P_{CCS}$ pRq *vale che:* $\forall \alpha \in Act$

- *se* $p \stackrel{\alpha}{\longrightarrow} p'$ *allora* $\exists\ q'\ tc:\ q \stackrel{\alpha}{\Longrightarrow} q'\ tc:\ p' R\, q'$
- *se* $q \stackrel{\alpha}{\longrightarrow} q'$ *allora* $\exists\ p'\ tc:\ p \stackrel{\alpha}{\Longrightarrow} p'\ tc:\ p' R\, q'$

Due processi sono **debolmente bisimili** SSE esiste una relazione di bisimulazione debole R tc: pRq, ed è denotata con $p \approx^{BIS} q$.

L'operatore di **congruenza** è denotato con $\approx^{C}$ e vale che
$\approx^{C} \subseteq \approx^{BIS} \subseteq p_{CCS} \times P_{CCS}$, ovvero la congruenza è la più grande relzione contenuta in $\approx^{BIS}$.

Rispetto alla congruenza, per il CCS puro senza ricorsione, sono stati indotrodotti da Milner degli assioni che possono essere anche visti come regole di riscrittura.
Ax è un insieme finito di assiomi:

- Ax corretto: $Ax \vdash \ p = q \Rightarrow p \approx^C q$
- Ax completo: $p \approx^C q \Rightarrow Ax \vdash \ p = q$

Assiomi:

1. $p + (q + r) \approx^C (p + q) + r$ e $p|(q|r) \approx^C (p|q)|r$ (associatività)
2. $p + q \approx^C q + p$ e $p|q \approx^C q|p$ (commutatività)
3. $p + p \approx^C p$ ma $p|p \not\approx^C p$ (assorbimento 1)
4. $p + NIL \approx^C p$ e $p|NIL \approx^C p$ (assorbimento 2)
5. $p + \tau \cdot p \approx^C \tau \cdot p$ (τ all'inizio non può essere semplificato)
6. $\mu \cdot \tau \cdot p \approx^C \mu \cdot p, \ \forall \mu \in Act$ (τ dentro una sequenza può essere semplificato)
7. $\mu \cdot (p + \tau \cdot q) \approx^C \mu \cdot (p + \tau \cdot q) + \mu \cdot q$
8. $p = \sum_i \alpha_i \cdot p_i, \quad q = \sum_j \beta_j \cdot q_j, \quad \alpha, \beta \in Act$
   $p|q \approx^C \ \sum_i \alpha_i(p_i|q) + \sum_j \beta_j(p|q_j) + \sum_{\alpha_i = \beta_j} \tau \cdot (p_i|q_j)$ (teorema di espansione di Milner)
9. $p[f] \approx^C \ \sum_i f(\alpha_i)(p_i[f]), \ \forall f$ funzione di rietichettatura
10. $p_{\backslash L} \approx^C \ \sum_{\alpha_i, \overline{\alpha_i} \notin L} \alpha_i(p_{\backslash L}), \ \forall L \subseteq A$

$\approx^C$ è più restrittiva di $\approx^{BIS}$, ma è la più grande contentuta in $\approx^{BIS}$.

5.1 Bisimulazione debole come gioco

Gioco $G(p, q)$ con 2 **giocatori** p, q:

- **attaccante**, cerca di dimostrare che $p \not\approx^{BIS} q$
- **difensore**, cerca di dimostrare che $p \approx^{BIS}$

Un gioco $G(p, q)$ è fatto da più partite. Una **partita** è una sequenza finita o infinita di configurazioni $(p, q) = (p_0, q_0)...(p_i, q_i)$ in cui ad ogni mano si passa dalla configurazione corrente alla configurazione successiva (p_{i+1}, q_{i+1}).

Le **regole** del gioco sono:

- l'attaccante deve scegliere uno dei due processi e fa una mossa $\xrightarrow{\alpha}, \ \alpha \in Act$
- il difensore deve rispondere sull'altro processo con $\stackrel{\alpha}{\Longrightarrow}$

- in questo modo si decide una nuoba configurazione (p_{i+1}, q_{i+1})

Il **risultato** è:

- se un giorcatore non ha più mosse a disposizione, l'altro vince
- se la partita è infinita vince il difensore

Teorema 4. *Per ogni gioco $G(p,q)$ solo uno dei giocatori ha una strategia vincente.*

Teorema 5. *L'attaccante ha una strategia vincente per $G(p,q)$ SSE $p \not\approx^{BIS} q$. Il difensore ha una strategia vincente per $G(p,q)$ SSE $p \approx^{BIS} q$.*

5.2 Proprietà relazione di equivalenza

$\simeq \subseteq P_{CSS} \times P_{CCS}$: siano $p, q \in P_{CCS}$

- se $LTS(p) = LTS(q) \Rightarrow p \simeq q$
- si deve astrarre dagli stati e considerare solo le azioni
- se $p \simeq q \Rightarrow tracce(p) = tracce(q)$
- se $p \simeq q \Rightarrow p, q$ devono avere la stessa possibilità di generare deadlock con l'ambiente
- $\simeq$ deve essere una congruenza rispetto agli operatori del CCS

5.3 Proprietà di quivalenza forte rispetto alle tracce

$\sim^T$: $\forall p, q \in P_{CSS}$

- $LTS(p) = LTS(q) \Rightarrow p \simeq q$
- si deve astrarre dagli stati e considerare solo le azioni
- $p \sim^T q \Leftrightarrow tracce(p) = tracce(q)$
- è una cogruenza rispetto agli operatori del CCS
- non garantisce di preservare il deadlock nell'iterazione con l'ambiente

5.4 Proprietà della bisimulazione forte

$\sim^{BIS}$: $\forall p, q \in P_{CCS}$

- $LTS(p) = LTS(q) \Rightarrow p \sim^{BIS} q$
- si deve astrarre dagli stati e considerare solo le azioni
- $p \sim^{BIS} q \Leftrightarrow tracce(p) = tracce(q)$ e $p \sim^{BIS} q \Rightarrow p \sim^T q$, $\sim^{BIS} \subseteq \sim^T$
- è una cogruenza rispetto agli operatori del CCS
- preserva il deadlock nell'iterazione con l'ambiente
- è troppo restrittiva perchè non astrae dalle τ

5.5 Proprietà di equivalenza debole rispetto alle tracce

$\approx^T$: $\forall p, q \in P_{CSS},\ \sim^T$ è più restrittiva di $\approx^T$, ovvero $p \sim^T q \Rightarrow p \approx^T q,\ \sim^T \subseteq \approx^T$

- $LTS(p) = LTS(q) \Rightarrow p \approx^T q$
- si deve astrarre dagli stati e considerare solo le azioni
- $p \sim^T q \Leftrightarrow tracce(p) = tracce(q)$
- è una cogruenza rispetto agli operatori del CCS
- non garantisce di preservare il deadlock nell'iterazione con l'ambiente

5.6 Proprietà della bisimulazione debole

$\approx^{BIS}$: $\forall p, q \in P_{CCS},\ \sim^{BIS}$ è più restrittiva di $\approx^{BIS}$, ovvero
$p \sim^{BIS} q \Rightarrow p \approx^{BIS} q,\ \sim^{BIS} \subseteq \approx^{BIS}$

$\sim^{BIS}$ / $\approx^{BIS}$ è più restrittiva di $\sim^T$ / $\approx^T$, ovvero

- $p \sim^{BIS} q \Rightarrow p \sim^T q$
- $p \approx^{BIS} q \Rightarrow p \approx^T q$
- $\Rightarrow \sim^{BIS} \subseteq \sim^T$ e $\approx^{BIS} \subseteq \approx^T$
- $\approx^{BIS}$ è la più grande equivalenza di bisimulazione debole
- preserva la possibile di generare/non generare deadlock nell'iterazione con l'ambiente
- astrae da azioni non osservabili (τ) e dai cicli non osservabili (τ loop)

$p \in P_{CCS}$ è un processo deterministico SSE $\forall x \in Act$, se $p \xrightarrow{x} p'$ e $p \xrightarrow{x} p''$ allora $p' = p''$.

Proposizione 1. *siano $p, q \in P_{CCS}$. Se p, q sono deterministici e $p \sim^T q$ allora $p \sim^{BIS} q$.*

Nota: $R \subseteq P_{CCS} \times P_{CCS}$ relazione di equivalenza è una congruenza se $\forall c[\cdot]$ contesto CCS, $pRq \Rightarrow c[p]\ R\ c[q],\ p, q \in P_{CCS}$.

