Modelli della Concorrenza

Tommaso Ferrario (@TommasoFerrario18)

Telemaco Terzi (@Tezze2001)

October 2023

Indice

Capitolo 1

Introduzione

Possiamo definire diverse semantiche per la programmazione sequenziale:

- **Operazionale**: ho una macchina astratta e definisco i vari passi della computazione:

$$Input \longrightarrow M \longrightarrow Output \tag{1.1}$$

- **Denotazionale**: dato un programma funzionale P ho una funzione definita come:

$$f : \text{Dati}_I \to \text{Dati}_O \ (\lambda - \text{calcolo}) \tag{1.2}$$

- **Assiomatica**: l'input e l'output sono rappresentati come formule logiche. Questa tipologia prende il nome di *Triple di Hoare*.

$$\{Input\}P\{Output\} \tag{1.3}$$

Nella programmazione sequenziale si hanno due punti chiave che devono essere garantiti:

- **Terminazione del programma**.
- **Composizionalità** tra più comandi per ottenere un programma, ovvero:

$$\begin{aligned} s_1 : x = 2 \ \ &\{x = V\}\ x = 2\ \{x = 2\} \\ s_2 : x = 3 \ \ &\{x = V\}\ x = 3\ \{x = 3\} \\ &\{x = V\}\ s_1;\ s_2\ \{x = 3\} \end{aligned} \tag{1.4}$$

  Se esiste un programma s_1' tale che mi permette di ottenere lo stesso risultato si s_1, allora posso sostituirlo a s_1.

$$\begin{aligned} s_1' : &\{x = V\}\ x = 1;\ x = x + 1\ \{x = 2\} \\ &\{x = V\}\ s_1';\ s_2\ \{x = 3\} \end{aligned} \tag{1.5}$$

Capitolo 2

Calculus of Communicating Systems

Dati due processi p_1 ed p_2 si ha che essi sono specificati in **esecuzione concorrente** con l'utilizzo della seguente notazione:

$$p_1|p_2 \text{ dove } p_1, p_2 \in Proc_{CCS} \tag{2.1}$$

L'esecuzione in concorrenza può portare a diverse complicanze qualora non venga rispettato, per esempio, un certo ordine di esecuzione. Si ha quindi il **non determinismo**, ovvero il risultato ottenuto dall'esecuzione dei programmi dipende dall'ordine di esecuzione di essi. Inoltre, si perde la composizionalità dei processi.

Esempio 1 (**Non determinismo e nessuna composizionalità**). *Supponiamo di avere due programmi p_1 e p_2 definiti nel seguente modo:*

$$\begin{aligned} p_1 &= \{x = V\}x = 2\{x = 2\} \\ p_2 &= \{x = V\}x = 3\{x = 3\} \end{aligned} \tag{2.2}$$

con V che indica un qualunque valore.

L'esecuzione in parallelo di questi due programmi mi permette di ottenere i seguenti risultati:

$$\{x = V\}\ p_1|p_2\ \{x = 2 \vee x = 3\} \tag{2.3}$$

avendo quindi una situazione di non determinismo. Inoltre, definendo un nuovo programma p_1' possiamo osservare come la proprietà di composizionalità non risulta più valida.

$$p_1' = \{x = V\}\ x = 1; x = x + 1\ \{x = 2\} \tag{2.4}$$

Se proviamo a sostituire questo programma al posto di p_1 otteniamo:

$$\{x = V\}\ p_1'|p_2\ \{x = 2 \vee x = 3 \vee x = 4\} \tag{2.5}$$

Per mantenere il principio di composizionalità si cambia linguaggio di rappresentazione. Per modellare sistemi concorrenti esistono diverse cose:

- **Algebre di processi**: CCS e CSP
- **Automi a stati finiti**: tutti gli automi a stati finiti e i riconoscitori di linguaggi, ricadono anche i sistemi di transizione etichettati, quest'ultimi utili per modellare la semantica interleaving delle algebre dei processi.
- **Reti di petri**

2.1 Algebre dei processi

Hoare ha introdotto un nuovo paradigma di programmazione, il paradigma **CSP** (*Communicating Sequential Processes*). In questo paradigma non si ha più una memoria condivisa, ma un insieme di processi ciascuno con una sua memoria privata. Si ha un'interazione tra processi tramite lo scambio di messaggi del tipo hand-shacking, avendo quindi la sincronizzazione basata sullo scambio di informazioni. Viene fatto anche un processo particolare rappresentante la memoria condivisa. Avremo quindi:

$$x \mid p_1 \mid p_2 \tag{2.6}$$

dove x che rappresenta la memoria condivisa dai due processi.

Un diverso paradigma è quello proposto da Milner, il quale propose il lambda calcolo (λ-calcolo) per passare dal paradigma sequenziale a quello concorrente. In questo si studia in modo approfondito la composizionalità, sfruttando la composizione tra funzioni, cercando di non perderla nel concorrente. Introduce quindi una sorta di λ-calcolo concorrente, introducendo il **CCS** (*Calculus of Communicating Systems*), in cui pensa ad un calcolo algebrico per sistemi comunicanti. Adotta anche lui un paradigma che studia un sistema formato da componenti, chiamati processi. Questi processi comunicano tramite la sincronizzazione delle operazioni. Per la gestione di queste si utilizza la seguente notazione:

- a: indichiamo un processo generico che invia il messaggio.
- $\overline{a}$: indichiamo un processo generico che riceve.

Un **sistema**, quindi, è un insieme di processi il cui comportamento è gestito da un calcolo algebrico, si punta alle algebre di processi, ovvero linguaggi di specifica di sistemi concorrenti che si ispirano al calcolo dei sistemi comunicanti.

I messaggi di scambio corrispondono ad uno scambio di valori di variabili e questo è rappresentabile dall'algebra. Utilizzando questa tecnica, i processi possono interagire anche con l'ambiente esterno attraverso delle porte. Sfruttando questa tecnica non si ha più un sistema chiuso.

Dato un sistema P, si scrive:

$$P = p_1 \mid p_2 \mid p_3 \tag{2.7}$$

se P è formato dai processi p_1, p_2 e p_3, processi che sono interagenti a due a due. Ogni processo ha comunque una memoria privata.

Milner risolve il problema della *composizionalità* tramite l'uso di diverse porte che permettono ad un processo di comunicare con altri o con l'ambiente esterno. Quindi ogni processo può essere visto come un insieme di sotto-processi interagenti che però interagiscono tramite sincronizzazione con i processi esterni tramite una porta. Bisogna comunque mantenere il comportamento complessivo. Per il processo esterno è come se sostituissi il processo con cui comunica con il suo sotto-processo. Si introduce il concetto di **equivalenza all'osservazione**, che permette di sostituire un processo p_i con uno p_i' se sono equivalenti rispetto all'osservazione, ovvero se e solo se un qualsiasi osservatore esterno non è in grado di distinguere i due processi.

Con il termine *osservare* ci si riferisce all'interazione con il sistema dove agisce il processo. Questo deve essere valido per ogni possibile osservatore. Se questo è garantito la sostituzione di un processo non va a modificare l'esecuzione complessiva, senza incorrere in deadlock o altre problematiche. Per verificare l'equivalenza esistono diverse tecniche tra cui la **bisimulazione**.

2.2 Labeled Transition System

Per modellare la semantica interleaving delle algebre dei processi si utilizzano i **Labeled Transition System** (LTS).

Definizione 1 (**Labeled Transition System**). *Possiamo definire un **Labeled Transition System** (LTS) come un automa che specifica il comportamento di un processo. Esso è definito a partire da una quadrupla:*

$$LTS = (S, Act, T, s_0) \tag{2.8}$$

dove:

- S*: rappresenta un insieme di stati, che a differenza degli automi può non essere finito.*
- Act*: è un insieme delle possibili azioni (possono essere nomi o simboli).*
- T*: è una relazione definita come* $T \subseteq S \times Act \times S$ *tale che:*

$$(s, a, s') \in T \equiv s \xrightarrow{a} s'$$

- s_0*: rappresenta lo stato iniziale. Questo campo non sempre è presente.*

La transizione $s \xrightarrow{a} s'$ può essere estesa a:

$$s \xrightarrow{w} s', \text{ con } w \in Act^* \tag{2.9}$$

Dimostrazione 1. *È possibile dimostrare questa estensione per induzione:*

- ***Base**: $w = \varepsilon$, ovvero w è la stringa vuota, allora $s = s'$.*
- ***Passo induttivo**: $w = a.x$ con $a \in Act$ e $x \in Act^*$, allora:*

$$s \xrightarrow{a \cdot x} s' \iff s \xrightarrow{a} s'' \xrightarrow{x} s' \tag{2.10}$$

Quindi si può definire l'estensione:

$$\rightarrow = \bigcup_{a \in Act} \xrightarrow{a} \tag{2.11}$$

e definire la chiusura transitiva e riflessiva:

$$\xrightarrow{*} = \bigcup_{w \in Act^*} \xrightarrow{w} \tag{2.12}$$

2.3 CCS - Calculus of Communicating Systems

Il CCS è un linguaggio di specifica per sistemi concorrenti. È un linguaggio che astrae tutte le questioni legate ai dati.

Definizione 2 (**Calculus of Communicating Systems**). *Per definire il **Calculus of Communicating Systems** (CCS) puro dobbiamo definire:*

- *K: insieme di nomi di processi, che possono anche essere simboli di un alfabeto.*
- *A: insieme di nomi di azioni.*
- *$\overline{A}$: insieme di co-nomi di azioni contenute in A:*

$$\overline{A} = \{\overline{a} \mid a \in A\} \Rightarrow \overline{\overline{a}} = a \tag{2.13}$$

- *$Act = A \cup \overline{A} \cup \{\tau\}$: insieme delle azioni, dove $\tau \notin A$ corrisponde all'**azione di sincronizzazione** tra a e $\overline{a}$, ovvero la sincronizzazione è avvenuta. A e $\overline{A}$ sono azioni **osservabili** e si indica con:*

$$L = A \cup \overline{A}$$

mentre τ non è osservabile. Ricordando che osservare un'azione significa poter interagire con essa.

Alla fine un processo CCS è definito così:

$$P = a.P, P \in K \tag{2.14}$$

Significa che P esegue a e si comporta come P (ricorsione).

$$P_1 = a.P_2, P_1, P_2 \in K \tag{2.15}$$

Ogni processo CCS può essere descritto da un LTS associato e attraverso delle regole di inferenza strutturate nel seguente modo:

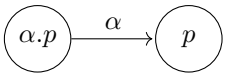
$$\frac{Premesse}{Conseguenze} \tag{2.16}$$

Dare un significato tramite LTS, regole di inferenza e sintassi, è detto semantica operazionale strutturale.

- **Processo vuoto**: è definito come $P = \mathbf{Nil}$ o 0 che ha un solo stato senza transazioni.

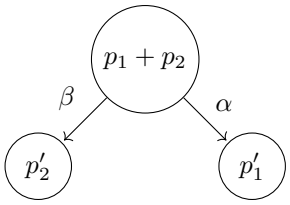
- **Prefisso**: in cui si ha $\alpha.P$ dove $P \in Proc_{CCS}$ e $\alpha \in Act$. Questo può essere rappresentato mediante inferenza nel seguente modo:

$$\frac{}{\alpha.P \xrightarrow{\alpha} P} \tag{2.17}$$



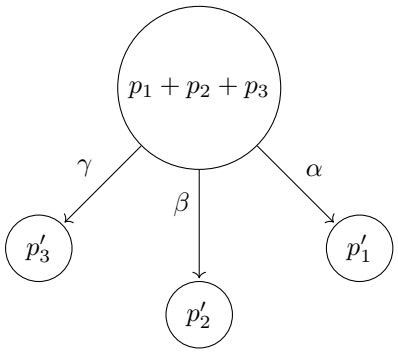
- **Somma**: siano $p_1, p_2 \in Proc_{CCS}$ posso comporli utilizzando l'operatore $+$ nel seguente modo $p_1 + p_2$. In questo caso per definire le regole di inferenza necessito $\alpha, \beta \in Act$ e $p'_1, p'_2 \in Proc_{CCS}$:

$$\frac{p_1 \xrightarrow{\alpha} p'_1}{p_1 + p_2 \xrightarrow{\alpha} p'_1} \quad \vee \quad \frac{p_2 \xrightarrow{\beta} p'_2}{p_1 + p_2 \xrightarrow{\beta} p'_2} \tag{2.18}$$



Questo caso può essere generalizzato per più processi. Posso avere $\sum_{i \in I} p_i$ avendo multiple somme di processi:

$$\frac{p_j \xrightarrow{\alpha} p'_j}{\sum_{i \in I} p_i \xrightarrow{\alpha} p'_j},\ j \in J \tag{2.19}$$

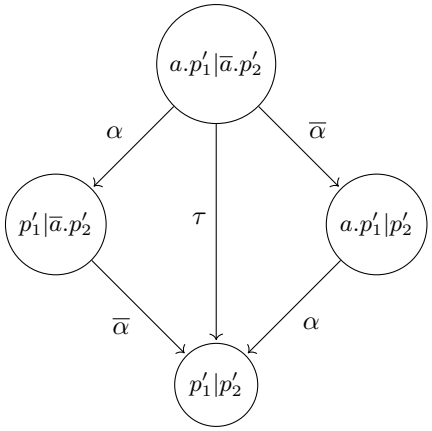


Nel caso di $I = \emptyset$ avrò $\sum_{i \in I} p_i = Nil$. Nel momento in cui ho 2 processi in una somma che con la medesima azione hanno 2 transizioni diverse allora si introduce un comportamento non deterministico perché non so quale dei due viene eseguito.

- **Composizione parallela**: indicata con il simbolo $p_1|p_2$ e utilizzando le regole di inferenza:

$$\frac{p_1 \xrightarrow{\alpha} p'_1}{p_1|p_2 \xrightarrow{\alpha} p'_1|p_2} \quad \vee \quad \frac{p_2 \xrightarrow{\overline{\alpha}} p'_2}{p_1|p_2 \xrightarrow{\overline{\alpha}} p_1|p'_2} \quad \vee \quad \frac{p_1 \xrightarrow{\alpha} p'_1 \wedge p_2 \xrightarrow{\overline{\alpha}} p'_2}{p_1|p_2 \xrightarrow{\tau} p'_1|p'_2} \tag{2.20}$$

in quest'ultimo caso, non potremo più avere altre sincronizzazioni.



- **Restrizione**: sia $L \subseteq A$. Si ha che:

$$p_{\setminus L} \tag{2.21}$$

indica che il processo p **non** può interagire con il suo ambiente con azioni in $L \cup \overline{L}$ ma le azioni in $L \cup \overline{L}$ sono locali a p:

$$\frac{p \xrightarrow{\alpha} p'}{p_{\setminus L} \xrightarrow{\alpha} p'_{\setminus L}} \tag{2.22}$$

- **Rietichettatura**: ovvero il cambiamento di nome ad una componente, ad una certa azione, per poter riusare tale nome. Ho quindi una funzione f tale che:

$$f : Act \to Act \tag{2.23}$$

inoltre, deve sempre essere garantito che:

  - $f(\tau) = \tau$
  - $f(\overline{a}) = \overline{f(a)}$

Ho quindi $p_{[f]}$ tale che:

$$\frac{p \xrightarrow{\alpha} p'}{p_{[f]} \xrightarrow{f(a)} p'_{[f]}} \tag{2.24}$$

- **Ridenominazione dei processi**: $K = P$ con K uguale al nome di processo e $P \in Proc_{CCS}$:

$$\frac{P \xrightarrow{\alpha} P' \wedge K = P}{K \xrightarrow{\alpha} P'} \tag{2.25}$$

Nel caso di composizione parallela posso eseguire le due operazioni o in una sequenza o nell'altra. Ho quindi una simulazione sequenziale non deterministica del comportamento del sistema dato dalla composizione parallela.

Per poter dire che una certa implementazione soddisfa ($\models$) una certa specifica o se due implementazioni diverse soddisfano la stessa specifica ci serve una relazione di equivalenza tra processi CCS, ovvero una relazione del tipo:

$$R \subseteq Proc_{ccs} \times Proc_{ccs} \tag{2.26}$$

tale che sia *riflessiva*, *simmetrica* e *transitiva*.

Inoltre si richiede:

- astrazione delgi stati in modo tale da considerare solo le azioni Act.
- astrazione delle sincronazzazioni interne, ovvero dalle τ.
- rispetto del non determinismo.

Milner poi asserisce che R deve essere inoltre una **congruenza** rispetto agli operatori del CCS.

Definizione 3 (**Congruenza**)**.** *Una relazione di equivalenza R è una **congruenza** se e solo se:*

$$\forall p, q \in Proc_{CCS} \wedge \forall c[.] \ contesto \ CCS \tag{2.27}$$

*avendo quindi un **contesto**, un'espressione CCS con qualcosa di mancante, CCS sostituibile con qualcosa, allora:*

$$se \ p \ R \ q \ allora \ si \ ha \ c[p] \ R \ c[q] \tag{2.28}$$

Con LTS perdo la concorrenza perché tutto viene eseguito come esecuzione sequenziale non deterministica (semantica interleaving). Con le reti di Petri si potrà simulare il fatto che sia parallelo.

Abbiamo visto come associare CCS a LTS passando per le regole di inferenza. Ora consideriamo un sistema di processi che interagisce con l'ambiente allora vogliamo sapere come trovare una relazione di equivalenza per sostituire un processo con un altro.

Nel caso di un processo sequenziale basta sostituire il modulo con altro rispettando i domini di I e O. Nel parallelo dobbiamo mantenere rispettate le interazioni con l'ambiente. Quindi una relazione di equivalenza con una congruenza rispetto agli operatori ccs.

Da una relazione di equivalenza $\simeq$ tra i processi $p, q \in Proc_{CCS}$ ci si aspetta che:

- $LTS(p)$ isomorfo $LTS(q)$ allora $p \simeq q$
- Si astraggono gli stati e si considerano solo le azioni.
- $p \simeq q \Rightarrow Tracce(p) = Tracce(q)$
- $p \simeq q$ allora p e q devono avere la stessa possibilità di generare deadlock nell'interazione con l'ambiente.
- $\simeq$ deve essere una congruenza rispetto agli operatori CCS, cioè deve essere possibile sostituire un sotto-processo con un suo equivalente.

Definizione 4 (**Isomorfismo**). *Dati due sistemi di transizione etichettati $LTS_1 = (Q_1, Act_1, T_1, q_{01})$ e $LTS_1 = (Q_2, Act_2, T_2, q_{02})$, possiamo affermare che LTS_1 è **isomorfo** a LTS_2 se e solo se:*

- $\alpha : Q_1 \to Q_2$ *è una funzione biunivoca.*
- $\beta : Act_1 \to Act_2$ *è una funzione biunivoca.*
- $\alpha(q_{01}) = q_{02}$
- $(q_1, a, q_2) \in T_1 \iff (\alpha(q_1), \beta(a), \alpha(q_1')) \in T_2$

Questa definizione è troppo stretta perché è come sostituire cose uguali. Introduciamo quindi una relazione di equivalenza come $p \sim q \iff Tracce(p) = Tracce(q)$, per tracce si intende la stessa sequenza di azioni possibile.

Definizione 5 (**Tracce**). *Se $P \in Proc_{CCS}$ allora:*

$$Tracce(P) = \{w \in Act^* \mid \exists P' \in Proc_{CCS},\ P \xrightarrow{w} P'\} \tag{2.29}$$

dove:

- *Se $w = \varepsilon$ allora $P = P'$*
- *Se $w = x_1.x_2.\ldots.x_n$ con $x_i \in Act$ $\exists P_1', P_2', \ldots, P_n' \in Proc_{CCS}$ tale che:*

$$P \xrightarrow{x_1} P_1' \xrightarrow{x_2} \ldots \xrightarrow{x_n} P_n' = P' \tag{2.30}$$

Definizione 6 (**Equivalenza rispetto alle tracce**). *Quindi $P_1 \overset{T}{\sim} P_2 \iff Tracce(P_1) = Tracce(P_2)$ come equivalenza rispetto alle tracce.*

Osservazione 1. *Con l'equivalenza rispetto alle tracce si introducono le seguenti osservazioni:*

- *$LTS(p)$ isomorfo $LTS(q)$ implica $p \overset{T}{\sim} q$*
- *Si ha un'astrazione di stati.*
- *$p \overset{T}{\sim} q \iff Tracce(p) = Tracce(q)$*
- *È una **congruenza** tra gli operatori CCS.*
- *Non garantisce di preservare la stessa probabilità di deadlock (o assenza) nell'interazione con l'ambiente.*

Lo studio delle tracce non è più sufficiente nel caso di sistemi concorrenti. Si necessità quindi di una nozione più restrittiva.