Teorema 6. *$\forall p, q \in P_{CCS}$ se $p \approx^{BIS} q$ allora*

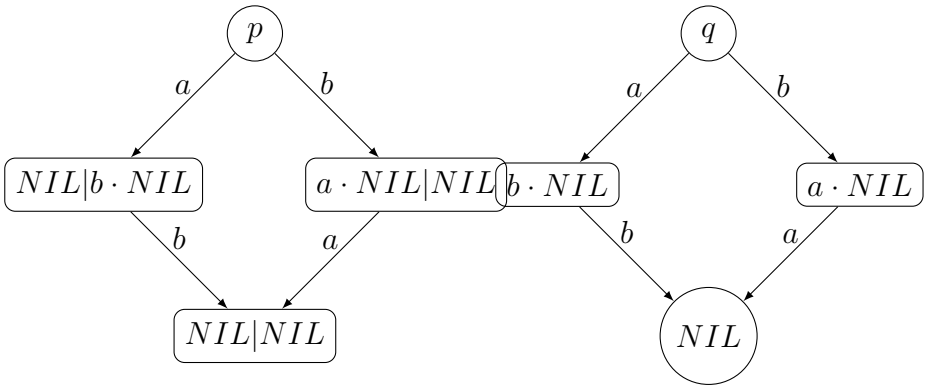
- $\alpha \cdot p \approx^{BIS} \alpha \cdot q,\ \forall \alpha \in Act$ *(prefisso)*
- $p|r \approx^{BIS} q|r\ \wedge\ r|p \approx^{BIS} r|q,\ \forall r \in Act$ *(composizione parallela)*
- $p[f] \approx^{BIS} q[f],\ \forall f$ *funzione di rietichettatura (rietichettatura)*
- $P_{\setminus L} \approx^{BIS} q_{\setminus L},\ \forall L \subseteq A$ *(restrizione)*
- $\tau \cdot a \cdot NIL \approx^{BIS} a \cdot NIL, ma \tau \cdot a \cdot NIL + b \cdot NIL \not\approx^{BIS} a \cdot NIL + b \cdot NIL$

allora $\approx^{BIS}$ non è una confruenza rispetto al CCS.

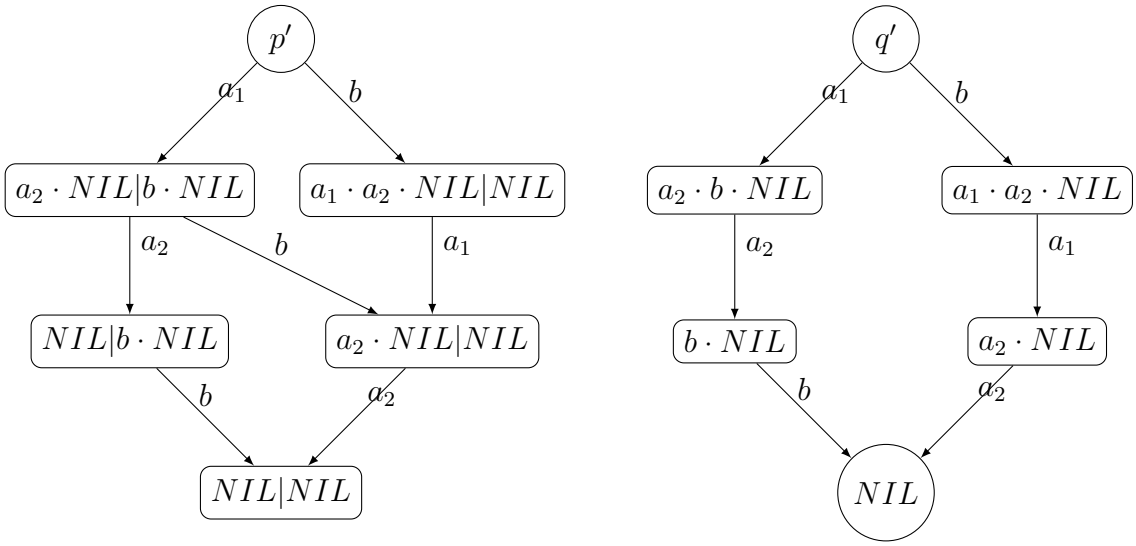
Semantica composizione parallela

È di estrema importanza l'atomicità delle azioni.

Esempio 1. $p = a \cdot NIL|b \cdot NIL \quad \sim^{BIS} \quad q = a \cdot b \cdot NIL + b \cdot a \cdot NIL$



Se si sostituisce $a = a_1 \cdot a_2$ non considerando più a come azione atomica, il risultato cambia:
$p' = p_{[a \leftarrow a_1 \cdot a_2]} = a_1 \cdot a_2 \cdot NIL|b \cdot NIL, \quad \not\sim^T \quad q' = q_{[a \leftarrow a_1 \cdot a_2]} = a_1 \cdot a_2 \cdot b \cdot NIL + b \cdot a_1 \cdot a_2 \cdot NIL$
$a_1 \cdot b \cdot a_2 \in tracce(p')$ ma $a_1 \cdot b \cdot a_2 \notin tracce(q')$ quindi $tracce(p') \neq tracce(q')$



6 Reti di Petri

6.1 Sistemi elementari

Definizione 15. *$N = (B, E, F)$ è una **rete elementare** SSE:*

- *B è un insieme finito di condizioni/stati locali rappresentate da $\circ$*
- *E è un insieme finito di eventi/transizioni locali rappresentate da $\square$*

tale che $B \cap E = \varnothing \ \wedge \ B \cup E \neq \varnothing$ *insiemi disgiunti non vuoti*

- $F \subseteq (B \times E) \cup (E \times B)$ *relazione di flusso rappresentata da* $\rightarrow$ *la precondizione dell'evento, e da* $\leftarrow$ *la postcondizione dell'evento*

tale che $dom(F) \cup cod(F) = B \cup E$.

Sia $x \in B \cup E = X$

- ${}^\bullet x = \{y \in X \ tc: \ (y, x) \in F\}$ **pre-elementi** di x
- $x^\bullet = \{y \in X \ tc: \ (x, y) \in F\}$ **post-elementi** di x

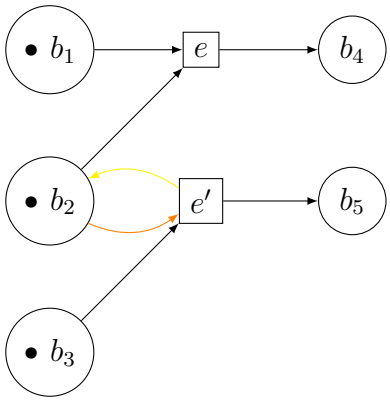
Sia $A \subseteq B \cup E$, allora ${}^\bullet A = \bigcup_{x \in A} {}^\bullet x$ e $A^\bullet = \bigcup_{x \in A} x^\bullet$ località e dualità tra stati e transizioni. Ogni evento è collegato solo alle sue pre e post condizioni.

Il **caso** è un insieme di condizioni $c \subseteq B$ che rappresentano l'insieme di condizioni vere di una certa configurazione.

6.2 Regola di scatto

Definizione 16. *Sia* $N = (B, E, F)$ *una rete elementare e* $c \subseteq B$ *caso. L'evento* $e \in E$ *è* ***abilitato*** *in* c *SSE* ${}^\bullet e \subseteq c \ \wedge \ e^\bullet \cap c = \varnothing$, *e si denota con* $c[e>$. *Questo indica che quando un evento è abilitato deve avere tutte le sue precondizioni vere e le sue postcondizioni false.*

Esempio 2. *La freccia gialla* $c[e'>$ *non è abilitato in* c *in quanto la postcondizione dev'essere per forza falsa.*
La freccia arancione $c[e'>$ *è abilitato perchè si hanno precondizioni vere e postcondizioni false, ma si crea conflitto perchè lo scatto di* e' *disabilita* e *e viceversa. Se si considera il sottosistema formato solo da* b_2, e' *e le due frecce colorate, questo non potrà mai scattare.*



Definizione 17 (regola di scatto)**.** *Se* $c[e>$ *allora quando* e ***occorre*** *in* c *si genera un nuovo caso* c' *denotato con* $c[e > c'$ *tale che* $c' = (c -^\bullet e) \cup e^\bullet$.

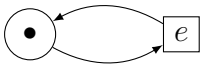
Data una rete elementare $N = (B, E, F)$, $U \subseteq E$ $c_1, c_2 \subseteq B$ allora

- U è un **insieme di eventi indipendenti** SSE $\forall e_1, e_2 \in U \ tc:$
  $e_1 \neq e_2 \Rightarrow ({}^\bullet e_1 \cup e_1^\bullet) \cap ({}^\bullet e_2 \cup e_2^\bullet) = \varnothing$

- U è un **passo abilitato in** c_1 SSE U è un insieme di eventi indipendenti $\wedge\ \forall e \in U :\ c_1[e >$
- U è un **passo da** c_1 **a** c_2 SSE $c_1[U >\ \wedge\ c_2 = (c_1 -^\bullet U) \cup U^\bullet$

Definizione 18. *$N = (B, E, F)$ è una **rete pura** SSE $\forall e \in E\ \ {}^\bullet e \cap e^\bullet = \varnothing$, ovvero se le pre e le post condizioni non sono lo stesso stato.*

Esempio 3. *in questo caso l'evento e non scatterà mai perchè, sebbene le sue precondizioni siano vere, sono vere anche le sue postcondizioni, non rispettando così la regola di scatto:*



Definizione 19. *$N = (B, E, F)$ è una **rete semplice** SSE $\forall x, y \in B \cup E$ ${}^\bullet x =^\bullet y\ \wedge\ x^\bullet = y^\bullet\ \Rightarrow x = y$*

Definizione 20. *Un **sistema elementare** $\Sigma = (B, E, F;\ c_{in})$ è definito da una rete $N = (B, E, F)$ da $c_{in} \subseteq B$ caso iniziale.*

Definizione 21. *L'**insieme dei casi raggiungibili** C_Σ del sistema elementare Σ è il più piccolo sottoinsieme di 2^B tale che:*

- $c_{in} \in C_\Sigma$
- $c \in C_\Sigma,\ U \subseteq E,\ c' \subseteq B$ *se* $c[U > c' \Rightarrow c' \in C_\Sigma$

Definizione 22. *U_Σ è l'**insieme dei passi** di Σ :*
$U_\Sigma = \{U \subseteq E\ tc :\ \exists\ c, c' \in C_\Sigma :\ c[U > c'\}$

6.3 Comportamento dei sistemi elementari

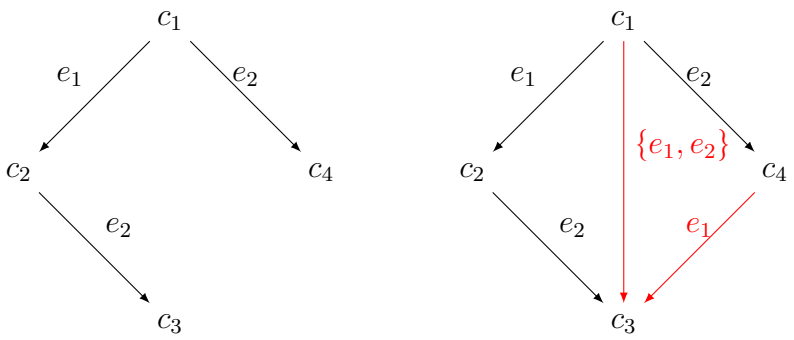
- **comportamento sequenziale** (interleaving) $c_{in}[e_1 > c_1[e_2 > ...[e_n > c_n$ $c_{in}[e_1 e_2 ... e_n > c_n$
- **comportamento non sequenziale/comportamento concorrente** (step semantics) $c_{in}[U_1 > c_1[U_2 > ...[U_n > c_n\ c_{in}[U_1 U_2 ... U_n > c_n$

Dato $\Sigma = (B, E, F;\ c_{in})$ sistema elementare si ha il:

- **grafo dei casi raggiungibili**: $CG_\Sigma = (C_\Sigma, U_\Sigma, A, c_{in})$, con:
  - C_Σ insieme dei casi raggiungibili
  - U_Σ insieme dei (possibili) passi
  - $A = \{(c, U, c')\ tc :\ c, c' \in C_\Sigma,\ U \subseteq U_\Sigma :\ c[U > c'\}$ insieme degli archi etichettati, con $A \subseteq C_\Sigma \times U_\Sigma \times C_\Sigma$
- **grafo dei casi sequenziale**: $SCG_\Sigma = (C_\Sigma, E, A, c_{in})$, con:
  - C_Σ insieme dei casi raggiungibili
  - E insieme degli eventi
  - $A = \{(c, e, c')\ tc :\ c, c' \in C_\Sigma,\ e \in E :\ c[e > c'\}$ insieme degli archi etichettati, con $A \subseteq C_\Sigma \times E \times C_\Sigma$

Dyamond property

Dato $\Sigma = (B, E, F;\ c_{in})$ un sistema elementare e SCG_Σ grafo dei casi sequenziale, siano $e_1, e_2 \in E$ e $c_1, c_2, c_3, c_4 \in C_\Sigma$ allora vale $({}^\bullet e_1 \cup e_1^\bullet) \cap ({}^\bullet e_2 \cup e_2^\bullet) = \varnothing$, ovvero:



Dim. *Si vuole dimostrare che esistono gli archi rossi, ovvero si vuole dimostrare che* $\{e_1, e_2\}$ *sono indipendenti e si possono far scattare in un passo.*
$\{e_1, e_2\}$ *sono indipendenti, quindi bisogna dimostrare che* $({}^\bullet e_1 \cup e_1^\bullet) \cap ({}^\bullet e_2 \cup e_2^\bullet) = \varnothing$
Sapendo che c_1 *abilita sia* e_1 *che* e_2 *si ha che:*

- $c_1[e_1 > \Rightarrow\ {}^\bullet e_1 \subseteq c_1\ \wedge\ e_1^\bullet \cap c_1 = \varnothing\ \Rightarrow\ {}^\bullet e_1 \cap e_2^\bullet = \varnothing$
- $c_1[e_2 > \Rightarrow\ {}^\bullet e_2 \subseteq c_1\ \wedge\ e_2^\bullet \cap c_1 = \varnothing\ \Rightarrow\ {}^\bullet e_2 \cap e_1^\bullet = \varnothing$

Quindi si ha $c_1[e_1 > c_2[e_2 >$
$\Rightarrow\ {}^\bullet e_1 \subseteq c_1 \wedge e_1^\bullet \cap c_1 = \varnothing\ \Rightarrow\ {}^\bullet e_1 \cap\ {}^\bullet e_2 = \varnothing$
$\Rightarrow\ {}^\bullet e_2 \subseteq c_2 \wedge e_2^\bullet \cap c_2 = \varnothing\ \Rightarrow\ e_1^\bullet \cap e_2^\bullet = \varnothing$
$\Longrightarrow\ ({}^\bullet e_1 \cup e_1^\bullet) \cap ({}^\bullet e_2 \cup e_2^\bullet) = \varnothing$
Questo significa che $e_1,\ e_2$ *possono scattare in un unico passo perchè sono indipendenti ed entrambi abilitati in* c_1.

Definizione 23. *Date due funzioni biunivoche* $\alpha : S_1 \to S_2,\ \beta : E_1 \to E_2$ *allora* $\langle \alpha, \beta \rangle : A_1 \to A_2$ *con* $A_1 = (S_1, E_1, T_1, s_{01}),\ A_2 = (S_2, E_2, T_2, s_{02})$. *Questo è un* ***isomorfismo*** *SSE:*

- $\alpha(s_{01}) = s_{02}$
- $\forall s, s' \in S_1,\ \forall e \in E_1 :\ (s, e, s') \in T_1\ SSE\ (\alpha(s), \beta(e), \alpha(s')) \in T_2$

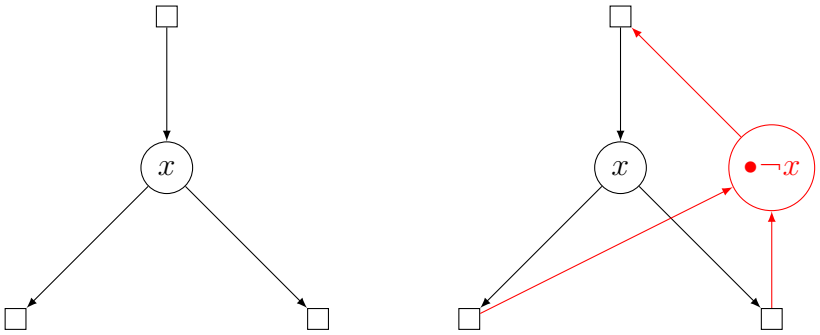
Due sistemi sono **equivalenti** se hanno grafo dei casi isomorfo.

Problema della sintesi

Dato $LTS = A = (S, E, T, s_0)$ un sistema di transazioni etichettato, ci si chiede se $\exists\ \Sigma$ sistema elementare tale che il grafo dei casi CG_Σ sia isomorfo ad A. Se questo sistema elementare esiste, lo si vuole costruire.
Questo problema è stato risolto con la *teoria delle regioni*. Inoltre A dovrà soddisfare la dyamond property.

Definizione 24. *Sia* $\Sigma = (B, E, F;\ c_{in}),\ e \in E,\ c \in C_\Sigma$. (e, c) *è un* ***contatto*** *SSE* ${}^\bullet e \subseteq c\ \wedge\ e^\bullet \cap c \neq \varnothing$, *ovvero le precondizioni sono tutte vere ma non tutte le postcondizioni sono false.*
Un $\Sigma = (B, E, F;\ c_{in})$ *è* ***senza contatti*** *SSE* $\forall e \in E,\ \forall c \in C_\Sigma\ {}^\bullet e \subseteq c$
$\Rightarrow e^\bullet \cap c = \varnothing$, *ovvero tutte le precondizioni sono vere e tutte le postcondizioni sono false.*

Si può trasformare un Σ con contatti in un Σ' senza contatti senza modificarne il comportamento semplicemente aggiungendo la condizione complemento di ogni condizione - Σ e Σ':



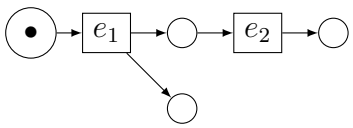
Se il sistema è senza contatti si può usare la regola di scatto semplificata: $c[e > \text{SSE}\ {}^\bullet e \subseteq c$

6.4 Situazioni fondamentali

Sia $\Sigma = (B, E, F;\ c_{in})$ un sistema elementare, $c \in C_\Sigma,\ e_1, e_2 \in E$. e_2 dipende causalmente da e_1.

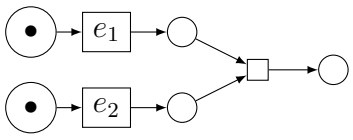
Sequenza

e_1, e_2 sono in **sequenza** in c SSE $c[e_1 > \ \wedge\ \neg c[e_2 > \ \wedge\ c[e_1 e_2 >$



Concorrenza

e_1, e_2 sono **concorrenti** in c SSE $c[\{e_1 e_2\} >$



Conflitto

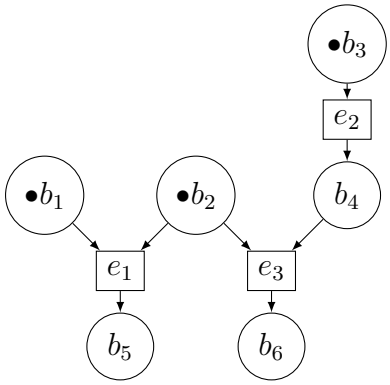
Si verifica un **conflitto** SSE $c[e_1 > \ \wedge c[e_2 > \ \wedge \neg c[\{e_1 e_2\} >$.