2.3.1 Bisimulazione forte

Dal momento che l'equivalenza secondo la Traccia può portare a Deadlock allora si utilizzerà l'uguaglianza secondo l'osservatore, ovvero se l'osservatore, interagendo, non nota differenze. In una variante si chiama Bisimulazione, perché devo poter distinguere tra le 2. In sostanza M_1 simula M_2 e M_2 simula M_1.

Definizione 7 (**Relazione di Bisimulazione Forte**). *Data una relazione binaria $R \subseteq Proc_{CCS} \times Proc_{CCS}$ è una relazione di **bisimulazione (forte)** ($\overset{Bis}{\sim}$) se e solo se $\forall p, q \in Proc_{CCS}$: pRq vale che:*

- $\forall \alpha \in Act = A \cup \overline{A} \cup \{\tau\}$ *se ho* $p \xrightarrow{\alpha} p'$ *allora deve esistere un processo*

$$\exists q' \text{ tale che } q \xrightarrow{\alpha} q' \text{ e si ha } p' R q' \tag{2.31}$$

- *E viceversa, ovvero* $\forall \alpha \in Act = A \cup \overline{A} \cup \{\tau\}$ *se ho* $q \xrightarrow{\alpha} q'$ *allora deve esistere un processo*

$$\exists p' \text{ tale che } p \xrightarrow{\alpha} p' \text{ e si ha } p' R q' \tag{2.32}$$

Due processi p *e* q *sono fortemente bisimili, indicato con la notazione* $p \overset{Bis}{\sim} q$ *se e solo se* $\exists R \subseteq Proc_{CCS} \times Proc_{CCS}$, *relazione di bisimulazione forte tale che:*

$$pRq \ \wedge \ \overset{Bis}{\sim} = \bigcup \{R \subseteq Proc_{CCS} \times Proc_{CCS} \mid R \text{ è una relazione di bisimulazione forte}\} \tag{2.33}$$

Teorema 1. *Se prendo* $\overset{Bis}{\sim} \subseteq Proc_{CCS} \times Proc_{CCS}$ *si dimostra che è una relazione di equivalenza, ovvero è riflessiva, simmetrica e transitiva. Quindi:*

$$p \overset{Bis}{\sim} q \iff \forall \alpha \in Act \text{ se } p \xrightarrow{\alpha} p' \text{ allora } \exists q' \text{ tale che } q \xrightarrow{\alpha} q' \ \wedge \ p' \overset{Bis}{\sim} q' \tag{2.34}$$

e se:

$$q \xrightarrow{\alpha} q' \text{ allora } \exists p' \text{ tale che } p \xrightarrow{\alpha} p' \ \wedge \ p' \overset{Bis}{\sim} q' \tag{2.35}$$

Teorema 2. *Se due processi sono fortemente bisimili allora sono sicuramente equivalenti rispetto alle tracce. Non vale il viceversa.*

$$p \overset{Bis}{\sim} q \Rightarrow p \overset{T}{\sim} q \tag{2.36}$$

Osservazione 2. *Per vedere che due processi sono bisimili devo quindi, per ogni esecuzione, ottenere due processi ancora bisimili (potendo quindi fare le azioni corrispondenti da entrambe le parti).*

Esempio 2. *Consideriamo i processi* $p_1 = a.b.Nil + a.c.Nil$ *e* $p_2 = a.(b.Nil + c.Nil)$ *Osserviamo subito che i*

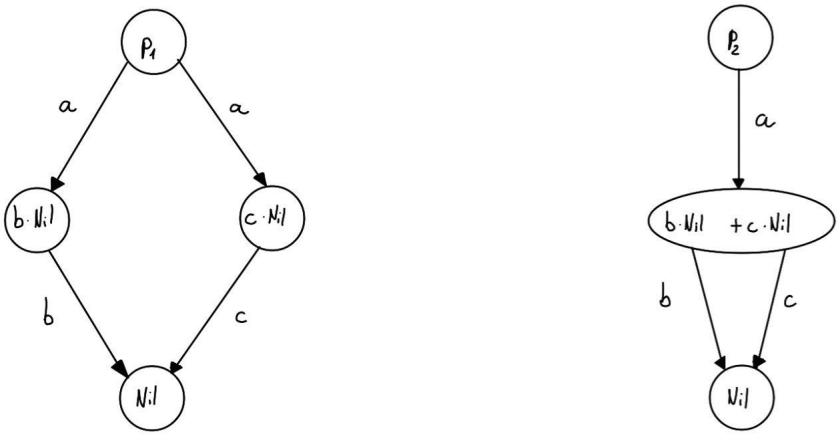


Figura 2.1: LTS dei processi p_1 e p_2

due processi sono equivalenti rispetto alle tracce, difatti:

- $Tracce(p_1) = \{\varepsilon, a, a.b, a.c\}$
- $Tracce(p_2) = \{\varepsilon, a, a.b, a.c\}$

Vediamo se i due processi sono anche bisimili.

- *Da* p_1 *possiamo eseguire l'azione* a *e possiamo fare lo stesso anche da* p_2. *Dobbiamo però chiederci se gli stati di arrivo sono anche essi in relazione di bisimulazione.*
- *Gli stati interessati sono* $b.Nil$ *e* $b.Nil + c.Nil$: *dal primo possiamo eseguire* b, *che è fattibile anche dal secondo, ma dal secondo possiamo eseguire* c, *che non è eseguibile dal primo (la bisimulazione richiede che entrambi gli stati siano simili tra loro), dunque i due processi non sono bisimili.*

Formalmente:

- $p_1 \xrightarrow{a} b.Nil$
- $p_2 \xrightarrow{a} b.Nil + c.Nil$

$b.Nil \overset{Bis}{\not\sim} b.Nil + c.Nil$ *inoltre, possiamo osservare che:* $b.Nil \overset{T}{\not\sim} b.Nil + c.Nil$

Esempio 3. *Consideriamo due processi che simulano dei buffer che possono contenere due elementi:*

- $B_0^1|B_0^1$ *dove* $B_0^1 = in.B_1^1$ *e* $B_1^1 = \overline{out}.B_0^1$
- $B_0^2 = in.B_1^2$, $B_1^2 = \overline{out}.B_0^2 + in.B_2^2$ *e* $B_2^2 = \overline{out}.B_1^2$

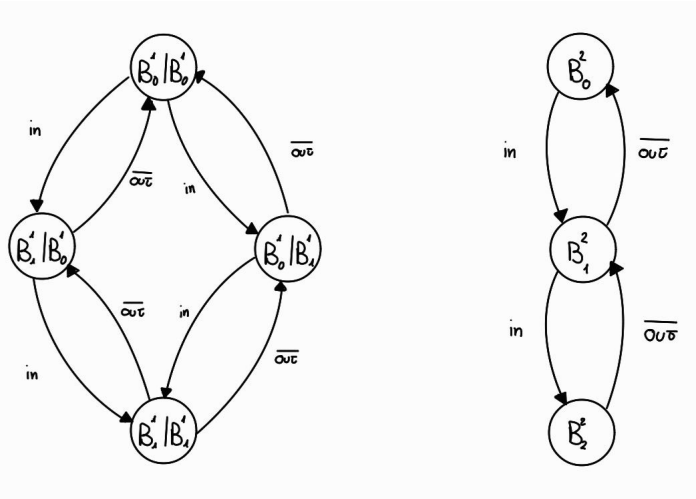


Figura 2.2: LTS dei processi $B_0^1|B_0^1$ e B_0^2

In questo esempio, possiamo osservare che:

$$(B_0^1|B_0^1) \overset{Bis}{\sim} B_0^2$$

B_0^2 *può essere messo in relazione con* $B_0^1|B_0^1$ *in quanto se vado in uno tra* $B_1^1|B_0^1$ *e* $B_0^1|B_1^1$ *posso sempre fare in e* $\overline{out}$*. Inoltre, Posso fare un discorso analogo per* B_2^2 *e* $B_1^1|B_1^1$*. Essendo questi ultimi quindi bisimili lo sono anche il nodo centrale coi due possibili nodi nel caso della composizione e di conseguenza lo sono anche* B_0^2 *e* $B_0^1|B_0^1$*.*

La bisimulazione si può testare mettendo i due processi in parallelo e restringendo le operazioni, se entrambi arrivano alla fine utilizzando la sincronizzazione allora sono bisimili.

Osservazione 3. *La Bisimulazione forte è una congruenza anche rispetto agli operatori CCS, ovvero se* $p, q \in Proc_{CCS} \wedge p \overset{Bis}{\sim} q$ *allora:*

- $\alpha.p \overset{Bis}{\sim} \alpha.q$
- $p + r \overset{Bis}{\sim} q + r$ *e* $r + p \overset{Bis}{\sim} r + q, \forall r \in Proc_{CCS}$
- $p|r \overset{Bis}{\sim} q|r$ *e* $r|p \overset{Bis}{\sim} r|q \forall r \in Proc_{CCS}$
- $\forall f : Act \to Act$ *funzione di rietichettatura, se* $p[f] \overset{Bis}{\sim} q[f]$ *allora* $p \overset{Bis}{\sim} q$
- $p_{\backslash L} \overset{Bis}{\sim} q_{\backslash L}$ *e* $L \subseteq Act$

Osservazione 4. *Inoltre valgono le seguenti proprietà:*

- ***Commutatività rispetto alla somma****:* $p + q \overset{Bis}{\sim} q + p$

- ***Commutatività rispetto alla parallelizzazione***: $p|q \overset{Bis}{\sim} q|p$
- ***Associatività rispetto alla somma***: $(p+q)+r \overset{Bis}{\sim} p+(q+r)$
- ***Associatività rispetto alla parallelizzazione***: $(p|q)|r \overset{Bis}{\sim} p|(q|r)$
- ***Annullamento rispetto alla somma***: $p + Nil \overset{Bis}{\sim} p$
- ***Annullamento rispetto alla parallelizzazione***: $p|Nil \overset{Bis}{\sim} p$

Osservazione 5. *Per ogni coppia $p, q \in Proc_{CCS}$ vale:*

- *$LTS(p)$ isomorfo $LTS(q)$ implica $p \overset{Bis}{\sim} q$*
- *Si ha un'astrazione di stati.*
- *$p \overset{Bis}{\sim} q \Rightarrow p \overset{T}{\sim} q \Rightarrow \overset{Bis}{\sim} \subseteq \overset{T}{\sim}$*
- *È una congruenza tra gli operatori CCS.*
- *Preserva la stessa probabilità di deadlock (o assenza) nell'interazione con l'ambiente.*
- *$\overset{Bis}{\sim}$ è troppo restrittiva perché $a.b.Nil \overset{Bis}{\not\sim} a.\tau.b.Nil$. In sostanza $\overset{Bis}{\sim}$ e $\overset{T}{\sim}$ non astraggono le τ.*

2.3.2 Bisimulazione debole

La bisimulazione forte rischia di essere troppo restrittiva. Si passa quindi alla definizione di **equivalenza debole rispetto alle tracce** $\overset{T}{\approx}$ e **bisimulazione debole** $\overset{Bis}{\approx}$. La definizione di queste nuove relazioni obbliga a modificare la definizione della funzione di transizione. La relazione di transizione debole è definita come:

$$\Rightarrow \subseteq Proc_{CCS} \times Act \times Proc_{CCS} \tag{2.37}$$

Possiamo rappresentare tale funzione come $p \overset{\alpha}{\Rightarrow} p'$ dove $\alpha \in Act$ se e solo se:

- Se $\alpha = \tau$ allora posso eseguire una sequenza qualsiasi, anche nulla, di τ:

$$p \overset{\tau}{\rightarrow}^* p' \begin{cases} p = p' & \text{se non ci sono } \tau \text{ da eseguire} \\ p \overset{\tau}{\rightarrow} p_1 \overset{\tau}{\rightarrow} \ldots \overset{\tau}{\rightarrow} p' & \text{altrimenti} \end{cases} \tag{2.38}$$

- Se $\alpha \in A \cup \overline{A}$ allora vale: $p \overset{\tau}{\rightarrow}^* \overset{\alpha}{\rightarrow} \overset{\tau}{\rightarrow}^*$

Come fatto per la relazione forte, definiamo la relazione di transizione per sequenze di azioni $w \in Act^*$ come $p \overset{w}{\Rightarrow} p'$ se e solo se:

- Se $w = \varepsilon$ oppure $w = \tau^*$ allora ho:

$$p \overset{\tau}{\rightarrow}^* p' \tag{2.39}$$

- Se $w = a_1 \ldots a_n$ con $a_i \in A \cup \overline{A}$ allora:

$$p \overset{a_1}{\Rightarrow} p_1 \overset{a_2}{\Rightarrow} \ldots \overset{a_n}{\Rightarrow} p' \tag{2.40}$$

  dove ogni a_i può essere preceduto/seguito da una qualsiasi sequenza di τ.

Definizione 8 (**Equivalenza debole rispetto alle tracce**)**.** *Definiamo l'**equivalenza debole rispetto alle tracce**, la quale è rappresentata come $p \overset{T}{\approx} q$ se e solo se:*

$$tracce_{\Rightarrow}(p) = tracce_{\Rightarrow}(q) \text{ ovvero } \forall w \in (A \cup \overline{A}) \text{ ho che } p \overset{w}{\Rightarrow} \iff q \overset{w}{\Rightarrow} \tag{2.41}$$

quindi se i due processi possono eseguire la stessa sequenza di azioni.

Posso definire le tracce come:

$$tracce_{\Rightarrow}(p) = \{w \in (A \cup \overline{A})^* | p \overset{w}{\Rightarrow}\} \tag{2.42}$$

Osservazione 6. *In primis si ha che:*

$$p \overset{T}{\sim} q \Rightarrow p \overset{T}{\approx} q \Rightarrow \overset{T}{\sim} \subseteq \overset{T}{\approx} \tag{2.43}$$

Per ogni coppia $p, q \in Proc_{CCS}$ vale:

- *$LTS(p)$ isomorfo $LTS(q)$ implica $p \overset{T}{\approx} q$*
- *Si ha un'astrazione di stati.*
- *È una congruenza tra gli operatori CCS.*
- *Non preserva la stessa probabilità di deadlock (o assenza) nell'interazione con l'ambiente.*

Definizione 9 (**Bisimulazione debole**)**.** *Data una relazione R definita come $R \subseteq Proc_{CCS} \times Proc_{CCS}$. Dico che R è una relazione di **bisimulazione debole** se e solo se:*

$$\forall p, q \in Proc_{CCS} \text{ tale che } pRq \text{ vale che } \forall a \in Act \tag{2.44}$$

- *Se $p \xrightarrow{a} p'$ allora $\exists q'$ tale che $q \overset{a}{\Rightarrow} q'$ e $p'Rq'$*
- *E deve valere anche il viceversa: $q \xrightarrow{a} q'$ allora $\exists p'$ tale che $p \overset{a}{\Rightarrow} p'$ e $p'Rq'$*

Due processi p e q sono in relazione di bisimulazione debole $p \overset{Bis}{\approx} q$ se e solo se esiste una relazione di bisimulazione R tale che pRq si ha che vale:

$$\overset{Bis}{\approx} = \bigcup \{R | R \text{ è di bisimulazione debole}\} \tag{2.45}$$

Esempio 4. *Consideriamo i processi $r = a.(b.Nil + \tau.c.Nil)$ e $k = a.(b.Nil + \tau.c.Nil) + a.c.Nil$.*

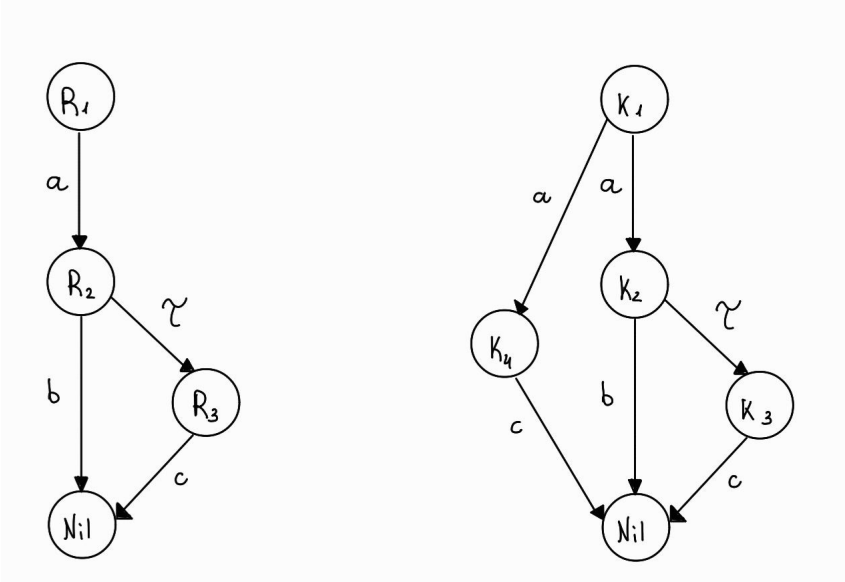


Figura 2.3: LTS di r e k

$$r \overset{Bis}{\approx} k$$

Definizione 10 (**Processo deterministico**)**.** *Sia $p \in Proc_{CCS}$ è un processo deterministico se e solo se vale che:*

$$\forall \alpha \in Act,\ p \xrightarrow{\alpha} p' \wedge p \xrightarrow{\alpha} p'' \Rightarrow p' = p'' \tag{2.46}$$

Osservazione 7. *Siano $p, q \in Proc_{CCS}$, se p, q sono deterministici e $p \overset{T}{\sim} q$ ($p \overset{T}{\approx} q$) allora $p \overset{Bis}{\sim} q$ ($p \overset{Bis}{\approx} q$).*

Osservazione 8. *Le proprietà della relazione di bisimulazione debole sono:*

- *È un'equivalenza.*

- *Preserva la possibilità di generare (o non generare) deadlock nell'interazione con l'ambiente (non ci sono problemi con i deadlock).*
- *Astraiamo dagli stati, azioni inosservabili (τ) e dai cicli.*

$$Nil \overset{Bis}{\approx} \tau.p \tag{2.47}$$

La bisimulazione debole **non** è una congruenza rispetto agli operatori CCS.

Teorema 3. *Se $p, q \in Proc_{CCS}$, $p \overset{Bis}{\approx} q$ allora:*

- $\alpha.p \overset{Bis}{\approx} \alpha.q, \forall \alpha \in A$
- $p|r \overset{Bis}{\approx} q|r \wedge r|p \overset{Bis}{\approx} r|q, \forall r \in Proc_{CCS}$
- $p[f] \overset{Bis}{\approx} q[f], \forall f : Act \to Act$ *funzione di rietichettatura.*
- $p_{\backslash L} \overset{Bis}{\approx} q_{\backslash L}, \forall L \subseteq Act$
- *Rispetto all'operatore $+$ e alla ricorsione questa cosa non è vera:*

$$\tau.a.Nil \overset{Bis}{\approx} a.Nil, \tau.a.Nil + b.Nil \overset{Bis}{\not\approx} a.Nil + b.Nil \tag{2.48}$$

Congruenza

Possiamo cercare la più grande relazione di congruenza $\overset{C}{\approx}$ che più si avvicina a $\overset{Bis}{\approx}$.

$$\overset{C}{\approx} \subseteq \overset{Bis}{\approx} \subseteq Proc_{CCS} \times Proc_{CCS} \tag{2.49}$$

Non sarà altro che una bisimulazione ristretta senza ricorsione e con processi (agenti) finiti. Per fare ciò dovremo definire un insieme finito di assiomi Ax che permetta di dimostrare la congruenza, tale che:

- Ax *corretto* ($Ax \vdash p = q \Rightarrow p \overset{C}{\approx} q$)
- Ax *completo* ($p \overset{C}{\approx} q \Rightarrow Ax \vdash p = q$)

1. **Legge associativa**:
$$p + (q + r) \overset{C}{\approx} (p + q) + r \text{ e } p|(q|r) \overset{C}{\approx} (p|q)|r \tag{2.50}$$
2. **Legge commutativa**:
$$p + q \overset{C}{\approx} q + p \text{ e } p|q \overset{C}{\approx} q|p \tag{2.51}$$
3. **Legge di assorbimento**:
$$p + p \overset{C}{\approx} p \text{ (ma } p|p \overset{C}{\not\approx} p) \tag{2.52}$$
4. $p + Nil \overset{C}{\approx} p$ e $p|Nil \overset{C}{\approx} p$
5. $p + \tau.p \overset{C}{\approx} \tau.p$
6. $\mu.\tau.p \overset{C}{\approx} \mu.p$
7. $\mu.(p + \tau.q) \overset{C}{\approx} \mu.(p + \tau.q) + \mu.q$
8. Se p e q sono delle somme: $p = \sum_i \alpha_i.p_i$ e $q = \sum_j \beta_j.q_j$, $\alpha, \beta \in Act$:
$$p|q \overset{C}{\approx} \sum_i \alpha_i(p_i|q) + \sum_j \beta_j.(p|q_j) + \sum_{\alpha_i = \overline{\beta}_j} \tau.(p_i|q_j)$$

(**Teorema di espansione di R. Milner**) Questo assioma mi permette di rappresentare la composizione parallela come la somma di tutte le possibili alternative che si hanno con l'operazione di composizione parallela.

9. $p[f] \overset{C}{\approx} \sum_i f(\alpha_i).(p_i[f])\ \forall f$ funzione di etichettatura.
10. $p_{\setminus L} \overset{C}{\approx} \sum_{\alpha_i, \overline{\alpha}_i \notin L} \alpha_i.(pi_{\setminus L})\ \forall L \subseteq A$

Il problema della bisimulazione è che confronta i processi in base alle azioni che devono essere atomiche, se dovessimo raffinare un'azione in 2 più piccole allora le equivalenze di bisimulazione perdono effetto.

Dal momento che gli LTS hanno un esecuzione non deterministica e questo non permette di evidenziare le dipendenze o le indipendenze, per questo sono state create le reti di petri.

Gioco verifica bisimulazione debole

Per confrontare due processi p e q si può utilizzare un gioco $G(p, q)$ con 2 giocatori:

- **Attaccante**: il quale cerca di dimostrare che $p \overset{Bis}{\not\approx} q$
- **Difensore**: il quale cerca di dimostrare che $p \overset{Bis}{\approx} q$

Un gioco è composto da più partite, dove ogni partita è una sequenza finita o infinita di configurazioni:

$$(p_0, q_0), (p_1, q_1), \ldots, (p_i, q_i), \ldots$$

In ogni mano si passa dalla configurazione corrente (p_i, q_i) alla successiva (p_{i+1}, q_{i+1}) con le seguenti regole:

- L'Attaccante sceglie uno dei due processi della configurazione corrente (p_i, q_i) e fa una $\xrightarrow{\alpha}$ mossa $(\alpha \in Act)$
- Il Difensore deve rispondere con una $\overset{\alpha}{\Rightarrow}$ mossa nell'altro processo.

La coppia di processi (p_{i+1}, q_{i+1}) così ottenuta diventa la nuova configurazione corrente. La partita continua con un'altra mano.

La partita può terminare in uno dei due seguenti modi:

- Se un giocatore non può muovere, l'altro vince.
- Se la partita è infinita, vince il difensore.

Diverse partite possono concludersi con vincitori diversi, ma per ogni gioco, un solo giocatore può vincere ogni partita.

Una **strategia** per un giocatore è un insieme di regole che indicano di volta in volta che mossa fare. Tali regole dipendono solo dalla configurazione corrente. Un giocatore ha una **strategia vincente** per un gioco $G(p, q)$ se seguendo quella strategia è in grado di vincere tutte le partite del gioco.

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

- *L'Attaccante ha una strategia vincente per $G(p, q)$ se e solo se $p \overset{Bis}{\not\approx} q$.*
- *Il Difensore ha una strategia vincente per $G(p, q)$ se e solo se $p \overset{Bis}{\approx} q$.*

Nota 1. *Il gioco della bisimulazione può essere usato sia per dimostrare che due processi sono bisimili, che per dimostrare che non lo sono.*

Per dimostrare che i processi sono bisimili, bisogna mostrare che il Difensore ha una strategia vincente, cioè che, per ogni mossa dell'Attaccante, il Difensore ha almeno una mossa che lo porterà a vincere.

Per dimostrare che i processi **non** sono bisimili, bisogna mostrare che l'Attaccante ha una strategia vincente, cioè che, in ogni configurazione, l'Attaccante è in grado di scegliere su quale processo operare e con quale azione, in modo che per ogni successiva mossa del Difensore, l'Attaccante ha almeno una mossa che lo porterà a vincere.

Capitolo 3

Reti di Petri

Abbiamo introdotto un'algebra di processi come CCS dove processi sequenziali interagiscono tra loro tramite hand-shaking. Un altro modello usato, con varie implementazioni, sono gli automi a stati finiti. Si passa ora alle reti di Petri.

La critica di Petri è che in un sistema distribuito non sia individuabile uno stato globale, che in un sistema distribuito le trasformazioni di stato siano localizzate e non globali, che non esista un sistema di riferimento temporale unico. Quindi la simulazione sequenziale non deterministica (semantica a "interleaving") dei sistemi distribuiti è una forzatura e non rappresenta le reali caratteristiche del comportamento del sistema, ovvero la località, la distribuzione degli eventi e la relazione di dipendenza causale e non causale tra gli eventi.

Nel modello proposto da Petri, le azioni vengono rappresentate come nodi dell'automa e non più come etichette della transizione. Oltre a ciò, le azioni e le coazioni che abbiamo definito in CCS per sincronizzare due processi diventano una singola azione di sincronizzazione.

Petri sviluppò una teoria matematica fondata sui principi della fisica moderna, che sia una teoria dei sistemi in grado di descrivere il flusso di informazione e permetta di analizzare sistemi con organizzazione complessa.

Lo **stato** è definito da una collezione di stati locali.

3.1 Reti elementari

Definizione 11 (**Rete**). *Una **rete** è definite come:*

$$N = (B, E, F) \tag{3.1}$$

dove:

- *B è un insieme finito di condizioni, anche detti stati locali, proposizioni vere o false. Rappresentato da:*

  $\bigcirc$

- *E è un insieme finito di eventi, **trasformazioni locali** di stato. Rappresentato da:*

  $\square$

  $$B \cap E = \emptyset \wedge B \cup E \neq \emptyset \tag{3.2}$$

- *$F \subseteq (B \times E) \cup (E \times B)$ è una **relazione di flusso** Rappresentato da:*

  $\rightarrow$

  Inoltre, la relazione di flusso è tale per cui non esistano elementi isolati, in quanto non avrebbero senso. Si ha, formalmente, che:

  $$dom(F) \cup ran(F) = B \cup E \tag{3.3}$$

  ovvero non ho condizioni/eventi isolati, in quanto non avrebbero senso, avrei una condizione costante e un evento che non accade mai; quindi, dominio e codominio di F coprono l'insieme di condizioni ed eventi.

Sia $x \in X$ dove l'insieme X è definito come $X = B \cup E$, allora possiamo definire:

- ${}^\bullet x = \{y \in X : \ (y,x) \in F\}$ sono i **pre-elementi** di x. Posso anche definirli come precondizioni o pre-eventi.
- $x^\bullet = \{y \in X : \ (x,y) \in F\}$ sono i **post-elementi** di x. Posso anche definirli come post-condizioni o post eventi.

Sia $A \subseteq B \cup E$ allora posso definire:

- ${}^\bullet A = \bigcup_{x \in A} {}^\bullet x$
- $A^\bullet = \bigcup_{x \in A} x^\bullet$

Nelle reti c'è sempre una relazione di dualità tra due elementi, per esempio tra condizioni ed eventi, tra pre-eventi e post-eventi, tra precondizioni e post condizioni. Inoltre, si ha la caratteristica della località, quindi si hanno stati locali e trasformazioni di stato locali.

La rete $N = (B, E, F)$ descrive la struttura statica del sistema, il comportamento è definito attraverso le nozioni di caso e di regola di scatto o regola di transizione.

Un **caso** o **configurazione** è un insieme di condizioni $c \subseteq B$ che rappresentano l'insieme di condizioni vere in una certa configurazione del sistema, un insieme di stati locali che collettivamente individuano lo *stato globale* del sistema.

- Condizione vera:

- Condizione falsa:

Definizione 12 (**Regola dello scatto**). *Sia $N = (B, E, F)$ una rete elementare e $c \subseteq B$. L'evento $e \in E$ è **abilitato**, ovvero può occorrere, in c, denotato $c[e >$, se e solo se:*

$$ {}^\bullet e \subseteq c \ \wedge \ e^\bullet \cap c = \emptyset \tag{3.4} $$

Se $c[e >$, allora quando e occorrere in c genera un nuovo caso c', denotato $c[e > c'$:

$$ c' = (c -^\bullet e) \cup e^\bullet \tag{3.5} $$

In altre parole, un evento e è abilitato se le sue precondizioni sono vere, le post-condizioni false. Lo scatto di e rende le precondizioni false e le post-condizioni vere, le altre condizioni rimangono inalterate.

Le reti si basano sul **principio di estensionalità**, ovvero sul fatto che il cambiamento di stato è locale:

Un evento è completamente caratterizzato dai cambiamenti che produce negli stati locali, tali cambiamenti sono indipendenti dalla particolare configurazione in cui l'evento occorre.

Definizione 13. *Sia $N = (B, E, F)$ una **rete elementare**, si ha che:*

- *N è **semplice** se e solo se:*

$$ \forall x, y \in B \cup E, \ {}^\bullet x = {}^\bullet y \ \wedge \ x^\bullet = y^\bullet \Rightarrow x = y \tag{3.6} $$

- *N è **pura** se e solo se:*

$$ \forall e \in E : \ {}^\bullet e \cap e^\bullet = \emptyset \tag{3.7} $$

Definizione 14. *Sia $N = (B, E, F)$ una rete elementare, $U \subseteq E$ e $c, c_1, c_2 \subseteq B$.*

- *U è un **insieme di eventi indipendenti** se e solo se:*

$$ \forall e_1, e_2 \in U : \ e_1 \neq e_2 \Rightarrow ({}^\bullet e_1 \cup e_1^\bullet) \cap ({}^\bullet e_2 \cup e_2^\bullet) = \emptyset \tag{3.8} $$

- *U è un **passo abilitato** (insieme di eventi concorrenti) in c anche scritto come $c[U >$ se e solo se:*

$$ U \text{ insieme di eventi indipendenti } \wedge \ \forall e \in U : \ c[e > \tag{3.9} $$

- *U è un **passo** da c_1 a c_2, anche scritto come $c_1[U > c_2$ se e solo se:*

$$ c_1[U > \ \wedge \ c_2 = (c_1 - {}^\bullet U) \cup U^\bullet \tag{3.10} $$

Se due eventi sono indipendenti allora possono eseguire in modo concorrente.

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

Definizione 15 (**Caso raggiungibile**). *L'insieme dei* ***casi raggiungibili*** *(C_Σ) del sistema elementare $\Sigma = (B, E, F; c_{in})$ è il più piccolo sottoinsieme di 2^B tale che:*

- $c_{in} \in C_\Sigma$
- *Se $c \in C_\Sigma, U \subseteq E, c' \subseteq B$ sono tali che: $c[U > c'$ allora $c' \in C_\Sigma$*

L'insieme C_Σ è un insieme **finito** perché si parla di sottoinsiemi di insiemi finiti che sono anch'essi finiti, ovviamente esplode in un esponenziale.

Definizione 16 (**Insieme dei passi**). *U_Σ è l'**insieme dei passi** di Σ:*

$$U_\Sigma = \{U \subseteq \ E | \exists c, c' \in C_\Sigma : c[U > c'\} \tag{3.11}$$

Sia $\Sigma = (B, E, F; c_{in})$ un sistema elementare, $c_i \in C_\Sigma, e_i \in E, U_i \subseteq E$ possiamo distinguere:

- **Comportamento sequenziale**: sequenze di occorrenze o di eventi ("interleaving", simulazione sequenziale non deterministica), ovvero una sequenza di eventi che possono occorrere dal caso iniziale, facendo scattare in maniera sequenziale gli eventi uno alla volta in c_n:

$$c_{in}[e_1 > c_1[e_2 > \ldots [e_n > c_n \text{ oppure } c_{in}[e_1 e_2 \ldots e_n > c_n \tag{3.12}$$

- **Comportamento non sequenziale** (semantica a passi): sequenze di passi ("step semantics") in quanto possiamo anche considerare insiemi di eventi, ovvero passi:

$$c_{in}[U_1 > c_1[U_2 > \ldots [U_n > c_n \text{ oppure } c_{in}[U_1 U_2 \ldots U_n > c_n \tag{3.13}$$

- **Comportamento non sequenziale** (semantica di ordine parziale): processi non sequenziali ("partial order semantics" - "true concurrency"). Il comportamento di tale sistema viene registrato in una rete di Petri.

Si considerano sia sequenze finite che sequenze infinite (di eventi o di passi).

Il comportamento di un sistema elementare $\Sigma = (B, E, F; c_{in})$ può essere rappresentato dal suo grafo dei casi.

Definizione 17 (**Grafo dei casi**). *Il **grafo dei casi** di Σ è il sistema di transizioni etichettato $CG_\Sigma = (C_\Sigma, U_\Sigma, A, c_{in})$ dove:*

- *C_Σ è l'insieme dei nodi del grafo (gli stati globali).*
- *U_Σ è l'alfabeto.*
- *A è l'insieme di archi etichettati:*

$$A = \{(c, U, c') | c, c' \in C_\Sigma, U \in U_\Sigma, c[U > c'\} \tag{3.14}$$

I nodi sono i casi sequenziali mentre gli archi sono gli insiemi di eventi indipendenti

Definizione 18 (**Grafo dei casi sequenziale**). *Il **grafo dei casi sequenziali** di $\Sigma = (B, E, F; c_{in})$ è il sistema di transizioni etichettato $SCG_\Sigma = (C_\Sigma, E, A, c_{in})$ dove:*

$$A = \{(c, e, c') | c, c' \in C_\Sigma, e \in E : c[e > c'\} \tag{3.15}$$

I nodi sono i casi sequenziali mentre gli archi sono gli eventi

3.1.1 Diamond property

Definizione 19 (**Diamond property**). *Sia $\Sigma = (B, E, F; c_{in})$ un sistema elementare, $CG_\Sigma = (C_\Sigma, U_\Sigma, A, c_{in})$ il suo grafi dei casi, $U_1, U_2 \in U_\Sigma : U_1 \cap U_2 = \emptyset, U_1 \neq \emptyset, U_2 \neq \emptyset$, e $c_1, c_2, c_3, c_4 \in C_\Sigma$, allora valgono:*

1. *Dato $c_1[e_1 >$ e $c_1[e_2 >$ segue che:*

$$^\bullet e_1 \cap e_2^\bullet = \emptyset \ \wedge \ ^\bullet e_2 \cap e_1^\bullet = \emptyset \tag{3.16}$$

infatti, se e_1 e e_2 sono entrambi abilitati in c_1, le loro precondizioni sono vere e le post-condizioni false, e quindi non è possibile che una condizione sia contemporaneamente precondizione di e_1 (vera) e anche post-condizione di e_2 (falsa), e viceversa.

Da $c_1[e_1 > c_2[e_2 >$ segue:

$$^\bullet e_1 \cap^\bullet e_2 = \emptyset \ \wedge \ e_1^\bullet \cap e_2^\bullet = \emptyset \tag{3.17}$$

in c_2, infatti, le precondizioni di e_1 sono false mentre le precondizioni di e_2 sono vere e quindi e_1 e e_2 non possono avere precondizioni in comune. Inoltre, sempre in c_2 le post-condizioni di e_1 sono vere, mentre quelle di e_2 sono false, e quindi e_1 e e_2 non possono avere post-condizioni in comune. Segue quindi 3.1.

Figura 3.1: Diamond property 1

2. *Supponiamo che $U_1 \cup U_2 \in U_\Sigma$ e che $U_1 \cap U_2 = \emptyset$, $U_1 \neq \emptyset, U_2 \neq \emptyset$. Allora se $c_1[U_1 \cup U_2 > c_3$ sicuramente $c_1[U_1 >$ e $c_1[U_2 >$ e anche: $c_1[U_1 > c_2[U_2 > c_3$ e $c_1[U_2 > c_4[U_1 > c_3$. Segue quindi 3.2.*

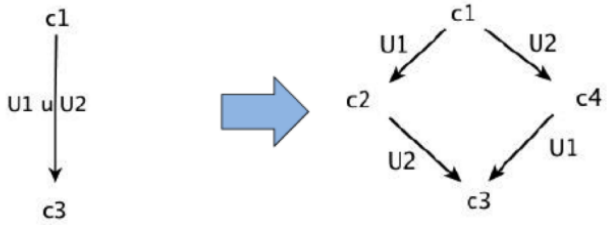


Figura 3.2: Diamond property 2

Per la *Diamond property*, nei sistemi elementari il grafo dei casi e il grafo dei casi sequenziale sono *sintatticamente equivalenti*, ovvero possono essere ricavati l'uno dall'altro.

Questo implica il fatto che due sistemi elementari hanno grafi dei casi isomorfi se e solo se hanno grafi dei casi sequenziale isomorfi.

Definizione 20 (**Equivalenza tra sistemi**). *Due sistemi Σ_1 e Σ_2 sono **equivalenti** se e solo se hanno grafi dei casi sequenziali, e quindi anche grafi dei casi, **isomorfi**.*

Definizione 21 (*Problema della sintesi*). *Dato un sistema di transizioni etichettato $A = (S, E, T, s_0)$, stabilire se esiste un sistema elementare $\Sigma = (B, E, F; c_{in})$ tale che: il suo grafo dei casi SCG_Σ sia isomorfo ad A. E, in caso affermativo, costruire Σ.*

Questo problema è stato risolto usando la teoria delle regioni. Oltre a ciò, A dovrà soddisfare la Diamond property.

Definizione 22 (**Contatto**). *Sia $\Sigma = (B, E, F; c_{in})$ un sistema elementare, $e \in E$, $c \in C_\Sigma$ allora (e, c) è un **contatto** se e solo se:*

$$^\bullet e \subseteq c \ \wedge \ e^\bullet \cap c \neq \emptyset \tag{3.18}$$

Definizione 23 (**Sistema senza contatti**). *Un sistema elementare* $\Sigma = (B, E, F; c_{in})$ *è* ***senza contatti*** *se e solo se:*

$$\forall e \in E,\ \forall c \in C_\Sigma \text{ si ha } {}^\bullet e \subseteq c \Rightarrow e^\bullet \cap c = \emptyset \tag{3.19}$$

È possibile trasformare un sistema elementare Σ con contatti in un sistema elementare Σ_0 che sia senza contatti aggiungendo a Σ il complemento di ogni condizione si ottiene, un sistema Σ_0 con grafo dei casi isomorfo a quello di Σ. Quindi se un sistema elementare Σ è senza contatti allora per verificare che un evento e sia abilitato

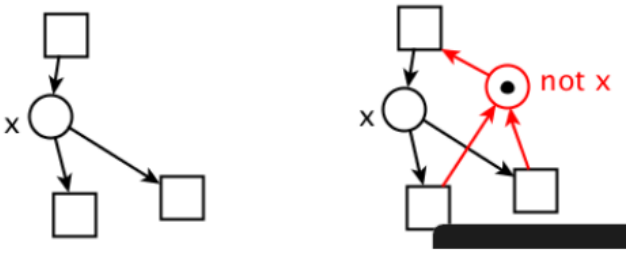


Figura 3.3: Complemento dell'operazione x.

in un caso raggiungibile c è sufficiente verificare che le precondizioni di e siano vere:

$$c[e > \text{ se e solo se } {}^\bullet e \subseteq c \tag{3.20}$$

Definizione 24 (**Sequenza**). *Sia* $\Sigma = (B, E, F; c_{in})$ *un sistema elementare,* $c \in C_\Sigma$ $e_1, e_2, \in E$, *allora* e_1 *ed* e_2 *sono in* ***sequenza*** *in* c *se e solo se:*

$$c[e_1 > \ \wedge \neg c[e_2 > \ \wedge\ c[e_1 e_2 > \ \ (c[e_1 > c'[e_2 >) \tag{3.21}$$

In altre parole, è presente una relazione di dipendenza causale tra e_1 *ed* e_2.