Figura 2: Esempio di conflitto in avanti (a sinistra) e all'indietro (a destra).

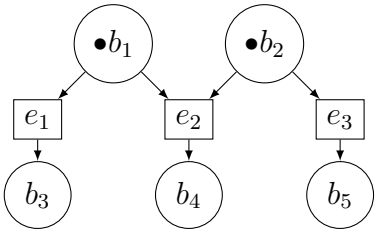
Confusione

Si parla di **confusione** quando il conflitto e la concorrenza interferiscono l'uno con l'altro. In particolare esistono due tipi di concorrenza:

Asimmetrica:



Simmetrica:



Definizione 25. *Siano* $N = (B, E, F)$, $N_1 = (B_1, E_1, F_1)$ *reti elementari.* $N1$ *è* ***sottorete*** *di* N *SSE:*

- $B_1 \subseteq B$
- $E_1 \subseteq E$
- $F_1 = F \cap [(B_1 \times E_1) \cup (E_1 \times B_1)]$

N_1 *è la* ***sottorete genereata da*** B_1 *SSE:*

- $B_1 \subseteq B$
- $E_1 = {}^\bullet B_1 \cup B_1^\bullet$
- $F_1 = F \cap [(B_1 \times E_1) \cup (E_1 \times B_1)]$

*N_1 è la **sottorete generata da** E_1 SSE:*

- $B_1 = {}^\bullet E_1 \cup E_1^\bullet$
- $E_1 \subseteq E$
- $F_1 = F \cap [(B_1 \times E_1) \cup (E_1 \times B_1)]$

6.5 Problema della mutua esclusione

Se e_1 scatta si sa che non potrà e_4 perchè entrambi usano la risorsa b_4. Questo comporta il fatto che non si avranno mai marcati contemporaneamente gli stati b_2, b_7 perchè e_1 e e_4 sono in conflitto. Gli eventi e_3, e_4 sono concorrenti. Si genera, dunque, una situazione di confusione (non si può dire se si è risolto o meno il conflitto). b_4 è conteso tra due processi e non c'è niente che dice come viene deciso se scatterà e_1 oppure e_4.

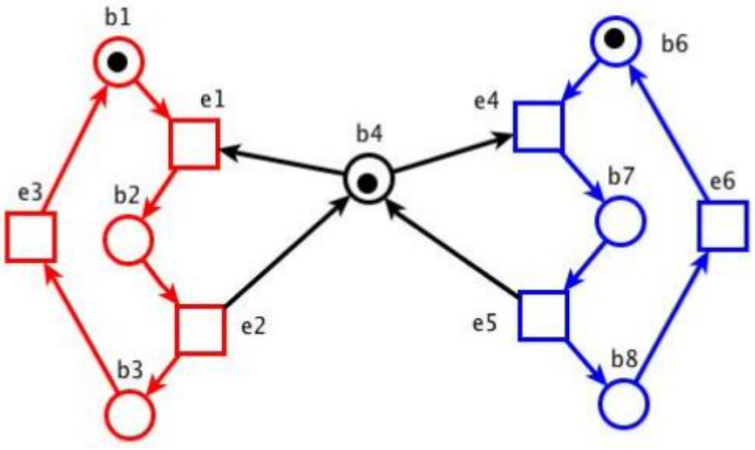


Figura 3: Esempio di mutua esclusione.

7 Semantiche ad ordini parziali

Definizione 26. *Sono dette **reti causali** (o reti di occorrenza senza conflitti e senza cicli delle reti) reti definite come $N = (B, E, F)$, che rispettano le seguenti condizioni:*

(i) $\forall b \in B : \ |{}^\bullet b| \leqslant 1 \ \wedge \ |b^\bullet| \leqslant 1$ *(no conflitti)*

(ii) $\forall x, y \in B \cup E : \ (x, y) \in F^+ \Rightarrow (y, x) \notin F^+$ *(no cicli)*

(iii) $\forall e \in E : \ \{x \in B \cup E \ tc : \ xF^*e\}$ *insieme finito, ovvero si ha un numero finito di pre-elementi per un dato evento*

La rete può essere **infinita**.

Associo ad una rete causale un **ordine parziale**: $(X, \leqslant) = (B \cup E, F^*)$. Siano $N = (B, E, F)$ una rete causale e $X = (B \cup E, \leqslant)$ un ordine parziale, allora:

- $x, y \in X:\ x, y$ elementi che occorrono della storia di X
- $x \leqslant y:\ x$ causa y
- $x\ li\ y:\ x \leqslant y\ \vee\ y \leqslant x.\quad x, y$ sono **casualmente dipendenti**
- $x\ co\ y:\ not(x < y)\ \wedge\ not(y < x).\quad x, y$ sono **causalmente indipendenti**
- $C \subseteq X$ è **co-set** SSE $\forall x, y \in C:\ x\ co\ y$
- $C \subseteq X$ è **taglio** SSE co-set è massimale
- $L \subseteq X$ è **li-set** SSE $\forall x, y \in L:\ x\ li\ y$
- $L \subseteq X$ è **linea** SSE li-set è massimale

Due elementi sono **ordinati** se c'è una sequenza di archi che li connette.

Definizione 27. *N è **K-densa** SSE ogni linea e ogni taglio si intersecano esattametne in un punto: $\forall h \in linee(N),\ \forall c \in tagli(N):\ |h \cap c| = 1$.*

Proposizione 2. *se N è finita $\Rightarrow$ N è K-densa.*

8 Processi non sequenziali

$\Sigma = (S, T, F;\ c_{in})$ è un sistema elementare senza contatto e finito tale che $S \cup T$ è finito.

Definizione 28. *$\langle N = (B, E, F); \phi\rangle$ è un **processo non sequenziale** di Σ SSE:*

- *(B, E, F) è una rete causale*
- *$\phi : B \cup E \to S \cup T$ è una mappa:*
  1. *$\phi(B) \subseteq S,\ \phi(E) \subseteq T$*
  2. *$\forall x_1, x_2 \in B \cup E:\ \phi(x_1) = \phi(x_2) \Rightarrow\ (x_1 \leqslant x_2) \vee (x_2 \leqslant x_1)$*
  3. *$\forall e \in E:\ \phi({}^\bullet e) = {}^\bullet\phi(e)\ \vee\ \phi(e^\bullet) = \phi(e)^\bullet$*
  4. *$\phi(\min(N)) = c_{in}$, con $\min(N) = \{x \in B \cup E\ tc:\ \nexists\ y : (y, x) \in F\}$ stati locali iniziali*

Proposizione 3. *se $\langle N = (B, E, F); \phi\rangle$ è un processo non sequenziale di Σ sistema elementare finito e senza contatti, allora $N = (B, E, F)$ è K-densa.*

$\forall K \subseteq B,\ K$ B-taglio di N tale che: K è finito e $\exists\ c \in C_\Sigma:\ \phi(K) = c$.

8.1 Processi ramificati

Definizione 29. *$N = (B, E, F)$ è una **rete di occorrenze** SSE:*

(i) $\forall b \in B : \ |{}^\bullet b| \leqslant 1$ *conflitti sono in avanti (forward)*

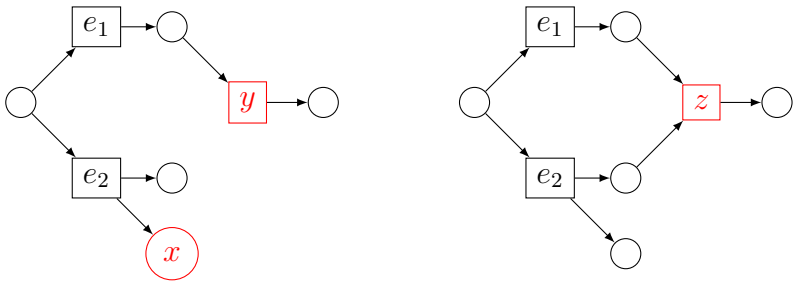
(ii) $\forall x, y \in B \cup E : \ (x, y) \in F^+ \Rightarrow (y, x) \notin F^+$ *non ci sono cicli*

(iii) $\forall e \in E : \ \{x \in B \cup E \ tc : \ xF^*e\}$ *è finito, ovvero si ha un numero finito di pre-elementi per un dato evento*

(iv) la relazione di conflitto $\#$ non è riflessiva

Definizione 30. *La relazione di **conflitto $\#$** è così definita: $\# \subseteq X \times X, \ X = B \cup E$ è definita come $x \# y$ SSE $\exists \ e_1, e_2 \in E : \ {}^\bullet e_1 \cap {}^\bullet e_2 \neq \varnothing \ \wedge \ e_1 \leqslant x \ \wedge \ e_2 \leqslant y$. Un processo ramificato rappresenta più di un comportamento del sistema.*

Gli elementi rossi sono gli elementi in conflitto tra loro: x, y in conflitto, ovvero $x \# y$ e z in conflitto con se stesso, ovvero $z \# z$.