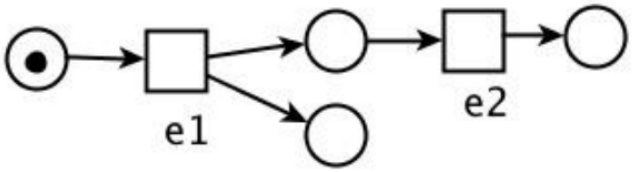


Figura 3.4: Rappresentazione della sequenza

Definizione 25 (**Concorrenti**). *Sia* $\Sigma = (B, E, F; c_{in})$ *un sistema elementare,* $c \in C_\Sigma$ $e_1, e_2, \in E$, *allora* e_1 *ed* e_2 *sono* ***concorrenti*** *in* c *se e solo se:*

$$c[\{e_1, e_2\} > \tag{3.22}$$

In altre parole, se e sol se e_1 *ed* e_2 *sono indipendenti ed entrambi abilitati in* c.

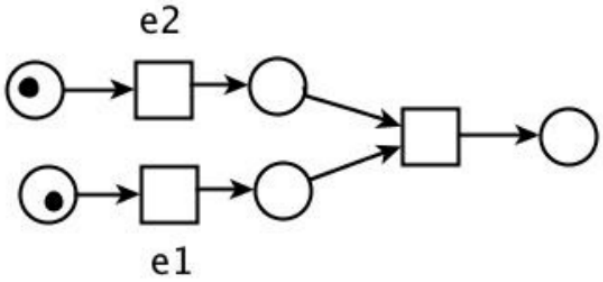


Figura 3.5: Rappresentazione della concorrenza

Figura 3.6: Rappresentazione di un conflitto

Definizione 26 (**Conflitto**). *Sia $\Sigma = (B, E, F; c_{in})$ un sistema elementare, $c \in C_\Sigma$ $e_1, e_2, \in E$, allora e_1 ed e_2 sono in **conflitto** in c se e solo se:*

$$c[e_1 > \ \wedge \ c[e_2 > \ \wedge \neg c[\{e_1, e_2\} > \tag{3.23}$$

In altre parole, sono entrambi abilitati ma l'occorrenza di uno disabilita l'altro.

Si definisce **confusione** quando non è possibile stabilire se è stato risolto un conflitto.

Definizione 27 (**Sottorete**). *Siano $N = (B, E, F)$ e $N_1 = (B_1, E_1, F_1)$ due reti elementari. Diciamo che:*

- $N_1 = (B_1, E_1, F_1)$ *è **sottorete** di N se e solo se:*
  - $B_1 \subseteq B$
  - $E_1 \subseteq E$
  - $F_1 = F \cap [(B_1 \times E_1) \cup (E_1 \times B_1)]$
- $N_1 = (B_1, E_1, F_1)$ *è **sottorete generata da** B_1 se e solo se:*
  - $B_1 \subseteq B$
  - $E_1 \subseteq^\bullet B_1 \cup B_1^\bullet$
  - $F_1 = F \cap [(B_1 \times E_1) \cup (E_1 \times B_1)]$
- $N_1 = (B_1, E_1, F_1)$ *è **sottorete generata da** E_1 se e solo se:*
  - $B_1 \subseteq^\bullet E_1 \cup E_1^\bullet$
  - $E_1 \subseteq E$
  - $F_1 = F \cap [(B_1 \times E_1) \cup (E_1 \times B_1)]$

Data una rete $N = (B, E, F, c_0)$ questa può essere ottenuta componendo altre reti di Petri. Si hanno in letteratura 3 modi principali:

1. La composizione **asincrona**: si sincronizzano le reti con uno stato
2. La composizione **sincrona**: si sincronizzano le reti con un'azione
3. La composizione **mista** tra sincrona e asincrona.

3.2 Processi non sequenziali

Prendiamo l'esempio di una rete ciclica per spegnere il fuoco (fig 3.7). In questo sistema si hanno 4 persone che spostano 4 secchi, le persone non possono oltrepassare le altre persone, questo significa che quando si incontrano dovranno scambiarsi il secchio vuoto con quello pieno. A sinistra della rete c'è il lago dal quale si riempie il secchio vuoto, a destra c'è il fuoco da spegnere. Possiamo essere interessati a tutta la storia dei movimenti dei secchi.

Definizione 28 (**Rete causale** o **Rete di occorrenze senza conflitti**). *Definiamo $N = (B, E, F)$ come una **rete causale**, detta anche rete di occorrenze senza conflitti, se e solo se:*

- $\forall b \in B : |^\bullet b| \leq 1 \wedge |b^\bullet| \leq 1$ *ovvero non si hanno conflitti; quindi, per ogni condizione si ha al più un pre-evento e un post evento. (Avendo quindi al più un arco entrante e al più uno uscente).*

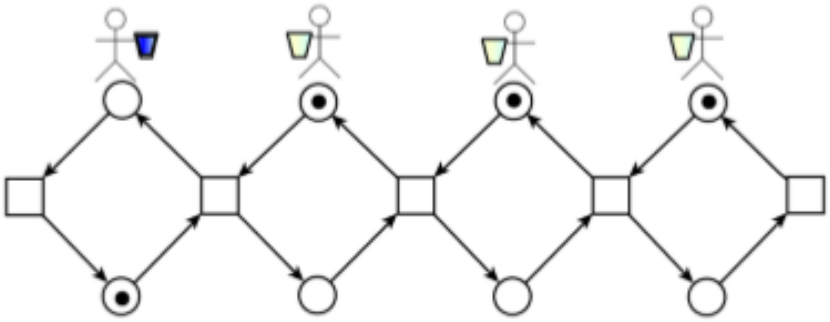


Figura 3.7: Esempio di una rete per modellare il sistema di spegnimento del fuoco

- $\forall x, y \in B \cup E : (x, y) \in F^+ \Rightarrow (y, x) \notin F^+$ *ovvero non si hanno cicli; quindi, presi due elementi collegati da una sequenza di archi orientati, avendo un cammino tra i due elementi (F^+ è la chiusura transitiva della relazione F) non ho anche un cammino opposto tra i due.*
- $\forall e \in E : \{x \in B \cup E | x F^* e\}$ *è finito, ovvero si ha un numero finito di pre-elementi di un certo elemento.*

Sono quindi reti che registrano un comportamento e quindi non si hanno conflitti (che in caso sono sciolti registrando solo quello che è effettivamente successo e non quello che potrebbe succedere). Si registra una run del sistema. Non si hanno nemmeno cicli perché ogni ripetizione dell'evento viene concatenata a quella prima.

La rete può quindi essere infinita, ma è composta da un insieme di elementi finito che si ripete. In ogni caso il passato di un evento è finito e registrato, anche se nel complesso il comportamento è infinito "in avanti". Con una rete causale si possono non distinguere più condizioni ed eventi.

Ad una rete causale è possibile associare un **ordine parziale**:

$$(X, \leq) = (B \cup E, F^*) \tag{3.24}$$

Dicendo che un elemento "è minore" di un altro se esiste un cammino orientato dall'uno all'altro.

Definizione 29. *Data una rete causale $N = (B, E, F)$ e dato un ordine parziale $(X, \leq)$ con $X = B \cup E$ si ha che si può interpretare la relazione d'ordine come indipendenza o dipendenza causale, ovvero presi $x, y \in X$ come elementi che occorrono nella storia di $X = B \cup E$ si hanno le seguenti diciture:*

- *$x \leq y$ (avendo un cammino da x a y) corrisponde a x causa y, ovvero si ha una relazione di dipendenza causale tra i due.*
- *x **li** y indica che $x \leq y \vee y \leq x$ e quindi corrisponde a x e y sono causalmente dipendenti. Si ha che **li** può venire letto come linea (x in linea con y) avendo che uno dei due precede l'altro.*
- *x**co** y indica che: $\neg(x < y) \ \wedge \ \neg(y < x)$ e quindi corrisponde a x e y sono **causalmente indipendenti**, avendo che i due elementi non si precedono a vicenda, non avendo ordine tra loro. Si ha che **co** sta per concurrency.*

*Le relazioni **li** e **co** sono riflessive e simmetriche ma non transitive.*

Definizione 30. *Data una rete causale $N = (B, E, F)$ e dato un ordine parziale $(X, \leq)$ con $X = B \cup E$ definiamo:*

$$C \subseteq X \tag{3.25}$$

come:

- ***co-set** se e solo se $\forall x, y \in C$: x **co** y, quindi C è una clique della relazione **co**.*
- ***taglio** se e solo se C è un **co-set** massimale (tutti gli elementi nel taglio sono in relazione **co**)*

*Definiamo C come **co-set** massimale se e solo se $\forall y \in X \backslash C$ si ha che:*

$$\exists c \in C : y \textbf{li} \ c \tag{3.26}$$

Definiamo:

$$L \subseteq X \tag{3.27}$$

come:

- ***li-set*** *se e solo se* $\forall x, y \in L: \ x \ \textbf{\textit{li}} \ y$
- ***linea*** *se e solo se* L *è un* ***li-set*** *massimale.*

Definiamo L *come* ***li-set*** *massimale se e solo se* $\forall y \in X \backslash L$ *si ha che:*

$$\exists l \in L: \ y \ \textbf{\textit{co}} \ l \tag{3.28}$$

Si ha quindi che:

- *In un* ***co-set*** *la relazione* ***co*** *è transitiva.*
- *In un* ***li-set*** *la relazione* ***li*** *è transitiva.*

Tagli e linee possono essere fatti sia di condizioni che di eventi.

Un taglio $C \subseteq X$ è detto B-taglio se $C \subseteq B$.
I tagli fatti di sole condizioni rappresentano casi raggiungibili dal sistema.
Una rete causale, quindi, registra il comportamento di un sistema elementare. Si hanno quindi, con i tagli, possibili osservazioni di configurazioni possibili nella storia del sistema.

Definizione 31. *Grazie alle reti causali, preso un elemento* $x \in X$*, possiamo definire:*

- ***past***(x)*, ovvero il passato dell'elemento, tutti gli elementi in relazione* $\leq$ *di* x*.*
- ***future***(x)*, ovvero il futuro dell'elemento, tutti gli elementi in relazione* $\geq$ *di* x*.*

Gli elementi nell'anti-cono sono in relazione ***co*** *con* x *e quindi possono essere concorrenti.*

Definizione 32 (k-densità). $N = (B, E, F)$ *rete causale,* $(X = (B \cup E), \leq)$ *ordine parziale. Si ha che* N *è* $k-$***densa*** *se e solo se:*

$$\forall h \in Linee(N), \ \forall c \in Tagli(N): \ |h \cap c| = 1 \tag{3.29}$$

dove $Linee(N)$ *e* $Tagli(N)$ *sono gli insiemi delle linee e dei tagli di* N*.*

Nota 2. *Se la rete causale* N *è finita allora* N *è anche* $K-$*densa.*

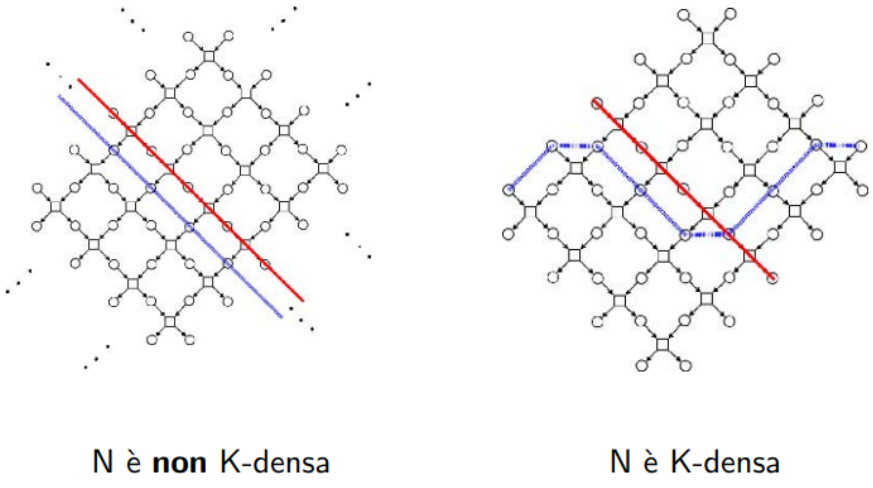


Figura 3.8: Esempio di rete K-densa

Definizione 33 (Processi non sequenziali). *Sia* $\Sigma = (S, T, F; c_{in})$ *un sistema elementare senza contatti e finito, ovvero con* $S \cup T$ *finito.*
$\langle N = (B, E, F); \phi \rangle$ *è un* ***processo non sequenziale*** *su* Σ *se e solo se:*

- (B, E, F) *è una* ***rete causale*** *nella quale si ammettono* ***condizioni isolate****.*

- $\phi : B \cup E \to S \cup T$ *è una mappa tale che:*
  1. $\phi(B) \subseteq S,\ \phi(E) \subseteq T$
  2. $\forall x_1, x_2 \in B \cup E :\ \phi(x_1) = \phi(x_2) \Rightarrow (x_1 \leq x_2)\ \vee\ (x_2 \leq x_1)$
  3. $\forall e \in E :\ \phi({}^\bullet e) = {}^\bullet \phi(e)\ \wedge\ \phi(e^\bullet) = \phi(e)^\bullet$
  4. $\phi(Min(N)) = c_{in}$ *dove* $Min(N) = \{x \in B \cup E |\ \nexists y :\ (y, x) \in F\}$ *ovvero non hanno un arco entrante, sono gli stati locali iniziali.*

Se $\langle N = (B, E, F); \phi \rangle$ è un **processo non sequenziale** di $\Sigma = (S, T, F, c_{in})$ **sistema elementare finito e senza contatti** allora:

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

I B-tagli corrispondono ai casi raggiungibili perché sono contemporaneamente veri.

Nota 3. *In sostanza per accorgersi subito se è un processo sequenziale allora:*

- *devono comparire tutti gli stati del caso iniziale*
- *se c'è un evento allora devono essere rappresentate tutte le sue precondizioni e postcondizioni*
- *non ci devono essere conflitti nel processo non sequenziale*

3.2.1 Reti di Occorrenze

Vogliamo avere una struttura che registri tutti i comportamenti possibili registrando le dipendenze e le indipendenze tra i processi. Posso avere una struttura che mi raccolga tutti i possibili processi sequenziali.

Definizione 34 (**Relazione di conflitto**). *Una* ***relazione di conflitto*** $\# \subseteq (B \cup E) \times (B \cup E)$ *è definita come*

$$x \# y \iff \exists e_1, e_2 \in E :^\bullet e_1 \cap^\bullet e_2 \neq \emptyset \wedge e_1 \leq x \wedge e_2 \leq y \tag{3.30}$$

due elementi sono in relazione se nel loro passato hanno 2 eventi in conflitto.

Definizione 35 (**Rete di occorrenze, si ammettono conflitti**). $N = (B, E, F)$ *è una* ***rete di occorrenze*** *se e solo se:*

- ***conflitti solo in avanti****:* $\forall b \in B :\ |{}^\bullet b| \leq 1$ *sono presenti dei conflitti solo in avanti.*
- ***aciclica****:* $\forall x, y \in B \cup E :\ (x, y) \in F^+ \Rightarrow (y, x) \notin F^+$ *non ci sono cicli*
- ***passato finito****:* $\forall e \in E : \{x \in B \cup E |\ xF^* e\}$ *è finito*
- ***relazione di conflitto*** $\#$ *non è riflessiva.*

In queste reti è ancora possibile associare a N un ordine parziale $(X, \leq) = (B \cup E, F^*)$:

- x **li** y sse $x \leq y$ o $y \leq x$
- $x \ \#\ y$ sse $\exists e_1, e_2 \in E :^\bullet e_1 \cap^\bullet e_2 \neq \emptyset \wedge e_1 \leq x \wedge e_2 \leq y$
- x **co** y sse $\neg(x < y)$ e $\neg(y < x)$ e $\neg x \# y$

Se i due elementi non sono ordinati allora saranno o in relazione *co* o in **conflitto**.

Nota 4. *La proprietà 2 e 3 delle reti causali in and tra loro implicano la proprietà 3 delle reti di occorrenze. Un ragionamento analogo può essere fatto per le proprietà 2 e 4 delle reti causali, le quali in and implicano la proprietà 4 delle reti di occorrenza.*

Nota 5. *Un processo non sequenziale può essere definito anche richiedendo che ϕ soddisfi: (1),(2),(3') e (4'), dove 3' e 4' implicano le proprietà 3 e 4 delle reti di occorrenza.*

Definizione 36 (**Processo ramificato**). *Sia $\Sigma = (S, T, F, c_{in})$ un sistema elementare senza contatti e finito. $\langle N = (B, E, F); \phi \rangle$ è un* ***processo ramificato*** *di Σ se e solo se:*

- (B, E, F) *è una rete di occorrenze (si ammettono condizioni isolate)*
- $\phi : B \cup E \to S \cup T$ *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. $\forall e \in E$ *: la restrizione di* ϕ *a* ${}^\bullet e$ *è una biiezione tra* ${}^\bullet e$ *e* ${}^\bullet\phi(e)$ *e la restrizione di* ϕ *a* $e^\bullet$ *è una biiezione tra* $e^\bullet$ *e* $\phi(e)^\bullet$
  4. *La restrizione di* ϕ *a* $Min(N)$ *è una biiezione tra* $Min(N)$ *e* c_{in}*.*

Definizione 37 (**Prefisso**)**.** *Sia* $\Sigma = (S, T, F, c_{in})$ *un sistema elementare finito e senza contatti e siano* $\Pi_1 = \langle N_1; \phi_1\rangle$*,* $\Pi_2 = \langle N_2; \phi_2\rangle$ *processi ramificati di* Σ*.*

Allora $\Pi_1 = \langle N_1; \phi_1\rangle$ *è un* ***prefisso*** *di* $\Pi_2 = \langle N_2; \phi_2\rangle$ *se e solo se* N_1 *è una sottorete di* N_2 *e* $\phi_{2|N_1} = \phi_1$ *(*ϕ_2 *"ristretto" a* N_1 *è uguale a* ϕ_1*).*

Nota 6. *Il prefissi di un processo ramificato sono i processi sequenziali finiti del del sistema elementare.*

Definizione 38 (**Unfolding**)**.** Σ*, sistema elementare, ammette un unico processo ramificato che è massimale rispetto alla relazione di prefisso tra processi. Tale processo massimale è chiamato* ***unfolding*** *di* Σ*, denotato* $Unf(\Sigma)$*.*

Definizione 39 (**Corsa**)**.** *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*** *(****run****).*

Osservazione 9. *Inoltre, ogni processo non sequenziale di* Σ *è un prefisso dell'unfolding* $Unf(\Sigma)$*.*

3.3 Sistemi elementari - reti P/T- Reti ad alto livello

Se dovessimo usare le reti elementari per modellare sistemi veri, avremmo un'esplosione di condizioni ed eventi. Una rappresentazione più compatta è data dalle **reti Posti e Transizioni** che utilizzano, invece delle condizioni booleane dei sistemi elementari, dei contatori.

Definizione 40 (**Sistema Posti e Transizioni**)**.** *Formalmente un sistema posti e transizioni è definito come:*

$$\Sigma = (S, T, F, K, W, M_0) \tag{3.31}$$

dove:

- (S, T, F) *è una rete.*
- ***funzione capacità degli stati****:* $K : S \to \mathbb{N}^+ \cup \{\infty\}$ *è la funzione che assegna ai posti le capacità*
- ***funzione peso degli archi****:* $W : F \to \mathbb{N}$ *è la funzione peso degli archi*
- ***funzione di marcatura iniziale****:* $M_0 : S \to \mathbb{N} : \forall s \in S\ M_0(s) \leq K(s)$ *è la marcatura iniziale. È importante osservare che il numero di marche è sempre minore o uguale alla capacità del posto*

Ad esempio, pensiamo ad un buffer a due posizioni, nei sistemi elementari avremmo due condizioni con due eventi deposita e preleva. Nei sistemi P/T possiamo usare un contatore come fosse un'unica condizione. Chiaramente si perde dell'informazione, in particolare non sappiamo più in quale posizione del buffer il produttore deposita e da quale posizione il consumatore consuma.

Questa tipologia di reti mi permette di definire degli archi pesati, archi con associato un numero, che abilitino o meno le transizioni.

Le reti ad alto livello, anche dette **reti colorate**, permettono di ristabilire l'informazione persa con le reti Posti e transizioni assegnando una struttura dati alle marche. Addirittura, una marca potrebbe rappresentare un processo a sé stante.

Definizione 41 (**Regola di scatto**)**.** *Data la* ***marcatura*** $M : S \to \mathbb{N}$ *e una transizione* $t \in T$*:*

$$M[t > \ \textit{se e solo se}\ \forall s \in S\ M(s) \geq W(s,t)\ \wedge\ M(s)\ +\ W(t,s) \leq K(s) \tag{3.32}$$

$$M[t > M' \ \textit{se e solo se}\ M[t > \ \wedge \forall s \in S\ M'(s) = M(s) - W(s,t) + W(t,s) \tag{3.33}$$

Definizione 42. $\Sigma = (S, T, F, K, W, M_0)$ *un* ***sistema*** P/T*, l'insieme delle* ***marcature raggiungibili*** *di* Σ *è* $[M_0 >$*, il più piccolo insieme tale che:*

- $M_0 \in [M_0 >$
- $M \in [M_0 > \wedge \exists t \in T : M[t > M' \implies M' \in [M_0 >$

Definizione 43. $\Sigma = (S, T, F, K, W; M_0)$ *un* ***sistema*** P/T *allora* $RG(\Sigma) = ([M_0 >, U_\Sigma, A, M_0)$ *è il* ***grafo di raggiungibilità*** *di* Σ*, dove*

$$A = \{(M, U, M') : M, M' \in [M_0 > \wedge U \in U_\Sigma \wedge M[U > M'\} \tag{3.34}$$

U è un multi-insieme di transizioni indipendenti che sono abilitate in un passo.

Nota 7. *Se U è un* ***singoletto****, quindi composto da una sola transizione, allora il* ***grafo di raggiungibilità*** *diventa* ***grafo di raggiungibilità sequenziale*** $SRG(\Sigma)$*.*

Anche qui, come nei sistemi elementari si può costruire l'insieme delle marcature raggiungibili e il relativo grafo di raggiungibilità. Generalmente in queste reti la Diamond property non è più valida, questo causa self-loop.

Anche per le reti P/T si possono avere dei contatti, ovvero, situazioni in cui il peso dell'arco è w, la capacità del nodo di arrivo k e la sua marcatura è $m_s \neq 0$ allora ho un **contatto** se $w + m_s > k$. Quindi non può scattare la regola visto che dovrebbero passare w marche allo stato s ma quest'ultimo ha già m_s marche ed aggiungendone altre si supera il vincolo sulla capacità del nodo.

Come per i sistemi elementari, anche nelle reti a P/T è possibile aggiungere stati (quindi posti) per eliminare situazione di **contatto**. In sostanza si aggiungono per ogni stato la versione **complementata** e questo permette di eliminare il vincolo sulla capacità, senza alterare il comportamento del sistema.

Definizione 44. *Un sistema P/T $\Sigma = (S, T, F, K, W; M_0)$ è* ***senza contatti*** *sse* $\forall M \in [M_0 >, \forall t \in T, \forall s \in S$*:*

$$M(s) \geq W(s, t) \implies M(s) + W(t, s) \leq K(s) \tag{3.35}$$

Se $\Sigma = (S, T, F, K, W; M_0)$ è un **sistema** P/T senza contatti allora t è **abilitata** in $M \in [M_0 > (M[t >)$ sse

$$\forall s \in S, M(s) \geq W(s, t) \tag{3.36}$$

Nei sistemi senza contatti la **capacità** non ha alcun ruolo nella regola di scatto perché la transizione può scattare se nei suoi posti di input ci sono abbastanza marche.

Definizione 45 (Reti marcate). *Un sistema Posti e transizione è una* ***rete marcata*** *se non ha vincoli sulla capacità dei posti e ha tutti gli archi di peso unitario.*

$$\forall s \in S, M_0(s) \in \mathbb{N} \wedge K(s) = \infty \wedge \forall t \in T, W(s, t) \leq 1 \wedge W(t, s) \leq 1 \tag{3.37}$$

Le reti marcate sono denotate da $(S, T, F; M_0)$ in quanto K, W sono ridondanti

Definizione 46 (Reti marcate sicure). $\Sigma = (S, T, F; M_0)$ *una* ***rete marcata*** *allora è* ***sicura (safe)*** *sse:*

$$\forall M \in [M_0 >, \forall s \in S : M(s) \leq 1 \tag{3.38}$$

Una rete marcata è **safe** se comunque evolva il comportamento avremmo in ogni posto sempre al più una marca, i posti hanno quindi 0 o 1 marca e possiamo interpretarli di nuovo come condizioni booleane. L'unica differenza con le reti elementari è che nelle reti safe i cappi sono abilitati, mentre nelle reti elementari questo non succede. Se eliminiamo i cappi dalle reti marcate safe, otteniamo il corrispettivo di un sistema elementare puro.

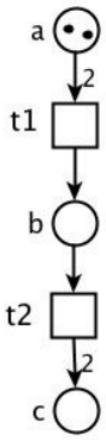
Definizione 47. *Sia $\Sigma = (S, T, F, K, W; M_0)$ è un* ***sistema*** *P/T tale che $\forall s \in S, K(s) = \infty$ e $N = (S, T, F)$ è* ***senza cappi****. Allora il sistema può essere rappresentato da un'unica matrice $\underline{N} : S \times T \to \mathbb{N}$ chiamata* ***matrice di incidenza****.*

Quindi avremo che

$$M_0[t_1 > \iff \underline{M_0} + \underline{t_1} \geq \underline{0} (\iff \underline{M_0} + \underline{N_{t_1}} \geq \underline{0}) \tag{3.39}$$

Quindi ogni cambio di stato lo possiamo codificare con una somma di vettori colonna, utilizzando l'**equazione di stato**:

$$M_0[\sigma > M_1 \implies \underline{M_0} + \underline{N} \cdot \underline{c_\sigma} = \underline{M_1} \tag{3.40}$$



Dove c_σ coincide con un vettore di $|T|$ elementi che specifica per ogni componente i-esima, quante volte compare la transizione t_i nella parola σ. Ad esempio:

$$\sigma = t_1 t_2 t_3 t_1 t_3 \implies c_\sigma = [2, 1, 2]$$

con:

- $c_\sigma[1] = 2$ perché ci sono 2 transizioni t_1
- $c_\sigma[2] = 1$ perché c'è 1 transizione t_2
- $c_\sigma[3] = 2$ perché ci sono 2 transizioni t_3

Esempio 5. *Supponiamo di avere la seguente rete P/T. Iniziamo a calcolare la matrice di incidenza calcolando ogni singola cella in questo modo:*

$$N[s,t] = \begin{cases} -W(s,t) & (s,t) \in F \\ W(s,t) & (t,s) \in F \end{cases}$$

Quindi in questo caso la tabella sarà

	t_1	t_2
a	-2	
b	1	-1
c		2

(3.41)

Successivamente calcolo le singole marcature in questo modo:

$$M_1 = M_0 + t_1 \quad M_2 = M_1 + t_2$$

	M_0	M_1	M_2
a	2		
b		1	
c			2

(3.42)

Si possono anche eseguire in sequenza utilizzando l'equazione di stato e quindi calcolare in un solo passo la marcatura di una sequenza di transizioni. La sequenza di transizioni è $\sigma = t_1 t_2$, quindi $c_\sigma = [1, 1]$ allora:

$$M_0[\sigma > M_2 \implies \underline{M_0} + \underline{N} \cdot \underline{c_\sigma} = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 & 0 \\ 1 & -1 \\ 0 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix} = \underline{M_2} \tag{3.43}$$

Osservazione 10. *Sia $\Sigma = (S, T, F, K, W, M_0)$ un **sistema** P/T tale che $\forall s \in S, K(s) = \infty$:*

- *Σ è **limitato** sse $\exists n \in \mathbb{N} : \forall s \in S, \forall M \in [M_0 >$:*

$$M(s) \leq n$$

- *Σ è **safe** sse*

$$\forall s \in S, \forall M \in [M_0 > : \ M(s) \leq 1$$

- Σ *è* ***limitato*** $\iff$ $[M_0>$ *è un insieme* ***finito*** *(*$RG(\Sigma)$ *è* ***finito****).*
- Σ *è* ***terminante*** *sse non ammette sequenze infinite*
- $M \in [M_0>$ *è una marcatura di* ***deadlock*** *sse* $\forall t \in T, \neg(M[t>)$
- Σ *è* ***deadlock-free*** *sse* $\forall M \in [M_0> \exists t \in T : M[t>$

  $$(\iff \nexists M \in [M_0>: M \text{ è una marcatura di deadlock})$$

- Σ *è* 1-***vivo*** *sse*

  $$\forall t \in T, \exists M \in [M_0>: M[t>$$

- Σ *è* ***vivo*** *sse*

  $$\forall t \in T, \forall M \in [M_0>, \exists M' \in [M>: M'[t>$$

  vivezza $\implies$ *assenza di* ***deadlock****.*
- Σ *è* ***reversiblile*** *(ciclico) sse*

  $$\forall M \in [M_0>: M_0 \in [M>$$

- Σ *è* ***reversibile*** $\iff$ *il suo grafo delle marcature raggiungibili è* ***strettamente connesso***
- *reversibilità* $+1$*-vivezza* $\implies$ *vivezza*

Per verificare le proprietà allora si può fare:

- analisi del grafo di raggiungibilità o di un prefisso dell'unfolding
- analisi strutturale del grafo della rete:
  - tecniche che sfruttano l'algebra lineare come le **equazioni di stato** che descrivono la dinamica e le S-invarianti e le T-invarianti
  - studio del grafo della rete come sottoinsiemi di nodi...
  - controllo delle condizioni necessarie e sufficienti per alcune sottoclassi di reti.

3.3.1 Reti ad alto livello

Dal momento che le reti P/T hanno un problema di perdita di informazioni, allora si può risolvere modellando le marche (contatori) con una struttura dati.

Capitolo 4

Dimostrazioni di correttezza

4.1 Logica proposizionale

Nella logica proposizionale, dal punto di vista teorico, il significato di una formula è rappresentato dal suo valore di verità.

Una volta definite sintassi e semantica, possiamo usare una logica per costruire delle dimostrazioni. Aver definito una semantica non vuol dire che di fronte ad una formula si è in grado di dire subito se sia valida o meno, dunque con la logica si sviluppa un metodo di dimostrazione e si cerca, in un numero finito di passi, di dimostrare la validità di una certa formula.

Un esempio, legato alla logica dell'aritmetica, è la formula che dice che per ogni numero primo, esiste sempre un numero primo più grande di esso; questa formula non è verificabile sperimentalmente essendo in numeri infiniti e va quindi dimostrata in altri modi.

Per ogni logica bisogna quindi definire un apparato deduttivo, cioè un insieme di **regole di inferenza**. Una regola di inferenza è una regola che dice "se hai già dimostrato queste premesse, allora puoi dedurre questa formula".

4.1.1 Sintassi

Passiamo ora al caso specifico della logica proposizionale che ci servirà per le dimostrazioni di correttezza.

Per costruire il linguaggio avremmo bisogno di:

- $PA = \{p_1, \dots, pi, \dots\}$ sono le proposizioni atomiche.
- $\bot,\ \top$ sono le costanti logiche, rappresentano formule o sempre false o sempre vere.
- $\neg,\ \wedge,\ \vee,\ \implies,\ \iff$ sono i connettivi logici, utilizzati per creare formule complesse.
- $(,\ )$ sono i delimitatori, utilizzati per la creazione di formule complesse.

Adesso dobbiamo definire la grammatica della logica proposizionale, per fare ciò si utilizza una definizione induttiva.

Definizione 48 (**Formule ben formate**). *Definiamo l'insieme delle formule ben formate FbF_p come:*

- $\bot,\ \top \in FbF_p$ *le costanti logiche sono delle formule ben formate.*
- $\forall p_i \in PA,\ p_i \in FbF_p$ *le proposizioni atomiche sono formule ben formate.*
- *Se* $A, B \in FbF_p$ *allora:*

$$(\neg A),\ (A \wedge B),\ (A \vee B),\ (A \implies B),\ (A \iff B)\ \in FbF_p \tag{4.1}$$

- *Nient'altro è una formula ben formata.*

Le formule atomiche rappresentano delle relazioni tra le variabili e le costanti oppure relazioni tra le variabili.

4.1.2 Semantica

Siamo ora interessati a conoscere il valore di una formula scritta attraverso la logica proposizionale. Il **valore di verità** di una formula dipende dai valori di verità delle sue proposizioni atomiche.

Il punto di partenza è quindi stabilire quali proposizioni atomiche sono da considerare vere. Questo si può fare formalmente definendo una **funzione di valutazione**:

$$v : PA \to \{0, 1\} \tag{4.2}$$

I connettivi della logica proposizionale hanno i seguenti valori di verità:

A	B	$A \wedge B$	$A \vee B$	$\neg A$	$A \implies B$	$A \iff B$
F	F	F	F	T	T	T
F	T	F	T	T	T	F
T	F	F	T	F	F	F
T	T	T	T	F	T	T

(4.3)

Questa funzione di valutazione può essere estesa induttivamente ad una funzione definita su tutte le formule ben formate, tale funzione prende il nome di **funzione di interpretazione** ed è definita come:

$$I_v : FbF_p \to \{0, 1\} \tag{4.4}$$

1. $I_v(\bot) = 0$ e $I_v(\top) = 1$ le costanti logiche hanno valore di verità definito.
2. $\forall p_i \in PA,\ I_v(p_i) = v(p_i)$ l'interpretazione di una preposizione atomica è data dal suo valore di verità.
3. Passo induttivo:

$$\begin{aligned} I_v(\neg A) &= 1 - I_v(A) \\ I_v(A \vee B) &= 1 \qquad \text{se e solo se } I_v(A) = 1 \text{ o } I_v(B) = 1 \\ &\dots \\ I_v(A \to B) &= 1 \qquad \text{se e solo se } I_v(A) = 0 \text{ o } I_v(B) = 1 \end{aligned} \tag{4.5}$$

Vediamo ora un po' di terminologia:

- A è **soddisfatta** da I_v se $I_v(A) = 1$.
- A è **soddisfacibile** se esiste I_v tale che $I_v(A) = 1$.
- A è una **tautologia** se $I_v(A) = 1$ per ogni I_v.
- A è una **contraddizione** se $I_v(A) = 0$ per ogni I_v.

Definizione 49 (**Logicamente equivalenti**)**.** *Due formule ben formate A e B sono logicamente equivalenti allora:*

$$A \equiv B \iff I_v(A) = I_v(B),\ \forall I_v \tag{4.6}$$

Si possono definire le seguenti **equivalenze logiche**:

- $A \wedge (A \vee B) \equiv A$
- $\neg(\neg A) \equiv A$
- $\neg(A \wedge B) \equiv \neg A \vee \neg B$
- $A \vee \neg A \equiv \top$
- $A \implies B \equiv \neg B \implies \neg A$
- $A \wedge \neg A \equiv \bot$

Definiamo ora i modelli, ovvero delle interpretazioni delle proposizioni atomiche, cioè una scelta di valori di verità per tutte le proposizioni atomiche.

Definizione 50 (**Modello**). *Un **modello** è un sottoinsieme delle proposizioni atomiche $M \subseteq PA$ a cui è associata un'interpretazione definita come $I_M : FbF_p \to \{0,1\}$ tale che:*

$$I_M(p_i) = 1 \text{ se e solo se } p_i \in M \tag{4.7}$$

Possiamo indicare la relazione tra modelli e formule come:

$$M \models A \tag{4.8}$$

la quale si può leggere come M modella A oppure come A è soddisfatta in M. Quindi scriveremo $M \models A$ se A risulta vera per la scelta particolare di M.

Definizione 51 (**Tautologia**). *Se $M \models A$ per tutti gli M, allora A è una tautologia e si indica $\models A$.*

Definizione 52 (**Soddisfacibilità**). *Se $M \models A$ per qualche M, allora A è soddisfacibile.*

Definizione 53 (**Insoddisfacibilità**). *Se $M \models A$ non è soddisfatta da nessun M, allora A è insoddisfabile.*

4.1.3 Apparato deduttivo

Possiamo ora rappresentare l'apparato deduttivo della logica proposizionale. Esso è composto da **regole di inferenza** scritte in questo modo:

$$\frac{A_1, \ldots, A_n}{B} \tag{4.9}$$

dove $A_i \in FbF_p$ sono le *premesse*, mentre $B \in FbF_p$ è la *conclusione*.

Alcune regole di inferenza sono ad esempio:

- Se ho dimostrato che A_1 e A_2 sono vere, sarà vera anche $A_1 \wedge A_2$:

$$\frac{A_1 \quad A_2}{A_1 \wedge A_2} \tag{4.10}$$

- Se ho dimostrato che $A_1 \wedge A_2$ è vera, sarà vera anche A_1:

$$\frac{A_1 \wedge A_2}{A_1} \tag{4.11}$$

- Modus Ponens:

$$\frac{A \quad A \implies B}{B} \tag{4.12}$$

- Modus Tollens:

$$\frac{A \implies B \quad \neg B}{\neg A} \tag{4.13}$$

Le regole di inferenza sono la base da cui è possibile costruire le dimostrazioni.

Definizione 54 (**Dimostrazione**). *Una **dimostrazione** è definita come una catena di regole $A_1, \ldots, A_n \vdash B$, ovvero da $A_1, \ldots, A_n$ si deriva B.*

Abbiamo ora due nozioni distinte:

- La validità in un modello $\models$.
- La derivabilità $\vdash$ introdotta perché in generale non siamo in grado di decidere direttamente la nozione semantica di validità e quindi cerchiamo di dimostrare una cosa.

Teorema 5 (**Validità, Correttezza**). *Se $A_1, \ldots, A_n \vdash B$ allora $A_1, \ldots, A_n \models B$. Ovvero, se riusciamo a derivare B da $A_1, \ldots, A_n$ in ogni modello in cui sono vere $A_1, \ldots, A_n$ allora è vera anche B.*

Questo teorema ci dice che abbiamo scelto bene le regole di inferenza e che queste non ci permettono di derivare cose non vere. Se questo teorema è valido, la logica è corretta.

Teorema 6 (**Completezza**). *Se $A_1, \ldots, A_n \models B$ allora $A_1, \ldots, A_n \vdash B$, ovvero se in ogni modello in cui sono vere $A_1, \ldots, A_n$ ed è soddisfatta anche B, si può derivare B da $A_1, \ldots, A_n$.*

Questo teorema ci dice che possiamo dimostrare tutto ciò che è vero. Se questo teorema è valido, la logica è completa.

4.2 Logica di Hoare

Questa logica si può vedere come costruita su due livelli diversi poiché permette di stabilire e definire il criterio di correttezza per un dato programma. In particolare, definisce questa correttezza tramite le definizioni di *precondizioni* e *post-condizioni* rappresentate da formule della logica proposizionale.

4.2.1 Primo livello

Al primo livello avremmo le proposizioni atomiche definite come relazioni fra le variabili del programma, tra queste relazioni possiamo trovare anche relazioni tra le variabili e le costanti.

Per dare una semantica, alle varie proposizioni atomiche definiamo la nozione di **stato della memoria**.