Definizione 31. *Sia $\Sigma = (S, T, F; \ c_{in})$ un sistema elementare finito e senza contatti. $\langle N = (B, E, F); \phi \rangle$ è un **processo ramificato** di Σ SSE:*

- *(B, E, F) è una rete di occorrenze*
- *$\phi : B \cup E \to S \cup T$ è una mappa:*
  1. *$\phi(B) \subseteq S, \ \phi(E) \subseteq T$*
  2. *$\forall e_1, e_2 \in E : \ ({}^\bullet e_1 = {}^\bullet e_2 \ \wedge \ \phi(e_1) = \phi(e_2)) \Rightarrow e_1 = e_2$*
  3. *$\forall e \in E : \ \phi({}^\bullet e) = {}^\bullet \phi(e) \ \vee \ \phi(e^\bullet) = \phi(e)^\bullet$*
  4. *$\phi(\min(N)) = c_{in}$, con $\min(N) = \{x \in B \cup E \ tc : \ \nexists \ y : (y, x) \in F\}$ stati locali iniziali*

8.2 Unfolding

Definizione 32. *Sia $\Sigma = (B, E, F; \ c_{in})$ un sistema elementare finito e senza contatti e $\Pi_1 = \langle N_1; \phi_1 \rangle$, $\Pi_2 = \langle N_2; \phi_2 \rangle$ processi ramificati. Π_1 è **prefisso** di Π_2 SSE N_1 è sottorete di N_2 e $\phi_{2|N_1} = \phi_1$ (ϕ_2 ristretto ad N_1 dev'essere uguale a ϕ_1).*

Σ ammette un unico processo ramificato che è massimale rispetto alla relazione di prefisso tra processi. Tale processo massimale è chiamato **unfolding** di Σ, ed è denotato con $unf(\Sigma)$. L'unfolding è un oggetto unico che rappresenta tutti i possibili comportamenti del sistema, quindi può essere sia finito che infinito.
Un processo non sequenziale è un processo ramificato $\Pi = \langle N; \phi \rangle$ tale che N sia una rete causale senza conflitti. Tale processo è chiamato anche **corsa** o **run**.

9 Introduzione logiche temporali e model-checking

In questa sezione verranno spiegate le condizioni per la quale è possibile dire che un programma concorrente è corretto. Sul programma concorrente, a causa del fatto che esso non è detto che termini, ovvero non è detto che abbia uno stato finale, non si possono applicare le regole della logica di Hoare.

Definizione 33. *Vengono definiti* ***sistemi reattivi*** *quei sistemi che sono concorrenti, distribuiti e asincroni.*

Il problema è capire se un sistema reattivo è corretto oppure no. Questo lo si può capire attraverso un **criterio di correttezza** che è una formula logica che fa riferimento a stati futuri e non al singolo stato finale.
Si rappresenterà il sistema reale in un sistema di transizioni definendo i criteri per i quali si stabilisce che se una formula di correttezza è vera nel sistema di transizioni allora è vera anche nel sistema reale.

Definizione 34. *Un* ***sistema di transizioni*** *è definito da un insieme finito di stati Q e da un insieme di transizioni di stati $T \subseteq Q \times Q$, nel quale una transizione è identificata da un insieme di stati:*

$$A = (Q, T)$$

Si definiscono i concetti di cammino e cammino massimale.
Un **cammino** $\pi = q_0\ q_1\ q_2..., \ (q_i, q_{i+1}) \in T \forall i$ è una sequenza di stati (anche infinita) che sono legati da una transizione.
Un **cammino massimale** è un cammino che non può essere ulteriormente esteso. Anch'esso può essere finito o infinito.
Un **suffisso di ordine** i **di** π è il cammino $\pi^{(i)} = q_i\ q_{i+1}....$

Definizione 35. *Dato un insieme di* ***proposizioni atomiche*** $Z = \{z_1, z_2, ...\}$ *e un sistema di transizioni $A = (Q, T)$, si associa ad ogni $q \in Q$ l'insieme delle proposizioni atomiche che sono vere in quel determinato stato tramite una funzione chiamata* ***funzione di interpretazione****:*

$$I : Q \to 2^Z$$

Il ***modello di Kripke****, quindi, è così definito:*

$$M = (Q, T, I)$$

Nota: nel modello di Kripke non è specificato uno stato iniziale.

10 Logica temporale lineare - LTL

La logica temporale linare è definita tramite sintassi, semantica e algoritmi, ovvero formule valide nel modello di Kripke.

Definizione 36. *Le* ***proposizioni atomiche*** *sono così definite:*
$AP = \{p_1, p_2, ..., p_i, ..., q, r, ...\}$

Formule ben formate - FBF_{LTL}

Le formule ben formate sono formule che possono essere verificate immediatamente in un certo stato:

- ogni proposizione atomica è una FBF_{LTL}
- $true,\ false$ sono FBF_{LTL}, e verranno indicati come $\top = true,\ \bot = false$
- induzione strutturale: $\alpha,\ \beta$ sono FBF_{LTL}, allora lo sono anche $\neg\alpha,\ \alpha \vee \beta,\ \alpha \wedge \beta,\ \alpha \Rightarrow \beta,\ \alpha \Leftrightarrow \beta$

Induzione strutturale

Se $\alpha,\ \beta$ sono FBF_{LTL}, anche le seguenti sono FBF_{LTL}:

- $X\alpha$: nel prossimo stato α è vera
- $F\alpha$: prima o poi α diventerà vera $= \diamond\alpha$
- $G\alpha$: α è sempre vera $= \square\alpha$
- $\alpha U \beta$: untill - α is true untill β is false $= U(\alpha, \beta)$ (prima è vera α e poi diventa vera β)

10.1 Semantica LTL

Le formule LTL vengono interpretate sul modello di Kripke:

1. è definito un criterio per stabilire se una formula α è vera in un cammino massimale π (computazione)
2. la formula α è vera in uno stato q del modello di Kripke se è vera in tutti i cammini massimali che partono da q

Dato $\pi = q_0\ q_1\ q_2...$ un cammino massimale e sia α una formula di LTL, allora α è vera nel cammino π si denota con

$$\pi \models \alpha$$

Relazione $\models$ per induzione

Siano $\alpha, \beta \in FBF_{LTL}$ e $p \in PA$ proposizioni atomiche, allora:

- $\pi \models p$ SSE $p \in I(q_0)$, con q_0 stato iniziale del cammino
- $\pi \models \neg\alpha$ SSE $\pi \not\models \alpha$
- $\pi \models \alpha \vee \beta$ SSE $\pi \models \alpha \ \vee\ \pi \models \beta$

Nota: gli operatori $\neg$ e $\vee$ sono sufficienti per costruire tutti gli altri connettivi logici ($\wedge,\ \Rightarrow, ...$)

10.2 Operatori temporali

Date $alpha, \beta \in FBF_{LTL}$, allora:

- $\pi \models X\alpha$ SSE $\pi^{(1)} \models \alpha$, dove $\pi^{(1)}$ è il suffisso del cammino π che esclude il primo stato, cioè $\pi = q_0\ q_1\ q_2..., \ \pi^{(1)} = q_1\ q_2...$
- $\pi \models F\alpha$ SSE $\exists\ i \in \mathbb{N}\ tc: \ \pi^{(i)} \models \alpha$
- $\pi \models G\alpha$ SSE $\forall i \in \mathbb{N}\ tc: \ \pi^{(i)} \models \alpha$
- $\pi \models \alpha U \beta$ SSE:
  1. $\exists\ i \in \mathbb{N}\ tc: \ \pi^{(i)} \models \beta \Rightarrow \pi \models F\beta$
  2. $\forall: \ h\ 0 \leqslant h < i, \ \pi^{(h)} \models \alpha$

Esempio 4. - $FG\alpha$ *"prima o poi α diventa sempre vera"*

- $GF\alpha$ *"è sempre vero che prima o poi α diventerà vera"*
- $G\neg(cs_1 \wedge cs_2)$ *"è sempre vero che non vale $cs_1 \wedge cs_2$" - mutua esclusione*
- $G(req \Rightarrow XFack)$ *"è sempre vero che se è spedita una richiesta allora dallo stato successivo in cui è vera req, prima o poi sarà vera ack"*
- $G(req \Rightarrow X(reqUack))$ *"è sempre vero che se vale req allora dallo stato successivo in cui vale req prima o poi diventa vera ack con req vera in tutti gli stati precedenti" - ack e req non possono essere veri contemporaneamente perchè prima di untill c'è X*
- $G(req \Rightarrow ((req \wedge \neg ack)\ U\ (ack \wedge \neg req)))$ *- ack e req non possono essere vere contemporaneamente*

Operatori derivati

- $\alpha W \beta \equiv G\alpha \vee (\alpha U \beta)$ **untill debole**, ovvero dice che, a differenza di U, può succedere che β non diventi mai vera
- $\pi \models \alpha R \beta$ SSE $\forall k \geqslant 0: \ (\pi^{(k)} \models \beta \vee \exists\ h < k: \ \pi^{(h)} \models \alpha)$ da cui deriva la seguente equivalenza: $\alpha R \beta \equiv \beta W(\alpha \wedge \beta)$ **release**

Formule equivalenti

$\alpha \equiv \beta$ SSE $\forall \pi: \ (\pi \models \alpha \Leftrightarrow \pi \models \beta)$ equivalenza

Esempio 5. *queste formule di equivalenza evidenziano una natura ricorsiva*

- $F\alpha \equiv \alpha \vee XF\alpha$ *"prima o poi diveterà vera α"*
- $G\alpha \equiv \alpha \wedge XG\alpha$ *"in ogni stato/sempre è vera α"*
- $\alpha U \beta \equiv \beta \vee (\alpha \wedge X(\alpha U \beta))$ *"prima o poi β diventerà vera sapendo che in tutti gli istanti precedenti è stata vera α"*

- $FGF\alpha \equiv GF\alpha$ *"prima o poi è vero $GF\alpha$", con $F\alpha$ che è il suffisso di π per il quale è vera α*

- $GFG\alpha \equiv FG\alpha$ *speculare alla precedente*

$T\ U\alpha \equiv F\alpha \equiv \alpha$ "α prima o poi diventa vera e in tutti gli stati precedenti è vero T (=true)".
$\neg F\neg\alpha \equiv G\alpha$ "non è vero che prima o poi non sarà vera α", ovvero "è sempre vero che è vera α".

Definizione 37. $\{X, U\}$ *è un* ***insieme minimale di operatori*** *dal quali si possono derivare tutti gli altri, ma non è l'unico insieme minimale!*

10.3 Negazione

Cosa significa "non è vero che $F\alpha$"?
Significa: "non è vero che $F\alpha$", ovvero "Non è vero che $\forall\pi$ prima o poi diventerà vera α". Questo può essere tradotto in "$\exists\ \pi$ in cui α è sempre falsa".
Cosa significa $\neg F\alpha$?
Significa: "$\forall\pi$ non è vero che prima o poi diventerà vera α", ovvero $G\neg\alpha$, che significa "$\forall\pi$ (in ogni cammino) α è sempre falsa".
ATTENZIONE: $\neg F\alpha$ NON è la negazione di $F\alpha$! La negazione di $F\alpha$ non si può esprimere in LTL perchè LTL esprime solamente formule valide $\forall\pi$, non può esprimere formule che esprimono $\exists\ \pi$.

10.4 Limiti espressivi

LTL non può esprimere proprietà del tipo "$\exists\ \pi$ in cui α è vera".

Esempio 6. *"**è sempre possibile** tornare allo stato iniziale" si può tradurre con "$\exists$ almeno un cammino dal quale è possibile tornare allo stato iniziale". Questo non è esprimibile tramite le formule LTL.*

Problema della mutua esclusione

I requisiti per la mutua esclusione sono tre:

(i) i due processi non sono mai contemporaneamente nella sezione critica

(ii) se un processo richiede la risorsa prima o poi entrerà nella sezione critica

(iii) se un solo processo richiede la risorsa, *deve poter* accedere alla sezione critica (l'altro processo non può impedirgli di accedere alla sezione critica)

Queste formule si traducono in LTL come:

1. $G\neg(sc_1 \wedge sc_2)$
2. $G(req_1 \Rightarrow Fsc_1)$
3. ??? non è esprimibile in LTL perchè indica una possibilità

10.5 Equivalenza risptto ad una logica

Definizione 38. *Due modelli di Kripke M_1, M_2 con stati "iniziali" q_0, s_0 si dicono* ***equivalenti*** *rispetto alla logica L se per ogni formula $\alpha \in FBF_L$*

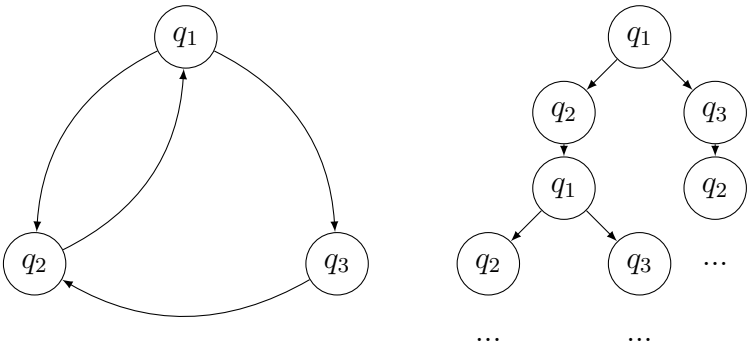
$$M_1,\ q_0 \vDash \alpha \ \Leftrightarrow\ M_2,\ s_0 \vDash \alpha$$

10.6 Albero di computazione

Un albero è un grafo aciclico orientato. L'albero di computazione è derivato da un modello di Kripke per poter rappresentare tutti i possibili cammini di quel dato modello.
L'albero di computazione si costruisce così:

- si sceglie uno stato iniziale e lo si disegna come radice dell'albero
- si guarda il numero di archi uscenti dal nodo iniziale e li si disegna nell'albero insieme al nodi destinazione
- si procede così per ogni nodo del modello di Kripke

Facciamo un **esempio** di un albero di computazione di un modello di Kripke:



10.7 Esercizi

Tradurre i seguenti enunciati in LTL:
1- "chi ruba, presto o tardi, finirà in galera".
prop. atomiche: hr="ho rubato"; c="mi trovo in galera"
riformulo: "se hai rubato prima o poi finirai in galera".

$$G(hr \Rightarrow XFc)$$

anche la soluzione $G(hr \Rightarrow hr\ U\ c)$ è valida.
variante1: nella frase da tradurre non si dice niente su chi non ruba, quindi aggiungo la proposizione "solo chi ruba va in galera".

$$\neg c\ W\ hr$$

ovvero non si va in prigione finchè non si ruba.
variante2: "chi ruba finirà in carcere, ma solo dopo aver parlato con un avvocato".
nuova prop. atomica: pa="ho parlato con un avvocato"

$$G(hr \Rightarrow (XFc \wedge (\neg c \, U \, pa)))$$

2- "se la cabina è in movimento verso l'alto, si trova all'altezza del secondo piano, ed è stato premuto il pulsante interno di richiesta del quinto piano, la cabina non cambierà direzione fino a quando avrà raggiunto il quinto piano".
prop. atomiche: su="la cabina sta salendo"; pi="cabina all'altezza del piano i"; ri="il pulsante interno del piano i è stato premuto".

$$G((su \wedge p2 \wedge rs) \Rightarrow (su \, U \, ps))$$

11 Computational Tree Logic - CTL - sintassi

Proposizioni atomiche: $AP = \{p_1, p_2, ..., q, r, ...\}$
Formule ben formate: FBF_{CTL}

- $\forall p \in AP, \; p \in FBF_{CTL}$
- $\forall \alpha, \beta \in FBF_{CTL}$:
  1. $\neg\alpha, \; \alpha \vee \beta \in FBF_{CTL}$ operatori dai quali si possono derivate tutti gli altri
  2. $AX\alpha, \; EX\alpha \in FBF_{CTL}$ con A=all, E=exists, e significano rispettivamente "$\forall \pi$ nel prossimo stato vale α" e "$\exists \; \pi$ nel quale nel prossimo stato vale α"
  3. $AF\alpha, \; EF\alpha \in FBF_{CTL}$
  4. $AG\alpha, \; EG\alpha \in FBF_{CTL}$
  5. $A(\alpha U \beta) \in FBF_{CTL}, \; E(\alpha U \beta) \in FBF_{CTL}$

Nota: $AFG\alpha \notin FBF_{CTL}$ perchè non è costruita correttamente in quanto ogni operatore non è preceduto o da A o da E.

Osservazione 1. *ci sono formule che sono in CTL e in LTL, ad esempio $AF\alpha$;*
ci sono formule che sono in LTL ma non in CTL, ad esempio $FG\alpha$;
ci sono formule che sono in CTL ma non in LTL, ad esempio $EF\alpha$.

Data una formula $f \in CTL$ è possibile che $\exists \; f' \in LTL$ tale che $f \equiv f'$.

Confronto tra LTL e CTL

Molte proprietà si possono esprimere sia in CTL che in LTL:

- invarianti, ad esempio $AG\neg p$ in CTL equivale a $G\neg p$ in LTL

- reattività (proprietà invarianti condizionali), ad esempio $AG(p \Rightarrow AFq)$ in CTL equivale a $G(p \Rightarrow Fq)$ in LTL

Ma ci sono alcune proprietà CTL che non si possono esprimere in LTL, ad esempio $AGEFp$, ovvero "da ogni stato raggiungibile in ogni cammino *è sempre possibile* raggiungere uno stato nel quale vale p" (si può dimostrare).
E altre proprietà LTL che non si possono esprimere in CTL, ad esempio FGp, ovvero "in ogni cammino prima o poi si raggiungerà uno stato in cui p rimarrà sempre vera" (si può dimostrare).
Nota: se due proposizioni sono equivalenti lo sono in tutti i modelli di Kripke.

11.1 Algoritmo per LTL

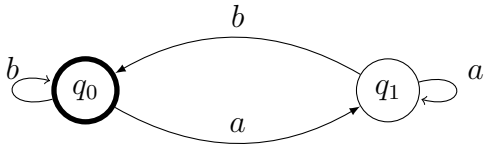
Esistono automi a stati finiti, detti automi di Büchi, che riconosco parole infinite su un alfabeto Σ e sono così definiti

$$B = (Q, q_0, \delta, F)$$

- Q: insieme finito di stati
- $q_0 \in Q$: stato iniziale
- $\delta \subseteq Q \times \Sigma \times W$: relazione di transizione
- $F \subseteq Q$: insieme degli stati accettanti

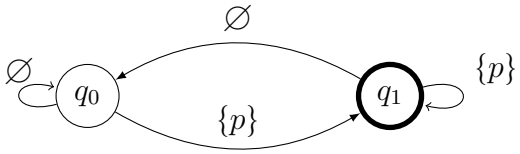
Una parola infinita $w = a_0a_1...$ è accettata da B se la sequenza infinita corrispondente di stati $q_0q_1...$ passa infinite volte per almeno uno stato in F.
Sia $L(B)$ il linguaggio dell'automa B, ovvero l'insieme di tutte le parole infinite accettate dall'automa B.