Definizione 55 (**Stato di memoria**). *Sia V l'insieme delle variabili del programma, definiamo uno* ***stato della memoria*** *come una fotografia della memoria del programma in un certo istante.*

Possiamo definire più formalmente quest'idea di stato della memoria come una funzione definita dall'insieme delle variabili ai numeri interi che assegna un valore ad ogni variabile.

$$\sigma : V \to \mathbb{Z} \tag{4.14}$$

Diremo quindi che la formula α è vera in σ scrivendo $\sigma \models \alpha$

4.2.2 Secondo livello

Le formule della logica di Hoare sono costruite tramite delle triple che prendono il nome di triple di Hoare, le quali sono definite come:

$$\{\alpha\}\ C\ \{\beta\} \tag{4.15}$$

dove:

- $\{\alpha\}$: è una formula che rappresenta le precondizioni.
- C: rappresenta un programma o un comando.
- $\{\beta\}$: è una formula che rappresenta le post-condizioni.

Le triple di Hoare si leggono come segue:

Se il comando C viene eseguito a partire da uno stato della memoria nel quale α è vera, allora l'esecuzione termina e nello stato finale β è vera.

4.2.3 Linguaggio

Definiamo ora il linguaggio che useremo con le triple di Hoare:

- Espressioni aritmetiche E definite come:
  - Se $z \in \mathbb{Z}$ allora z è un'espressione aritmetica $z \in E$.
  - $\forall e_1, e_2 \in E$ allora: $(e_1 + e_2), (e_1 - e_2), -e_1, (e_1 * e_2), (e_1/e_2), (e_1 \% e_2) \in E$
- Espressioni logiche B definite come:
  - Le costanti logiche $true$ e $false$ sono espressioni logiche $true,\ false \in B$.
  - $\forall b_1, b_2 \in B$ allora: $(b_1 \& b_2), (b_1 | b_2), !b_1 \in B$
  - $\forall e_1, e_2 \in E$ allora $(e_1 < e_2), (e_1 == e_2), (e_1 != e_2), (e_1 <= e_2) \in B$
- Comandi C definiti come:
  - skip $\in C$ questa istruzione non modifica la memoria.
  - Assegnamento $v := e\ \in C$ dove v è una variabile ed e è un espressione aritmetica.
  - Operatore di sequenza $C, D\ \in C$ dove C e D sono dei comandi.
  - Operatore di scelta if B then C else D endif $\in C$ dove B è un espressione logica, mentre C, D è sono dei comandi.

– Operatore di iterazione while B do C endwhile dove B è un espressione logica e C è un comando.

possiamo estendere il linguaggio introducendo altri comandi e strutture:

- do-while: do C while B endwhile $\equiv$ C; while B do C endwhile
- repeat: repeat C until B endrepeat $\equiv$ C; while $\neg B$ do C endwhile
- for: for $(D; B; F)$ C endfor $\equiv D;$ while B do $C; F$ endwhile
- procedure
- array

Una tripla di Hoare rappresenta un criterio di correttezza del programma, la sua regola di derivazione è definita come:

$$\frac{T_1, \ldots, T_m, f_1, \ldots, f_n}{T} \tag{4.16}$$

dove al numeratore sono specificate le premesse che possano contenere sia triple T_i che formule ben formate f_i, mentre al denominatore troviamo la conclusione che è una tripla T.

Vediamo ora come definire le regole di derivazione per i comandi introdotti:

1. **Skip**:

$$\frac{}{\{p\} \text{ skip } \{p\}} \tag{4.17}$$

   in questo caso la premessa è vuota e, visto che con questa operazione non modifico lo stato della memoria, le pre-condizioni e le post-condizioni sono le stesse.

2. **Sequenza**:

$$\frac{\{p\} \text{ C } \{p'\} \quad \{p'\} \text{ D } \{q\}}{\{p\} \text{ C; D } \{p\}} \tag{4.18}$$

   se le post-condizioni di C e le pre-condizioni di D sono le stesse posso mettere in sequenza i comandi ottenendo lo stesso risultato.

3. **Scelta**:

$$\frac{\{p \wedge B\} \text{ C } \{q\} \quad \{p \wedge \neg B\} \text{ D } \{q\}}{\{p\} \text{ if B then C else D endif } \{q\}} \tag{4.19}$$

   Se dimostriamo che:

   - Eseguendo C da uno stato dove vale la precondizione p e vale anche la condizione B al termine dell'esecuzione vale q.
   - Eseguendo D da uno stato dove vale p ma non vale B arriviamo comunque ad uno stato dove vale q.

   Possiamo derivare la regola della scelta dove prima dell'esecuzione dell'if vale p mentre dopo vale q.

4. **Implicazione** o **Conseguenza**:

$$\frac{p \rightarrow p' \quad \{p'\} \text{ C } \{q\}}{\{p\} \text{ C } \{q\}} \tag{4.20}$$

   inoltre:

$$\frac{\{p\} \text{ C } \{q\} \quad q \rightarrow q'}{\{p\} \text{ C } \{q'\}} \tag{4.21}$$

   Generalizzando possiamo dire che se abbiamo dimostrato una tripla e osserviamo una condizione che implica la precondizione della tripla, possiamo derivare la tripla con la condizione osservata al posto della precondizione. Discorso analogo è valido anche per la post-condizione.

5. **Assegnamento**:

$$\frac{}{\{q[E/q]\} \text{ x } := \text{ E } \{q\}} \tag{4.22}$$

   dove con $q[E/x]$ indico il fatto che sostituisco ogni occorrenza di x in q con E.

6. **Iterazione** (correttezza parziale):

$$\frac{\{inv \wedge B\}\ \text{C}\ \{inv\}}{\{inv\}\ \text{while}\ B\ \text{do C endwhile}\ \{inv \wedge \neg B\}} \tag{4.23}$$

l'idea per le istruzioni iterative è di trovare una formula che sia **invariante** rispetto al blocco dell'istruzione iterativa. Se troviamo un'invariante possiamo dire che se questa formula è vera all'inizio dell'istruzione iterativa, lo sarà anche alla fine e in più possiamo dire che al termine del blocco iterativo vale anche la negazione della condizione di ciclo.

In generale, avendo un'istruzione $W =$ while B do C endwhile , se deriviamo $\vdash \{p \wedge B\}C\{p\}$, allora possiamo dire che p è invariante rispetto a W.

Data un'istruzione iterativa, non c'è un solo invariante. Ad esempio, $True$ è invariante per ogni istruzione iterativa. Generalmente ci interessano gli invarianti utili alla dimostrazione in corso. Nella pratica, le istruzioni iterative sono inserite in programmi, di conseguenza la scelta dell'invariante dipende dal contesto e si abbina alla regola dell'implicazione.

A volte l'invariante non è assoluto, ma lo diventa aggiungendoci la condizione di ciclo. L'invariante in genere è violato guardando lo stato della memoria durante l'esecuzione del blocco, ma viene ripristinato al termine del blocco.

Definizione 56 (**Invariante di un ciclo**)**.** *Possiamo definire l'**invariante di un ciclo** come una formula che se risulta vera all'inizio di un iterazione è vera anche alla fine di essa.*

La logica di Hoare è una logica **corretta**, ovvero vale:

$$\vdash \implies \models \tag{4.24}$$

questo significa che tutte le triple che riusciamo a derivare sono sicuramente delle triple valide. Inoltre, è anche **completa** (relativamente):

$$\models \implies \vdash \tag{4.25}$$

tutto ciò che è valido è derivabile. Tuttavia è una completezza relativa a causa dell'incompletezza dell'aritmetica: potrebbe succedere di dover dimostrare una proprietà aritmetica che però è indimostrabile (caso molto remoto).

Come notazione usiamo che:

$$\vdash \{p\}\ \text{C}\ \{q\} \tag{4.26}$$

dove $\vdash$ segnala che la tripla è stata dimostrata con le regole di derivazione (si parla quindi di sintassi, viene infatti ignorato il significato ma si cerca solo di applicare le regole, ottenendo la conclusione come risultato di una catena di regole).

Usiamo anche:

$$\models \{p\}\ \text{C}\ \{q\} \tag{4.27}$$

dove $\models$ indica che la tripla è vera (si parla quindi di semantica, riferendosi al significato).

Dato che si ha completezza e correttezza dell'apparato deduttivo si hanno due situazioni.

- Ogni tripla derivabile è anche vera in qualsiasi interpretazione.
- Ogni tripla vera vorremmo fosse anche derivabile e il discorso verrà approfondito in seguito per la logica di Hoare

Nota 8. *Negli esercizi si assegna a $\vdash$ una sigla, posizionata come pedice o apice, per rappresentare la regola che viene applicata.*

4.2.4 Correttezza totale

Vogliamo ora analizzare la correttezza totale dell'istruzione while, ovvero si vuole verificare anche la terminazione del ciclo:

$$\{p\}\ \text{while B do C endwhile}\ \{q\} \tag{4.28}$$

Definizione 57 (**Correttezza parziale**)**.** *Definiamo la **correttezza parziale** come: "Se si esegue W a partire da uno stato in cui vale p e l'esecuzione termina, nello stato finale vale q"*

$$\overset{parz}{\vdash}\ \{p\}C_1\{q\} \tag{4.29}$$

Definizione 58 (**Correttezza totale**). *Definiamo la **correttezza totale** come: "Se si esegue W a partire da uno stato in cui vale p, l'esecuzione termina e nello stato finale vale q":*

$$\overset{tot}{\vdash} \{p\}C_1\{q\} \tag{4.30}$$

Vediamo ora una tecnica per dimostrare la correttezza totale. Supponiamo che E sia un'espressione aritmetica nella quale compaiono variabili del programma, costanti numeriche e operazioni aritmetiche, e che inv sia un invariante di ciclo per W, scelti in modo che:

1. $inv \implies E \geq 0$
2. $\overset{tot}{\vdash} \{inv \wedge B \wedge E = k > 0\}\ C\ \{inv \wedge E < k\}$

Allora:

$$\overset{tot}{\vdash} \{inv\}\ W\ \{inv \wedge \neg B\} \tag{4.31}$$

Nota 9. *E non è una formula logica, ma $E \geq 0$ è una formula logica. Lo 0 in $E \geq 0$ può essere sostituito da qualsiasi numero.*

4.2.5 Schema generale per la dimostrazione

Consideriamo una generica istruzione composta da una precondizione p e una post-condizione q e chiamiamo le istruzioni in sequenza V, W e Z. Inoltre, supponiamo che V e Z non contengano istruzioni di iterazioni.

$$\{p\}\ V;\ W;\ Z\ \{q\} \tag{4.32}$$

Il processo di dimostrazione inizia analizzando $Z\ \{q\}$ dal quale si ricava $wp(Z, q) \equiv s\ (\{s\}\ Z\ \{q\})$. A questo punto cerchiamo un invariante i per W tale che $(i \wedge \neg B) \implies s\ (\{i\}\ W;\ Z\ \{q\})$. Infine, cerchiamo una formula u tale che $\{p\} \vee \{u\}$ è $u \implies i\ (\{p\}\ V;\ W;\ Z\ \{q\})$.

4.2.6 Proprietà

La logica di Hoare gode delle proprietà di:

- **Correttezza**: $\vdash \implies \models$
- **Completezza** (relativa): $\models \implies \vdash$

Vogliamo ora risolvere il problema relativo al trovare una formula p, che dati un comando C e una formula q mi permette di ottenere:

$$\vdash \{p\}C\{q\} \tag{4.33}$$

Per fare ciò definiamo:

- V insieme delle variabili di C
- $\Sigma = \{\sigma | \sigma : V \to \mathbb{Z}\}$ insieme degli stati di memoria.
- Π insieme delle formule su V.
- $\models \subseteq \Sigma \times \Pi$ è definita come p è vera in σ:

  $$\sigma \models p \tag{4.34}$$

  partendo da questa possiamo definire due funzioni:

  - $t(\sigma) = \{p \in \Pi | \sigma \models p\}$ ovvero l'insieme delle formule vere in σ.
  - $m(p) = \{\sigma \in \Sigma | \sigma \models p\}$ ovvero l'insieme degli stati che soddisfano p.

  Possiamo generalizzare queste formule per insiemi, siano $S \subseteq \Sigma$ sottoinsieme di stati e $F \subseteq \Pi$ sotto-insieme di formule:

  - $t(S) = \{p \in \Pi | \forall s \in S : s \models p\} = \bigcap_{s \in S} t(s)$.
  - $m(F) = \{s \in \Sigma | \forall p \in F : s \models p\} = \bigcap_{p \in F} m(p)$.

A questo punto possiamo esprimere le formule della logica proposizionale attraverso operazioni tra insiemi:

- $m(\neg q) = \Sigma \backslash m(q)$
- $m(p \vee q) = m(p) \cup m(q)$
- $m(p \wedge q) = m(p) \cap m(q)$
- $m(p \implies q) = m(\neg p) \cup m(q)$, oltre a questo l'implicazione ha anche una relazione tra formule: "se p implica q, allora $m(p) \subseteq m(q)$ ovvero q è più debole di p.

Nota 10. *L'implicazione può assumere due significati:*

- ***Connettivo logico**:*
$$p \implies q \equiv \neg p \vee q \tag{4.35}$$
- ***Relazione tra formule**:*
$$p \implies q \text{ allora } m(p) \subseteq m(q) \tag{4.36}$$
q è più debole (mi da un'informazione minore) di p

Definite queste operazioni possiamo definire i criteri di scelta della precondizione *migliore* $C\{q\}$ cerchiamo la pre-condizione più debole (**weakest precondition**) p tale che:

$$\models \{p\}\ C\ \{q\} \tag{4.37}$$

p corrisponde al più grande insieme di stati a partire dai quali l'esecuzione di C porta a uno stato in $m(q)$.

Abbiamo definito che esiste sempre tale condizione, vediamo ora come calcolarla. Usiamo la notazione $wp(C, q)$ per definire la precondizione più debole per $C\{q\}$.

Teorema 7. $\models \{p\}C\{q\}$ *se e solo se* $p \implies wp(C, q)$

Le regole di calcolo di wp sono:

- **Assegnamento**: $wp(x := E, q) = q[E/x]$ sostituisco tutte le occorrenze di x con E.
- **Sequenza**: $wp(C_1; C_2, q) = wp(C_1, wp(C_2, q))$
- **Scelta**: $wp(C, q) = (B \wedge wp(C_1, q)) \vee (\neg B \wedge wp(C_2, q))$ dove C è definita come: C : if B then C_1 else C_2 endif

Capitolo 5

Model checking

5.1 Introduzione

Le **logiche temporali** sono una famiglia di logiche matematiche che permette di esprimere proprietà che cambiano nel tempo.

Quando si analizzano sistemi reattivi (quindi concorrenti) allora non vale più il ragionamento di avere uno stato precedente e uno stato successivo che varia nel tempo.

Sono stati introdotti diversi modelli per modellare i sistemi concorrenti, ora si introdurranno nuove tecniche di analisi.

Per esempio nel caso del produttore e consumatore, i soggetti sono indipendenti quindi si devono sincronizzare per l'accesso al buffer. Per questi programmi sarà importante garantire la correttezza del programma:

- Ogni oggetto deve essere prima prodotto.
- Ogni oggetto non può essere consumato più di una volta.
- Il sistema non raggiunge mai uno stato di **deadlock**.

Le reti di Petri ci permettono di controllare se si mantiene la mutua esclusione, ovvero che non esista una marcatura in cui entrambi i processi non possono essere negli stati della sezione critica. Noi vorremmo modellare il blocco di uno dei due processi se uno si trova nella zona critica.

Vogliamo quindi un modello per verificare se sono valide delle proprietà di interesse e se il modello rappresenta effettivamente il sistema.

Definizione 59 (**Sistema reattivo**). *Un sistema è **reattivo** se è un sistema concorrente, distribuito e asincrono.*

Una sottoclasse dei sistemi reattivi sono quelli sincroni con un clock, ma è una semplificazione.

I sistemi reattivi non obbediscono più al paradigma input-computazione-output, rendendo quindi impossibile utilizzare le triple di Hoare per dimostrare la correttezza di un programma. Possiamo sempre riutilizzare la logica di hoare per dimostrare alcuni pezzi, ma dovremmo usare un modo per unire i risultati.

Al posto delle triple di Hoare, utilizzeremo delle **asserzioni**, ovvero delle frasi al cui interno sono presenti elementi temporali che descrivono il comportamento del sistema.

Esempio 6. *Se è stato spedito un messaggio allora questo prima o poi verrà ricevuto dal destinatario.*

Esempio 7. *Se si accende una spia di allarme allora questa sarà accesa fino a quando non si spegne il dispositivo*

Vediamo ora come possiamo procedere nell'analisi dei sistemi concorrenti. Il problema che vogliamo risolvere è quello di stabilire se un sistema reattivo è corretto. Per fare ciò, seguiremo il seguente schema:

1. Si esprime il criterio di correttezza come una formula di un opportuno linguaggio logico.
2. Si modella il sistema nella forma di un sistema di transizioni.
3. Si valuta se la formula è vera nel sistema di transizioni. (algoritmi)

Definizione 60 (**Sistema di transizioni**). *Un **sistema di transizioni** è definito da:*

$$A = (Q, T) \tag{5.1}$$

dove:

- *Q insieme degli stati (non per forza finito).*
- *$T \subseteq Q \times Q$ insiemi di transizioni di stato.*

Definizione 61 (**Cammino**). *Definiremo un **cammino** come una sequenza di stati π dove:*

$$\pi = q_0 q_1 \dots q_n \ (q_i, q_{i+1}) \in T \ \forall i \tag{5.2}$$

(combacia con la definizione di cammino su grafi).

Definizione 62 (**Suffisso**). *Definiremo un **suffisso** di ordine i per un cammino π è il cammino che inizia da uno stato di indice i.*

$$\pi^{(i)} = q_i q_{i+1} \dots q_n \tag{5.3}$$

Definizione 63 (**Cammino massimale**). *Definiremo un **cammino massimale** se il cammino è finito e non può essere esteso, cioè solo quando il cammino raggiunge uno stato che non ha transizioni uscenti. (cammino massimale finito)*

Il cammino massimale è infinito quando non si ha uno stato pozzo.

Nota 11. *Ogni suffisso di un cammino massimale è a sua volta un cammino massimale.*

Posso rappresentare un cammino espandendo la definizione delle espressioni regolari, aggiungendo $^\omega$ per rappresentare una sequenza infinita. Bisogna prestare particolare attenzione al fatto che $\omega \neq *$, infatti $*$ rappresenta una sequenza arbitraria di valori.

Nota 12. *Posso esprimere un cammino come un suffisso di ordine 0.*

5.2 Logica Temporale Lineare

Definiamo la **sintassi** partendo dalle proposizioni atomiche:

$$AP = \{p_1, p_2, \dots, p_i, \dots\} \tag{5.4}$$

composte dalle asserzioni considerate vere a prescindere.

Esempio 8. *Il messaggio è stato spedito, la spia di allarme è accesa*

Definiamo anche le formule ben formate FBF_{LTL}:

- Ogni proposizione atomica è una formula ben formata.
- $\top$ e $\bot$ sono formule ben formate.
- Se $\alpha, \beta \in FBF_{LTL}$ allora $\alpha \vee \beta, \neg\alpha \in FBF_{LTL}$ (da queste si derivano tutti i connettivi logici).
- **Operatori temporali**: siano α e β formule ben formate, allora anche:
  - $X\alpha$ "next α" nel prossimo stato α è vera.
  - $F\alpha$ "future α" prima o poi α sarà vera.
  - $G\alpha$ "globally α" α è sempre vera.
  - $\alpha U \beta$ "α until β" α è vera fino a quando β diventa vera.

  sono formule ben formate.

Per definire la sintassi utilizzeremo i **Modelli di Kripke** che prendono i sistemi di transizione e li arricchiscono associando a ogni stato $q \in Q$ l'insieme delle proposizioni atomiche che sono vere in q. Quindi possiamo definire:

$$I \ : \ Q \to 2^{AP} \tag{5.5}$$

dove 2^{AP} è l'insieme delle parti di AP. Un modello di Kripke sarà definito come:

$$A = (Q, T, I) \tag{5.6}$$

Con un modello di Kripke possiamo identificare per uno stato quali sono le proposizioni atomiche vere e per capire se le FBF_{LTL} sono vere allora dovrò analizzare i cammini che partono dallo stato in esame.

Vediamo ora come associare una **semantica** alle formule ben formate. Per fare ciò procediamo in due fasi:

1. Definiamo un **criterio** per stabilire se una formula α è vera in un cammino massimale π.
2. Diciamo che la formula è **vera** rispetto a uno stato q se è vera in tutti i **cammini massimali** che partono da q.

Fissiamo quindi un cammino generico π e α una formula ben formata allora:

$$\pi \vDash \alpha \iff \text{ significa che } \alpha \text{ è vera nel cammino } \pi \tag{5.7}$$

Definiremo la relazione $\vDash$ per induzione.

Definizione 64 ($\vDash$)**.** *Supponiamo che α e β siano due formule ben formate e p una preposizione atomica:*

- ***caso base**:*
  - $\pi \vDash \top$
  - $\pi \nvDash \bot$
  - $\pi \vDash p \iff p \in I(q_0)$
- ***passo induttivo**:*
  - ***Operatori logici**: supponiamo $\alpha, \beta \in FBF_{LTL}$*
    - $\pi \vDash \neg\alpha \iff \pi \nvDash \alpha$
    - $\pi \vDash \alpha \vee \beta \iff (\pi \vDash \alpha) \vee (\pi \vDash \beta)$
  - ***Operatori temporali**: supponiamo $\beta, \gamma \in FBF_{LTL}$*
    - $\pi \vDash X\beta \iff \pi^{(1)} \vDash \beta$
    - $\pi \vDash F\beta \iff \exists i \in \mathbb{N} : \pi^{(i)} \vDash \beta$
    - $\pi \vDash G\beta \iff \forall i \in \mathbb{N} : \pi^{(i)} \vDash \beta$
    - $\pi \vDash \beta U \gamma \iff$ *:*
      - $\exists i \in \mathbb{N}$ *tale che:* $\pi^{(i)} \vDash \gamma$ *cioè* $\pi \vDash F\gamma$*.*
      - $\forall h, 0 \leq h < i, \pi^{(h)} \vDash \beta$

      Se γ è vera fin da subito, quindi $i = 0$ allora β è superfluo.

Esempio 9. *Ecco alcuni esempi:*

- *$FG\alpha$: esprime che da un certo momento in poi α sarà invariante, se $\alpha = \bot$ allo stato iniziale ma $FG\alpha = \top$ vale dallo stato iniziale allora significa che non si tornerà più in quello stato.*
- *$GF\alpha$: α è vera in un numero infinito di stati.*
- *$G\neg(cs_1 \wedge cs_2)$: mutua esclusione se cs_1 indica che il processo è nella sezione critica, viceversa cs_2.*
- *$G(req \implies XFack)$: se si manda una richiesta, dallo stato successivo prima o poi si riceverà un ack.*
- *$G(req \implies (reqUack))$: se c'è la richiesta allora la richiesta è pendente fino a quando si ha un'ack. Si differenzia da quella precedente dal momento che la richiesta deve rimanere fino a quando non si manda l'ack.*
- *$G(req \implies ((req \wedge \neg ack)U(\neg req \wedge ack)))$: si chiede di non rispondere all'inizio e dopo la risposta la richiesta viene cancellata.*

Esempio 10. *Consideriamo la seguente rete di Petri (5.1), la quale rappresenta due processi che vogliono accedere ad una risorsa condivisa. Questa rete è composta da:*

- *rd la risorsa disponibile.*
- *sc1 e sc2 le due sezioni critiche.*
- *snc1 e snc2 le due sezioni non critiche.*
- *req1 e req2 le due richieste di usare la risorsa.*

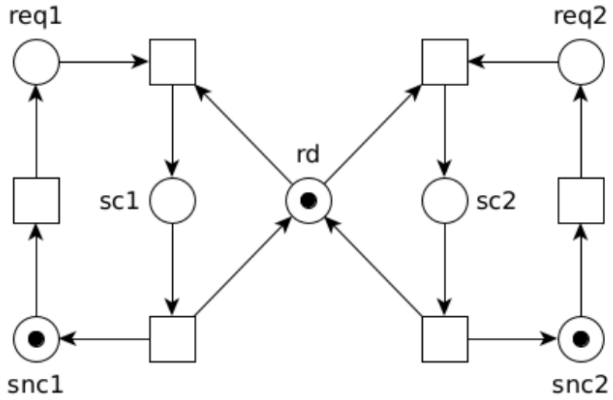


Figura 5.1: Rete di Petri per il model checking

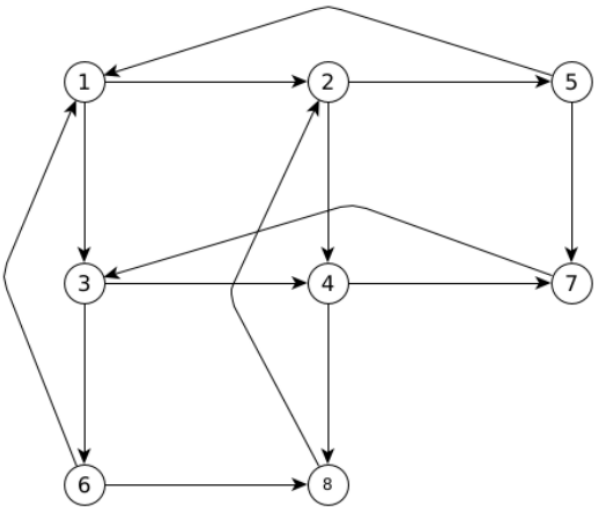


Figura 5.2: Grafo dei casi della rete di Petri

Per poter applicare le tecniche di model checking dobbiamo trasformare questa rete in un modello di Kripke. Per farlo dobbiamo definire gli stati e le transizioni. Una possibile soluzione è quella di calcolare il grafo dei casi (o grafo di raggiungibilità) della rete di Petri ed assegnare ad ogni nodo un insieme di proposizioni atomiche che sono vere in quel nodo.

Possiamo quindi definire il grafo dei casi dove per semplicità utilizziamo il nodo 1 *come stato iniziale: dove ogni nodo è rappresentato da:*

- $1 = \{rd, snc1, snc2\}$
- $2 = \{rd, req1, snc2\}$
- $3 = \{rd, snc1, req2\}$
- $4 = \{rd, req1, req2\}$
- $5 = \{sc1, snc2\}$
- $6 = \{snc1, sc2\}$
- $7 = \{sc1, req2\}$
- $8 = \{req1, sc2\}$

Una volta costruita la struttura, definiamo delle formule che vogliamo siano vere per la situazione in analisi. Studiamo quindi:

- $G\neg(sc_1 \wedge sc_2)$*: esprime che non si può raggiungere uno stato in cui entrambi i processi sono nella sezione critica.*
- $G(req1 \implies Fsc1)$*: esprime che se il processo 1 fa richiesta allora prima o poi entrerà nella sezione critica.*

Per verificare le formule dovremmo considerare tutti i cammini massimali che partono dallo stato 1 e verificare che le formule siano valide in tutti i cammini. Tuttavia, questo caso è complesso e i possibili cammini massimali sono potenzialmente infiniti.

Per la prima formula, la via più facile è, osservando che dallo stato si possono raggiungere tutti gli altri stati, scorrere la tabella delle formule vere in ciascun stato e osservare che in nessun stato è vera la formule $sc_1 \wedge sc_2$. Quindi non sarà vera globalmente.

Per la seconda formula non possiamo più applicare questa strategia, ma possiamo cercare di confutare la formula: basta trovare un cammino massimale che non soddisfi la formula.

In ogni caso questa seconda formula non è valida in quando potrei avere il cammino $\pi^{(1)}(2,4,8)^\omega$ in cui anche il secondo processo fa richiesta e ci entra. Posso entrare in un loop in cui solamente il processo due prende ogni volta la risorsa quindi la formula non è valida in quel cammino massimale.

Possiamo comunque trovare cammini in cui vale, come $(1,3,6)^\omega$, in quanto in quei tre stati $req1$ è sempre falsa.

*Una possibile soluzione è quella di aggiungere un nuovo stato 9, duplicando lo stato 4 in modo da implementare un meccanismo di **turno** in cui i processi si alternano nell'uso della risorsa.*

Come per tutte le logiche, anche per la logica temporale si può definire una equivalenza tra formule.

Definizione 65 (**Equivalenza tra formule**). *Possiamo definire l'**equivalenza tra formule** come:*

$$\alpha \equiv \beta \iff \forall \pi : (\pi \vDash \alpha \iff \pi \vDash \beta) \tag{5.8}$$

Esempio 11. *Vediamo ora alcuni esempi di equivalenza tra formule:*

- $F\alpha \equiv \alpha \vee XF\alpha$ *se prima o poi α sarà vera, allora o è vera in questo stato oppure dal prossimo stato α sarà prima o poi vera.*
- $G\alpha \equiv \alpha \wedge XG\alpha$ *Se α è sempre vera, allora è vera in questo stato e dal prossimo stato sarà sempre vera.*
- $\alpha U\beta \equiv \beta \vee (\alpha \wedge X(\alpha U\beta))$ *se β è vera non ci interessa altro, oppure α è vera e dal prossimo stato α sarà vera fino a quando β diventa vera.*
- $FGF\alpha \equiv GF\alpha$ *se prima o poi α sarà vera, allora da un certo momento in poi α sarà sempre vera.*
- $GFG\alpha \equiv FG\alpha$ *se α è sempre vera, allora da un certo momento in poi α sarà sempre vera.*
- $\top U\alpha \equiv F\alpha$*: questo significa che l'operatore F non è essenziale.*
- $\neg F\neg\alpha \equiv G\alpha$*: questo significa che G può essere definito a partire da F quindi l'insieme minimale di operazioni sono $\{X,U\}$. Questo insieme minimale non è unico.*

Le equivalenze spesso sfruttano la ricorsione dal momento che si basano su sistemi di transizione.

Le equivalenze logiche permettono di definire degli **operatori derivati**, per velocizzare la scrittura delle formule. Alcuni esempi di operatori derivati sono:

- **Until debole**: $\alpha W\beta \equiv G\alpha \vee (\alpha U\beta)$ con questo operatore si può esprimere la proprietà in cui α è sempre vera oppure α è vera fino a quando β diventa vera. W differisce da U perché non si richiede che la seconda formula diventi vera per rendere vera W.
- **Release**: $\alpha R\beta$ per cui

  $$\pi \vDash \alpha R\beta \iff \forall k \geq 0 : (\pi^{(k)} \vDash \beta \vee \exists h < k : \pi^{(h)} \vDash \alpha) \tag{5.9}$$

  In sostanza o β è sempre vera oppure β potrà essere falsa solo quando α diventa vera. Definito in questo modo allora si può dire:

  $$\alpha R\beta \equiv \beta W(\alpha \wedge \beta) \tag{5.10}$$

Definizione 66 (**Insieme di operatori minimale**). *L'insieme degli operatori è detto **minimale** se ogni altro operatore può essere definito a partire da questi. Un esempio di insieme di operatori minimale per le logiche temporali è $\{X,U\}$.*

Attenzione all'uso della negazione nelle formule LTL, cosa significa "non è vero $F\alpha$"? Significa che $\exists$ un cammino per cui non vale $F\alpha$. Mentre $\neg F\alpha \equiv G\neg\alpha$ quindi $\neg F\alpha$ non è la negazione di $F\alpha$. Quindi dobbiamo stare attenti ai quantificatori universali nascosti.

Negli esercizi sulla logica temporale ci possono essere diverse risposte giuste. Ecco degli esercizi sulla logica temporale lineare.

Esempio 12. *Traduci questa frase:*

Chi ruba, presto o tardi finirà in galera.

possiamo considerare 2 proposizioni atomiche:

- *hr: ho rubato*
- *c: sono in carcere*

Si può riformulare la frase come una legge, quindi come una cosa che vale sempre, portandoci ad utilizzare l'operatore G. In aggiunta, possiamo riformularla in un modo che ha una semantica più vicina alla semantica degli operatori logici.

Se rubi, in futuro finirai in galera.

Si può tradurre in

$$G(hr \implies XF\ c) \tag{5.11}$$

Utilizziamo l'operatore X per evitare che nel momento in cui rubo allora subito finisco in galera.

Possiamo provare a tradurre la frase

Solo chi ruba finirà in galera.

Quindi si può tradurre

$$\neg c\ W\ hr \equiv G(c \iff hr) \tag{5.12}$$

Esempio 13. *Traduci questa frase:*

Chi ruba finirà in carcere, ma solo dopo avere parlato con un avvocato.

si aggiunge anche la proposizione atomica: "ho parlato con un avvocato"

$$G(hr \implies (XF\ c \wedge (\neg c\ U\ pa))) \tag{5.13}$$

Si potrebbe pensare che XF c si possa escludere ma in realtà no perché U porta a dire che dopo aver parlato con un avvocato non vuol dire che c diventi vera.

Esempio 14. *Traduci questa frase:*

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, allora la cabina non cambierà direzione fino a quando avrà raggiunto il quinto piano.

si specificano le proposizioni atomiche:

- *su: "la cabina sta salendo"*
- *p_i: "cabina all'altezza del piano i"*
- *r_i: "pulsante interno del piano i è stato premuto"*

$$G((su \wedge p_2 \wedge r_5) \implies (su\ U\ p_5)) \tag{5.14}$$

La logica temporale che abbiamo studiato fino a questo momento (LTL) permette di esprimere diverse proprietà dei sistemi reattivi, ma ha dei limiti, infatti non permette di esprimere proprietà del tipo:

"esiste un cammino in cui vale α"

Attraverso le formule della logica LTL possiamo esprimere solo proprietà in cui si richiede che una certa formula sia vera *in tutti* i cammini massimali. Non possono essere espresse proprietà relative all'esistenza di un cammino in cui una certa formula è vera.

Il problema è che queste proprietà sono utili, per risolvere questa mancanza introduciamo una nuova logica. Questa logica può essere interpretata attraverso il concetto di **albero di computazione**.

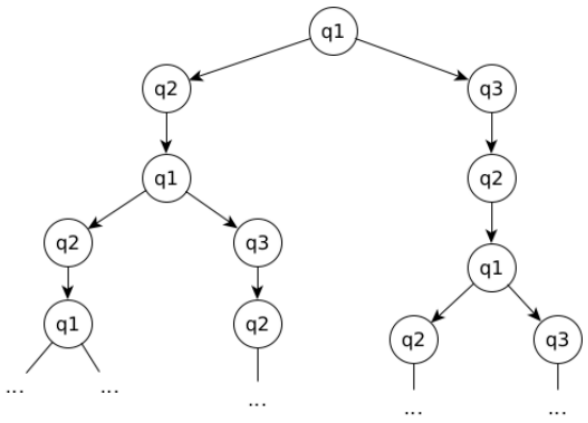


Figura 5.3: Esempio di computational tree

Definizione 67. *Un **albero di computazione** è una struttura che mi permette di rappresentare le possibili computazioni di un sistema di transizioni etichettato.*

*I cammini che partono dalla radice di questo albero sono definiti come **computazioni** del sistema. 5.3*

Nota 13. *Spesso si vuole esprime le possibili computazioni di un sistema allora possiamo costruire un albero (simile all'**unfolding** per le reti di petri) che rappresenta la computazione rispetto gli stati globali.*

L'albero può essere costruito in modo induttivo:

- *Si sceglie lo stato iniziale.*
- *Se si hanno transizioni verso altri stati allora li aggiungo come prossimi nodi.*
- *Così via ricorsivamente, se ci sono archi all'indietro allora si ricrea un altro nodo in avanti.*

Nel caso fosse ciclico allora sarà infinito, se abbiamo stati in deadlock allora il nodo sarà pozzo.

5.3 Computation Tree Logic - CTL

Possiamo definire ora la logica CTL (**Computation Tree Logic**), ovvero una logica temporale che si basa sugli alberi di computazione.

Definiremo la **sintassi** attraverso le FbF_{CTL}.

Definizione 68 (formule ben formate)**.** *Le formule ben formate sono:*

- $\forall p \in AP, p \in FbF_{CTL}$ *con* $AP = \{p1, p2, \dots\}$ *insieme delle preposizioni atomiche.*
- $\forall \alpha, \beta \in FbF_{CTL}$
  - $\neg\alpha, \alpha \vee \beta \in FbF_{CTL}$
  - $AX\ \alpha, EX\ \alpha \in FbF_{CTL}$ *per ogni / esiste un cammino dallo stato corrente tale che nello stato prossimo vale* α
  - $AF\ \alpha, EF\ \alpha \in FbF_{CTL}$ *per ogni / esiste un cammino dallo stato corrente tale che prima o poi vale* α
  - $AG\ \alpha, EG\ \alpha \in FbF_{CTL}$ *per ogni / esiste un cammino dallo stato corrente tale che vale sempre* α
  - $A(\alpha\ U\ \beta), E(\alpha\ U\ \beta) \in FbF_{CTL}$ *per ogni / esiste un cammino dallo stato corrente tale che* α *è vera fino a quando* β *non diventa vera. (*β *deve per forza diventare vera)*

In particolare abbiamo aggiunto A e E i quali rappresentano due quantificatori sui cammini. Essi sono definiti come segue:

- $A \equiv \forall$ è il quantificatore universale e si interpreta come per ogni cammino.
- $E \equiv \exists$ è il quantificatore esistenziale e si interpreta come esiste un cammino tale che.

Esempio 15. *Traduciamo*

Dopo l'accensione della spia, sarà sempre possibile riportare il sistema allo stato iniziale.

Definiamo:

- *s: la spia è accesa*
- *init: il sistema si trova nello stato iniziale*

$$AG(s \implies AX\ EF\ init) \tag{5.15}$$

Esempio 16 (Esercizio d'esame)**.** *Dato il sistema di transizione riportato in figura 5.4, vogliamo verificare la seguente proprietà:*

- $AF\ q$
- $AG(EF(p \vee q))$, $GF(p \vee q)$
- $EX(EX\ r)$, $XX\ r$
- $AG(AF\ q)$

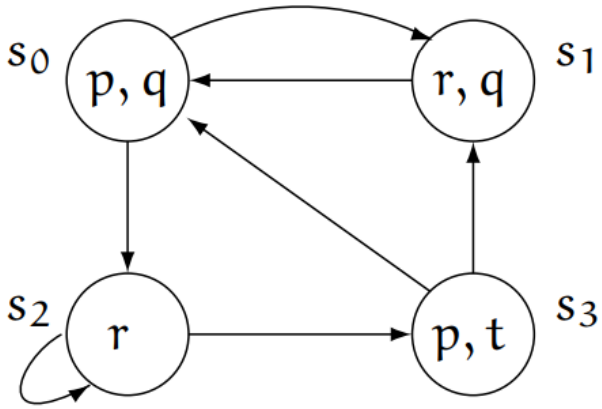


Figura 5.4: Sistema di transizione

Vediamo ora come risolvere i vari punti:

- *$AF\ q$: in questo caso conviene suddividere la formula ed analizzare gli elementi semplici, come F e q. A questo punto controlliamo per ogni stato dove vale q, in questi stati, per come abbiamo definito F la formula risulta vera. Quindi negli stati s_0 e s_1 la formula è vera. Successivamente verifico se sui cammini che partono dagli altri stati q diventa vera. Questo è vero per il cammino che parte da S_3, ma non per quello che parte da s_2, dal momento che in quest'ultimo stato è presente un cappio quindi può non andare mai in s_3.*
- *$GF(p \vee q)$: avendo questa formula non basta analizzare dove valgono p o q perché abbiamo G quindi dobbiamo controllare anche che per tutti gli stati successivi p o q siano vere. Partendo dallo stato s_0 la formula è falsa in quanto si arriva in s_2 dove nessuno delle due sono vere. s_2, s_3 vero ma anche s_1 è falsa.*
- *$AG(EF(p \vee q))$: è vera per tutti gli stati perché esiste sempre un cammino con una deviazione in cui vale $p \vee q$.*
- *$XX\ r$: non è vera in s_0 perché abbiamo trovato un cammino per cui non è vera $s_0 \to s_1 \to s_0$. Mentre s_1 è verificata.*
- *$EX(EX\ r)$: è vera in s_0 perché abbiamo trovato un cammino per cui è vera $s_0 \to s_2 \to s_2$ e a noi ne basta una sola.*
- *$AG(AF\ q)$: non vale in s_2 perché possiamo rimanere all'infinito, Mentre negli altri è sempre vera.*

Nota 14. *Ricorda che per la semantica della logica LTL allora alcune delle FbF_{LTL} sono equivalenti alle FbF_{CTL} che hanno l'operatore A e viceversa.*

Per esempio

$$AF\alpha \equiv F\alpha \tag{5.16}$$

Anche per questa logica definiremo l'equivalenza logica.