Esempio 7. *q_0 stato iniziale e finale*



$w_1 = bbbbbbbbb...$ $w_1 \in L(B)$
$w_2 = bbaaabbbb...$ $w_2 \in L(B)$
$w_3 = bababababab...$ $w_3 \in L(B)$
$w_4 = baabbbaaa...$ $w_4 \notin L(B)$ *perchè non si arriva mai allo stato finale*

Esempio 8. *q_0 stato iniziale e q_1 stato finale. La formula GFp è vera nel seguente automa?*



$w_1 = \varnothing\{p\}\{p\}\varnothing\{p\}\varnothing\varnothing\varnothing...$ $w_1 \notin L(B)$ *perchè* GFp *non è vera in* B *in quanto non si finisce nello stato finale in cui vale* p
$w_2 = \varnothing\{p\}\varnothing\{p\}\varnothing\{p\}\varnothing...$ $w_2 \in L(B)$

Il problema è verificare se α è vera in (M, q_0), con M modello di Kripke.

1. si costruisce l'automa $B_{\neg\alpha}$, ovvero l'automa che accetta sequenze infinite (=parole infinite) in cui α non è vera
2. si trasforma M in un automa etichettato da insiemi di proposizioni atomiche
3. si calcola il prodotto sincrono (=esecuzione parallela di due automi in modo che eseguano le stesse transizioni) dei due automi $P\ S$
4. se $L(P\ S) = \varnothing \Rightarrow M,\ q_0 \vDash \alpha$, ovvero in $M \nvDash$ un cammino infinito in cui α non è verificata

11.2 Algoritmo per CTL

Definizione 39. *Sia* $M = (Q, T, I)$ *un modello di Kripke. Sia* α *una formula e definiamo l'****estensione di*** α *come l'insieme di tutti gli stati in cui* α *è valida:*

$$[[\alpha]] = \{q \in Q \mid M,\ q \vDash \alpha\}$$

- $\alpha = true,\ [[\alpha]] = true$
- $\alpha = false,\ [[\alpha]] = \varnothing$
- $\alpha = p,\ [[\alpha]] =$ *insieme degli stati in cui è vera* p
- $[[\alpha \vee \beta]] = [[\alpha]] \cup [[\beta]]$
- $[[\neg\alpha]] = \overline{[[\alpha]]}$ *(complemento)*

12 Insiemi parzialmente ordinati

Definizione 40. *Sia data una* ***relazione d'ordine parziale*** *su* A: $\leqslant\ \subseteq\ A \times A$. *Questa è:*

- *riflessiva:* $x \leqslant x,\ \forall x \in A$
- *antisimmetrica:* $x \leqslant y \wedge y \leqslant x \Rightarrow x = y,\ \forall x, y \in A$
- *transitiva:* $x \leqslant y \wedge y \leqslant z \Rightarrow x \leqslant z,\ \forall x, y, z \in A$

Sia $(A, \leqslant)$ un insieme parzialmente ordinato e $B \subseteq A$, allora:

- $x \in A$ è un **maggiorante** di B se $y \leqslant x,\ \forall y \in B$. B^* è l'insieme dei maggioranti di B
- $x \in A$ è un **minorante** di B se $x \leqslant y,\ \forall y \in B$. B_* è l'insieme dei minoranti di B

- B si dice **limitato superiormente** se $B^* \neq \varnothing$
- B si dice **limitato inferiormente** se $B_* \neq \varnothing$
- $x \in B$ è il **massimo** di B se $y \leqslant x,\ \forall y \in B$
- $x \in B$ è il **minimo** di B se $x \leqslant y,\ \forall y \in B$
- $x \in B$ è un elemento **massimale** in B se $x \leqslant y \Rightarrow y = x$, per $y \in B$
- $x \in B$ è un elemento **minimale** in B se $y \leqslant x \Rightarrow y = x$, per $y \in B$

Se x è il minimo di B^*, diciamo che x è l'**estremo superiore (join)** di B e scriviamo $x = \sup\{B\}$ o anche $x = \bigvee B$.
Se x è il massimo di B_*, diciamo che x è l'**estremo inferiore (meet)** di B e scriviamo $x = \inf\{B\}$ o anche $x = \bigwedge B$.
In particolare se $B = \{x, y\}$ scriveremo $x \bigvee y$ per indicare $\bigvee B$ (se esiste) e $x \bigwedge y$ per indicare $\bigwedge B$ (se esiste).

Definizione 41. *Un **reticolo** è un insieme parzialmente ordinato $(L, \leqslant)$ tc: $\forall x, y \in L,\ \exists\ x \bigvee y$ e $x \bigwedge y$, ovvero per ogni coppia di elementi di L esistono estremo superiore ed inferiore.*
*Un reticolo si dice **completo** se $\forall B \subseteq L$ esistono $\bigvee B$ e $\bigwedge B$.*

13 Insiemi parzialmente ordinati e funzioni monotòne

Definizione 42. *Dati due insiemi parzialmente ordinati $(A, \leqslant)$ e $(B, \leqslant)$, una funzione $f : A \to B$ si dice **monotòna** se $\forall x, y \in A$ vale*

$$x \leqslant y \Rightarrow f(x) \leqslant f(y)$$

Definizione 43. *Considerata $f : X \to X$ funzione monotòna, un elemento $x \in X$ si dice **punto fisso** di f se $f(x) = x$.*

Esempio 9. *Si consideri $A = 2^{\mathbb{N}},\ S \subseteq \mathbb{N}, con (A, \leqslant)$*
1- $f : f(S) = S \cup \{2, 7\}$

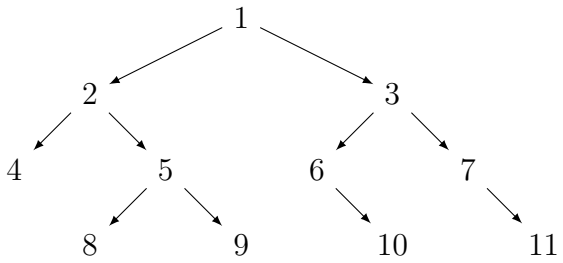
- *f è monotòna*
- *punti fissi: tutti i sottoinsiemi che contengono $\{2, 7\}$*
- *minimo punto fisso: $\{2, 7\}$*
- *massimo punto fisso: $\mathbb{N}$*

2- $f : f(S) = S \cap \{2, 7, 8\}$

- *f è monotòna*
- *punti fissi: tutti i sottoinsiemi che contengono $\{2, 7, 8\}$, ovvero $2^{\{2,7,8\}}$*

- *minimo punto fisso:* $\varnothing$
- *massimo punto fisso:* $\{2, 7, 8\}$

3- $A = \{1, 2, ..., 11\},\ (\mathcal{P}(A), \subseteq),\ \mathcal{P}(A) = 2^A$



Si ha $f : \mathcal{P}(A) \to \mathcal{P}(A)$ *definita come* $f(S) = S \cup \{x \in A \mid x \text{ è figlio di } y \in S\}$, *con* $S \subseteq A$.
Ad esempio: $f(\{2, 6\}) = \{2, 6, 4, 5, 10\}$ *con* $f(\{2, 6\}) \neq \{2, 6\}$

- f *è monotòna*
- $f(\{2, 6, 4, 5, 10\}) = \{2, 6, 4, 5, 8, 9, 10\} = M,\ f(M) = M$ *punto fisso. Un altro punto fisso è* $\{7, 11\}$ *perchè* $f(\{7, 11\}) = \{7, 11\}$
- $f(\varnothing) = \varnothing$
- *massimo punto fisso:* A
- *minimo punto fisso: ogni singola foglia è un punto fisso e* $\varnothing$ *è il minimo punto fisso (e in questo caso è anche il minimo reticolo)*

I punti fissi di f *sono sottoinsiemi* ***chiusi verso il basso****, ovvero: se un insieme contiene un nodo allora contiene anche tutti i nodi che discendono da quel nodo.*

13.1 Teoremi

Teorema 7 (di Knaster-Tarski o teorema del punto fisso). *Sia* $(L, \leqslant)$ *un reticolo completo e* $f : L \to L$ *una funzione monotòna. Allora* f *ha un minimo e un massimo punto fisso.*

Osservazione 2. *se* $f : \mathcal{P}(A) \to \overline{\mathcal{P}(A)}$ *(*$\overline{\mathcal{P}(A)}$ *complemento di* $\mathcal{P}(A)$*) e* S_1, S_2 *tali che* $S_1 \subseteq S_2$, *allora* f *non è monotòna perchè* $\overline{S_2} \subseteq \overline{S_1}$.

Dim. *si consideri il caso particolare in cui* $L = 2^A$, *per un insieme* A, *quindi* $f : 2^A \to 2^A$.
Si costruisce l'insieme $Z = \{T \subseteq A \mid f(T) \subseteq T\}$. *Gli elementi di* Z *saranno chiamati punti pre-fissi.*
Z *non può essere vuoto per ogni reticolo e per ogni funzione monotòna perchè tra i sottoinsiemi di* A *c'è anche* A *stesso, quindi* Z *contiene almento l'insieme* A.
Si suppone che f *abbia dei punti fissi, quindi* Z *li conterrebbe tutti per costruzione, in quanto* $f(T) = T \ \equiv \ f(T) \subseteq T \ \wedge \ f(T) \supseteq T$.
Si pone $m = \bigcap Z$ *intersezione di sottoinsiemi di* $A \Rightarrow m \in dom(f)$.

$\forall S \subseteq Z,\ m \subseteq S \Rightarrow f(m) \subseteq f(S)$ *perchè* f *è monotòna; segue che*
$f(m) \subseteq f(S) \subseteq S \ \Rightarrow f(m) \subseteq S$, *ma siccome* $m \subseteq S$ *allora* $f(m) \subseteq m \subseteq S$
$\Rightarrow f(m) \subseteq m$
$m \in Z$ ($m = \min Z$), *ovvero* m *è il minimo elemento di* Z *perchè è l'intersezione di tutti i sottoinsiemi di* A, *quindi* $\nexists$ *in* Z *un elemento minimale che non contenga* m.
$f(m) \subseteq m \ \Rightarrow f(f(m)) \subseteq f(m)$ *perchè* f *è monotòna. Se* $x = f(m)$ *allora* $f(x) \subseteq x \Rightarrow x \in Z$ *(come prima quando si conclude che* $m \in Z$*)* $\Rightarrow f(m) \in Z$
Siccome $m = \min Z \Rightarrow m \subseteq f(m)$, *perchè* $\nexists$ *in* Z *un elemento minimale che non contenga* m.
$\Rightarrow m = f(m)$
$m = f(m)$ *è un punto fisso di* Z *e in partiolare è il minimo punto fisso (perchè* $m = \min Z$*).*

$\#$

Nota: si può anche dimostrare allo stesso modo che esiste il massimo punto fisso.

Teorema 8 (di Kleene). *Sia* $f : 2^A \to 2^A$ *monotòna.* f *si dice* ***continua*** *se:*

- $X_1 \subseteq X_2 \subseteq ... \subseteq X_i \subseteq ...$
- $f(X_1) \subseteq f(X_2) \subseteq ... \subseteq f(x_i) \subseteq ...$

$\Rightarrow f(\bigcup x_i) = \bigcup f(x_i)$, *con* $\bigcup X_i \subseteq A$.
In generale non è detto che questo sia valido sapendo solamente che f *è monotòna, ma è valido sempre solo per funzioni monotòne continue.*

Se f è continua, allora:

1. il minimo punto fisso di f si può ottenere calcolando
   $f(\varnothing),\ f(f(\varnothing)),\ f(f(f(\varnothing))), ...$ e dopo un numero finito di passi si cascherà in un punto fisso, che sarà il minimo punto fisso.

2. il massimo punto fisso di f si può ottenere calcolando
   $f(A),\ f(f(A)),\ f(f(f(A))), ...$ e dopo un numero finito di passi si cascherà in un punto fisso, che sarà il massimo punto fisso.

Esempio 10 (algoritmo per LTL e CTL)**.** *Si consideri la formula* $\alpha = AF\beta$.
Si definisce la funzione $f_\alpha : 2^Q \to 2^Q$ *dipendente fortemente da* α.
$\forall H \subseteq Q,\ f_\alpha(H) = [[\beta]] \cup \{q \in Q \mid \forall (q, q') \in T :\ q' \in H\}$, *con* q' *stato d'arrivo in* H.

Osservazione 3. $f_\alpha(\varnothing) = [[\beta]] \cup \varnothing = [[\beta]]$ *(primo passo)*
Si può dimostrare che $[[\alpha]]$ *è il minimo punto fisso di* f_α *funzione monotòna.*
$f_\alpha(\varnothing) = [[\beta]],\ f_\alpha([[\beta]]) = ...$ *si itera finchè non si trova un punto fisso, e alla fine si troverà* $f_\alpha([[\alpha]]) = [[\alpha]]$.
$[[\alpha]]$ *è il minimo punto fisso di* f_α.

Si consideri la formula $\alpha = EG\beta$.
Si definisce la funzione $g_\alpha : 2^Q \to 2^Q$ dipendente fortemente da α.
$\forall H \subseteq Q,\ g_\alpha(H) = [[\beta]] \cap \{q \in Q \mid \exists\ (q, q') \in T :\ q' \in H\}$, con q' stato d'arrivo in H.

Osservazione 4. *$g_\beta(Q) = [[\beta]]$, ovvero si restringe via via l'insieme fino a raggiungere il massimo punto fisso.*
$[[\alpha]]$ è il massimo punto fisso di g_α.

Nota: anche per l'operatore temporale *untill* va cercato il minimo punto fisso.

14 Calcolo μ

Il **calcolo** μ consente di definire formule ricorsive.
Si vuole supporre di avere solo l'operatore temporale X e i quantificatori $A,\ E$.
È possibile esprimere la formula $EF\alpha$ solo con l'operatore temporale X?
Si srotola la formula: $EF\alpha \equiv \alpha \vee EX\alpha \vee EXEX\alpha \vee ...$ (formula infinita)
$\Rightarrow EF\alpha \equiv \alpha \vee EX(\alpha \vee EX\alpha \vee EXEX\alpha\vee)...$
$\Rightarrow EF\alpha \equiv \alpha \vee EX(EF\alpha)$ formula ricorsiva
Si fissa una notazione: $\mu Y.(\alpha \vee EXY)$, con Y = oggetto che si cerca di definire.
μ è l'operatore usato per indicare il minimo punto fisso.
Per la formula $AG\alpha$ si può fare un ragionamento analogo:
Si srotola la formula: $AG\alpha \equiv \alpha \vee AX\alpha \vee AXAX\alpha \vee ...$ (formula infinita)
$\Rightarrow AG\alpha \equiv \alpha \vee AX(\alpha \vee AX\alpha \vee AXAX\alpha\vee)...$
$\Rightarrow AG\alpha \equiv \alpha \vee AX(AG\alpha)$ formula ricorsiva
Si fissa una notazione: $\nu Y.(\alpha \vee AXY)$, con Y = oggetto che si cerca di definire
ν è l'operatore usato per indicare il massimo punto fisso.

14.1 Regole del calcolo μ

Siano date delle proposizioni atomiche $AP = \{p_1, p_2, ..., q, r, ...\}$
Date $\alpha, \beta \in FBF_{CTL}$:

1. $\alpha \vee \beta,\ \neg\alpha$ sono formule
2. $EX\alpha,\ AX\alpha$ sono formule
3. $\mu Y.f(Y)$ è una formula, con $f(Y)$ formula nella quale compare la variabile Y, con restrizioni sulle negazioni
4. $\nu Y.f(Y)$ è una formula, con $f(Y)$ formula nella quale compare la variabile Y, con restrizioni sulle negazioni

Il calcolo μ esprime la massima potenza espressiva più di tutte le logiche finora studiate, a discapito, però, della sua alta complessità e della potenziale "oscurità" delle sue formule.
Questo può essere riassunto con la seguente notazione

$$CTL^* \subset \mu - calculus$$

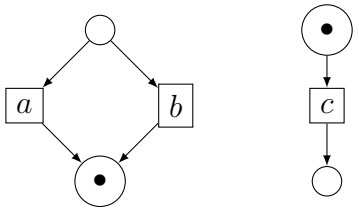
14.1.1 Complessità e aspetti algoritmici

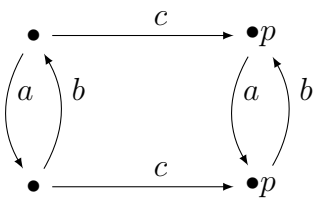
Dati M un modello di Kripke e f una formula, siano

- CTL: $O(|M| \times |f|)$, con $|M|$ = numero stati del modello di Kripke e $|f|$ = lunghezza della formula
- LTL: $|M| \times 2^{|f|}$

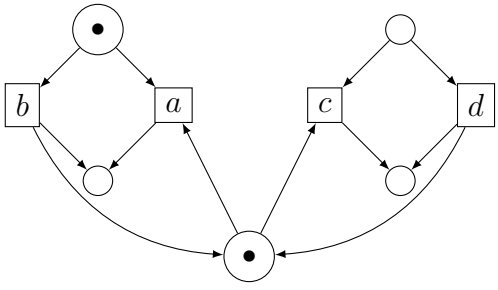
Sebbene possa sembrare che CTL possa essere più efficiente, in realtà è molto complicato fare un confronto tra LTL e CTL in quanto potrebbe essere che CTL sia più semplice di LTL ma con una fomrula f molto molto lunga.

15 Fairness





In questo caso Fp è falsa perchè può succedere che c non scatti mai.
Una possibilità però, potrebbe essere che sebbene c non scatti mai, c sia abilitata fin dall'inizio.
Un'esecuzione di un sistema è **unfair - fairness debole** se un evento è sempre abilitato da un certo punto in poi ma non scatta mai (esecuzione irrealistica).



In questo caso, invece, l'esecuzione di *abababababab...* è debolmente fair.
Un'esecuzione è **fortmente fair** se $GF(t \text{ abilitata}) \Rightarrow GF(t \text{ scatta})$

Esempio 11. $((ab)(cd)^{100})^{\infty}$ *è fortemente fair.*