Definizione 69 (**Formule equivalenti**)**.** *Due formule α e β sono equivalenti, indicato come $\alpha \equiv \beta$, se e solo se:*

$$\forall\pi \wedge \forall \models: (\pi \models \alpha \iff \pi \models \beta) \tag{5.17}$$

In entrambe le logiche, LTL e CTL, possiamo esprimere le seguenti proprietà:

- **Invariante**: si possono avere delle formule sempre vere:

$$AG\neg p \equiv G\neg p \tag{5.18}$$

- **Reattiva**: il valore logico di una formula implica la veridicità di un'altra:

$$AG(p \implies AF\ q) \equiv G(p \implies F\ q) \tag{5.19}$$

Le due logiche non hanno la stessa potenza espressiva ma nessuna delle due è più espressiva rispetto all'altra. Infatti per esempio la logica CTL può esprimere la seguente proprietà (**reset property**):

Da ogni stato raggiungibile in ogni cammino è sempre possibile raggiungere uno stato nel quale vale p.

Esprimibile secondo la seguente formula $AG\ EF\ p$, per cui non esiste una formula LTL che possa esprimere la proprietà.

Al contrario la logica LTL può esprimere la seguente proprietà

In ogni cammino, prima o poi si raggiungerà uno stato a partire dal quale p rimane sempre vera.

Esprimibile secondo la seguente formula $F\ G\ p$, per cui non esiste una formula CTL che possa esprimere la proprietà.

Quindi CTL e LTL sono delle logiche temporali che hanno poteri espressivi diversi, ovvero esistono formule che possono essere espresse in una logica ma non nell'altra.

Per ovviare a questo problema si definisce l'estensione di CTL (CTL*), in particolare, si rimuove dalla logica CTL il vincolo di avere un quantificatore che precede ogni operatore temporale.

Lo svantaggio di questa logica è che richiede un maggiore sforzo computazionale per dimostrare le proprietà, sarà quindi importante decidere quale logica usare. Come abbiamo dimostrato quando sono equivalenti le FbF,

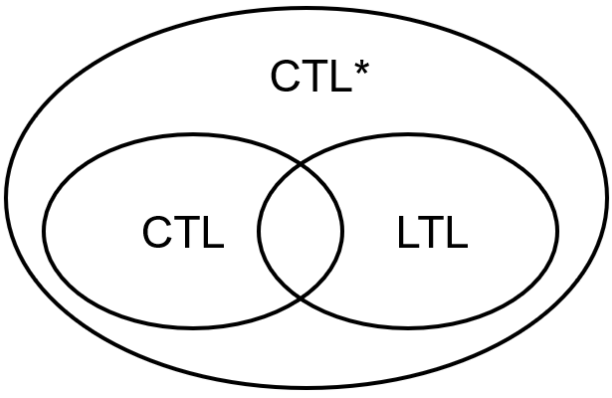


Figura 5.5: Relazione tra le logiche temporali

potremmo anche voler controllare se due modelli di Kripke sono equivalenti. Ragionevolmente si potrebbe pensare all'equivalenza dei modelli di Kripke solo quando sono isomorfi, ciò è troppo restrittivo.

Definizione 70 (**Equivalenza tra modelli**)**.** *Diremo due modelli M_1 e M_2, con stati iniziali q_0 e s_0, si dicono **equivalenti** rispetto a una logica L se, $\forall\alpha \in FbF_L$ vale che:*

$$M_1, q_0 \models \alpha \iff M_2, s_0 \models \alpha \tag{5.20}$$

Per dimostrare che due modelli sono equivalenti devo:

- Definire tutti i cammini massimali di M_1 e M_2 nella forma simile alle regex e si rimuovono i duplicati.
- Trasformare le espressioni ottenute sostituendo i nomi degli stati con le preposizioni atomiche che sono vere in un determinato stato.
- Se le espressioni dei cammini massimali di un modello sono uguali all'altro allora i due modelli sono equivalenti.

Esempio 17. *Vediamo ora un esempio per dimostrare che due modelli sono equivalenti. Consideriamo i modelli M_1 e M_2 riportati in figura 5.6. Per verificare l'equivalenza dobbiamo:*

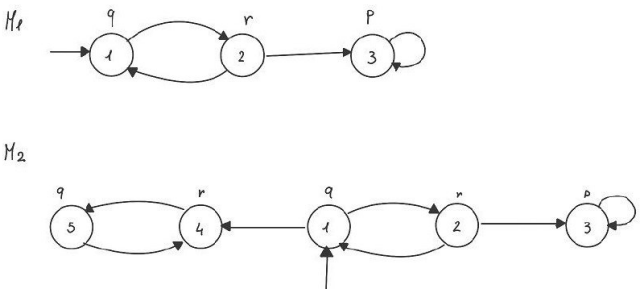


Figura 5.6: Modelli da confrontare

- *Costruire i cammini massimali dei due modelli.*
  - *Modello M_1:*
  $$\begin{array}{l}(1\ 2)^{\omega}\\(1\ 2)^{*}\ 3^{\omega}\end{array} \tag{5.21}$$
  - *Modello M_2:*
  $$\begin{array}{l}1\ (4\ 5)^{\omega}\\(1\ 2)^{\omega}\\(1\ 2)^{*}\ 3^{\omega}\\(1\ 2)^{*}\ 1\ (4\ 5)^{\omega}\end{array} \tag{5.22}$$
- *Sostituire i nomi degli stati con le preposizioni atomiche che sono vere in un determinato stato.*
  - *Modello M_1:*
  $$\begin{array}{l}(\{q\}\ \{r\})^{\omega}\\(\{q\}\ \{r\})^{*}\ \{p\}^{\omega}\end{array} \tag{5.23}$$
  - *Modello M_2:*
  $$\begin{array}{l}\{q\}\ (\{r\}\ \{q\})^{\omega} = (\{q\}\ \{r\})^{\omega}\\(\{q\}\ \{r\})^{\omega}\\(\{q\}\ \{r\})^{*}\ \{p\}^{\omega}\\(\{q\}\ \{r\})^{*}\ \{q\}\ (\{r\}\ \{q\})^{\omega} = (\{q\}\ \{r\})^{\omega}\end{array} \tag{5.24}$$
- *Verificare che le espressioni dei cammini massimali dei due modelli.*

Nota 15. *Il numero dei casi raggiungibili di una rete di petri è $\leq |P(c_{in})|$, se gli eventi abilitati sono indipendenti allora $|C_{\Sigma}| = |P(c_{in})|$.*

5.4 Insiemi parzialmente ordinati

Definizione 71 (**Relazione d'ordine parziale**). *Definiremo una **relazione d'ordine parziale** su un insieme A una relazione binaria definita come:*

$$\leq \subseteq A \times A \tag{5.25}$$

Allora $\leq$ deve soddisfare le seguenti proprietà:

- ***Riflessiva**: $\forall x \in A, x \leq x$*

- ***Antisimmetrica***: $\forall x, y \in A, x \leq y \wedge y \leq x \implies x = y$
- ***Transitiva***: $\forall x, y, z \in A, x \leq y \wedge y \leq z \implies x \leq z$

Inoltre $x \geq y \equiv y \leq x$, possiamo definire un ordine completo ovvero $x < y$ che vuol dire che $x \leq y \wedge x \neq y$.

Osservazione 11. *La più piccola relazione d'ordine parziale è la relazione identità.*

Raffinamento di una relazione d'ordine parziale:

Definizione 72 (**Maggiorante, Minorante**). *Sia $(A, \leq)$ un insieme parzialmente ordinato e $B \subseteq A$ allora:*

- *$x \in A$ è un **maggiorante** di B se $\forall y \in B, y \leq x$*
- *$x \in A$ è un **minorante** di B se $\forall y \in B, x \leq y$*

Può essere che $x \in B$ perché $B \subseteq A, x \in B \implies x \in A$.

Utilizziamo la notazione B^* per indicare l'insieme dei maggioranti di B, e B_* per indicare l'insieme dei minoranti di B. Inoltre, diremo che:

- B è **limitato superiormente** se $B^* \neq \emptyset$
- B è **limitato inferiormente** se $B_* \neq \emptyset$

Definizione 73 (**Massimo, Minimo, Massimale, Minimale**). *Sia $(A, \leq)$ un insieme parzialmente ordinato e $B \subseteq A$:*

- *Se $x \in B$ è il **minimo** di B se $x \leq y, \forall y \in B$, quindi x è anche un **minorante** per B. (max 1)*
- *Se $x \in B$ è il **massimo** di B se $x \geq y, \forall y \in B$, quindi x è anche un **maggiorante** per B. (max 1)*
- *Se $x \in B$ è il **minimale** in B se $x \geq y \implies x = y$.*
- *Se $x \in B$ è il **massimale** in B se $x \leq y \implies x = y$.*

Definizione 74 (**Estremo superiore, Estremo inferiore**). *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 $x \vee y$ per indicare $\bigvee B$, se esiste, e $x \wedge y$ per $\bigwedge B$.

Esempio 18. *Consideriamo i seguenti esempi:*

- *$(2^A, \subseteq)$: potremo dire che l'estremo superiore è l'unione mentre l'estremo inferiore è l'intersezione.*
- *$(\mathbb{N}^+, |)$: potremmo dire che l'estremo inferiore è MCD, Mentre l'estremo superiore è mcm.*

Le relazioni di ordine parziale possiamo rappresentarle come diagrammi di Hasse, in cui si rappresentano le relazioni mostrando i collegamenti col successivo, senza rappresentare la riflessività e la transitività.

Definizione 75 (**Reticolo**). *Definiremo un **reticolo** come un insieme parzialmente ordinato $(L, \leq)$, tale che, per ogni $x, y \in L$ esistono l'estremo superiore $(x \vee y)$ e l'estremo inferiore $(x \wedge y)$.*

Per i reticoli si può dimostrare per induzione che per ogni coppia esiste il join e il meet.

Definizione 76 (**Reticlo completo**). *Un reticolo si dice **completo** se per ogni sottoinsieme finito esistono il meet e join.*

Definizione 77 (**Funzione monotona**). *Siano due insiemi parzialmente ordinati $(A, \leq)$ e $(B, \leq)$. Definiremo una funzione di $f : A \to B$ si dice **monotona** se, per ogni $x, y \in A$, allora:*

$$x \leq y \implies f(x) \leq f(y) \tag{5.26}$$

In altri termini, dati due insiemi parzialmente ordinati allora una funzione è monotona se mantiene l'ordine dal dominio al codominio.

Definizione 78 (**Punto fisso**). *Data una funzione $f : X \to X$, un elemento $x \in X$ si dice **punto fisso** di f se $f(x) = x$.*

Esempio 19. *Vediamo alcuni esempi di punti fissi:*

- $f : \mathbb{R} \to \mathbb{R}$ $f(x) = x^2$ *ha due punti fissi,* 0 *e* 1*.*
- $g : \mathbb{R} \to \mathbb{R}$ $g(x) = \log(x)$ *non ha punti fissi* ($\emptyset$)*.*
- $h : \mathbb{R} \to \mathbb{R}$ $h(x) = x$ *l'insieme dei punti fissi è:* $\mathbb{R}$*.*

A noi interesseranno le funzioni monotone da un insieme su se stesso. Se $(A, \leq)$ è un insieme parzialmente ordinato, e $f : A \to A$ è una funzione monotona, possiamo chiederci se esistano un minimo e un massimo punto fisso.

Esempio 20. *Consideriamo* $A = 2^{\mathbb{N}}$ *e* $S \subseteq \mathbb{N}$*,* $f(S) = S \cup \{2, 7\}$*. Tutti i punti fissi sono gli insiemi che contengono* $\{2, 7\}$*, il minimo è* $\{2, 7\}$*, il massimo è* $\mathbb{N}$*.*

Teorema 8 (**Knaster-Tarski**)**.** *Sia* $(L, \leq)$ *un reticolo completo e* $f : L \to L$ *sia una funzione monotona. Allora* f *ha un minimo e un massimo punto fisso.*

Dimostrazione. Analizziamo la dimostrazione nel caso particolare in cui $L = \mathcal{P}(A)$ e $f : \mathcal{P}(A) \to \mathcal{P}(A)$ è monotona.

L'esistenza del minimo punto fisso viene dimostrata nei seguenti passaggi:

1. Costruiamo l'insieme $Z = \{T \subseteq A \mid f(T) \subseteq T\}$. Gli elementi di questo insieme Z saranno chiamati **punti pre-fissi**. È importante notare che l'insieme Z non può essere vuoto, in quanto $A \in Z$. Inoltre, se f ha dei punti fissi Z li contiene tutti.
2. A questo punto poniamo $m = \bigcap Z$ dal momento che $Z \neq \emptyset$
3. Per ogni $S \in Z$ allora $m \subseteq S$, per di più sapendo che f è monotona possiamo dire che:
$$f(m) \subseteq f(S) \tag{5.27}$$
4. Inoltre, per definizione di f abbiamo che $f(S) \subseteq S$ allora per la transitività di $\subseteq$ abbiamo:
$$f(m) \subseteq f(S) \subseteq S \tag{5.28}$$
5. Dal punto precedente possiamo arrivare a dire che $f(m) \subseteq \bigcap Z = m$, questo perché se $f(m)$ è un sottoinsieme di tutti i sottoinsiemi di Z sarà un sotto insieme della loro intersezione, quindi $m \in Z$ ed è il minimo.
6. Vogliamo ora dimostrare che sia punto fisso. Sapendo che $f(m) \subseteq m$ possiamo dire che:
$$f(f(m)) \subseteq f(m) \tag{5.29}$$
quindi per definizione di Z sappiamo che $f(m) \in Z$.
7. Dal momento che m è il più piccolo elemento di Z abbiamo:
$$m \subseteq f(m) \implies m = f(m) \tag{5.30}$$
quindi è un punto fisso, inoltre, è il punto fisso minimo.

Possiamo fare il ragionamento speculare per dimostrare il massimo punto fisso, in questo caso al posto che usare $\bigcap$ con $\bigcup$ e $\subseteq$ e poi avremo **post-fissi**.

L'esistenza del massimo punto fisso viene dimostrata nei seguenti passaggi:

1. Costruiamo l'insieme $Z = \{T \in Z \mid T \subseteq A \wedge T \subseteq f(T)\}$. Gli elementi di questo insieme Z saranno chiamati **punti post-fissi**. È importante notare che l'insieme Z non può essere vuoto, in quanto $A \in Z$. Inoltre, se f ha dei punti fissi Z li contiene tutti.
2. A questo punto poniamo $M = \bigcup Z$ dal momento che $Z \neq \emptyset$
3. Per ogni $S \in Z$ allora $S \subseteq M$, per di più sapendo che f è monotona possiamo dire che:
$$f(S) \subseteq f(M) \tag{5.31}$$

4. Inoltre, per definizione di f abbiamo che $S \subseteq f(S)$ allora per la transitività di $\subseteq$ abbiamo:
$$S \subseteq f(S) \subseteq M \tag{5.32}$$
5. Dal punto precedente possiamo arrivare a dire che $M = \bigcup Z \subseteq f(M)$, questo perché l'unione appartiene a Z per definizione, quindi $M \in Z$ ed è il massimo.
6. Vogliamo ora dimostrare che sia punto fisso. Sapendo che $M \subseteq f(M)$ possiamo dire che:
$$f(M) \subseteq f(f(M)) \tag{5.33}$$
quindi per definizione di Z sappiamo che $f(M) \in Z$.
7. Dal momento che M è il più grande elemento di Z abbiamo:
$$f(M) \subseteq M \implies M = f(M) \tag{5.34}$$
quindi è un punto fisso, inoltre, è il punto fisso massimo.

□

Sfruttando il teorema di Knaster-Tarski possiamo ottenere il seguente teorema:

Teorema 9 (**Teorema di Kleene**). *Sia $f : 2^A \to 2^A$ monotona allora diremo che f è **continua** se*
$$X_1 \subseteq X_2 \subseteq \cdots \subseteq X_n \subseteq \ldots \tag{5.35}$$
e come conseguenza del fatto che sia monotona si ha anche:
$$f(X_1) \subseteq f(X_2) \subseteq \cdots \subseteq f(X_n) \subseteq \ldots \tag{5.36}$$
allora:
$$f(\bigcup_{i=1}^{\infty} X_i) = \bigcup_{i=1}^{\infty} f(X_i) \tag{5.37}$$

Ci interessa la continuità della funzione perché, se f è continua, allora:

- Il minimo punto fisso di f si può ottenere calcolando ricorsivamente $f(\emptyset)$ fino a quando non troviamo il primo punto fisso.
$$f(\emptyset), f(f(\emptyset)), f(f(f(\emptyset))), \ldots \tag{5.38}$$
- Il massimo punto fisso di f si può ottenere calcolando ricorsivamente $f(A)$ fino a quando non troviamo il primo punto fisso.
$$f(A), f(f(A)), f(f(f(A))), \ldots \tag{5.39}$$

Possiamo legare questi teoremi alle logiche temporali CTL per avere un algoritmo di dimostrazione della formula. Supponiamo AFp, partiamo dall'insieme in cui tutti gli stati è vera p, allarghiamo questo insieme agli elementi che portano in questo insieme allora AFp vale sempre. Ripeto questa operazione fino a quando non posso aggiungere altri stati (punto fisso). Il problema è trovare questa funzione f che mi permetta di applicare questo algoritmo. Per una formula Gp è il contrario, si parte dall'insieme di stati per cui non vale p e ricorro da quali stati ricado in questa regione fino a quando non cambia nulla, il risultato è l'insieme complemento.

5.4.1 Algoritmo per CTL

Sia $M = (Q, T, I)$ un modello di Kripke, allora posso definire:

Definizione 79 (**Estensione**). *Sia α una formula CTL, definiamo l'**estensione** di α come:*
$$[[\alpha]] = \{q \in Q \mid M, q \models \alpha\} \tag{5.40}$$

A questo punto consideriamo la formula $\alpha \equiv AF\beta$, a cui è associata la funzione $f_\alpha : \mathcal{P}(Q) \to \mathcal{P}(Q)$. Per ogni $H \subseteq Q$ si ha:
$$f_\alpha(H) = [[\beta]] \cup \{q \in Q \mid \forall (q, q') \in T : q' \in H\} \tag{5.41}$$

Osservazione 12. $f_\alpha(\emptyset) = [[\beta]]$.

Definita in questo modo allora f è monotona e continua, quindi possiamo usare l'algoritmo per ottenere il minimo punto fisso di insiemi per cui vale la formula.

$$[[\alpha]] \text{ è il minimo punto fisso di } f_\alpha.$$

Col medesimo ragionamento possiamo considerare la formula $\alpha \equiv EG\beta$, a cui è associata la funzione $g_\alpha : \mathcal{P}(Q) \to \mathcal{P}(Q)$. Per ogni $H \subseteq Q$ si ha:

$$g_\alpha(H) = [[\beta]] \cap \{q \in Q \mid \exists (q, q') \in T : q' \in H\} \tag{5.42}$$

Osservazione 13. $g_\beta(Q) = [[\beta]]$.

In questo caso cercheremo il massimo punto fisso di stati per cui non vale la formula.

$$[[\alpha]] \text{ è il massimo punto fisso di } g_\alpha.$$

Dal momento che abbiamo visto l'insieme degli operatori minimali allora, ragionando secondo un algoritmo, allora definiremo solo una funzione per X e una per U per la dimostrazione e poi trasformeremo le formule in quegli operatori.

Nota 16. *Cerca la funzione che dimostra $A(\alpha U \beta)$ portalo all'orale.*

Definiamo una funzione per $\gamma \equiv A(\alpha U \beta)$, cioè f_γ:

$$f_\gamma(H) = [[\beta]] \cap \{q \in [[\alpha]] \mid \exists (q, q') \in T, q' \in H\} \tag{5.43}$$

5.5 Algoritmi per il model checking

Precedentemente è stato introdotto un algoirtmo per verificare se una formula della logica CTL è valida, ciò è possibile mediante un modello di Kripke e una funzione monotona, utilizzando i punti fissi.

Questo ragionamento non si può riutilizzare per la dimostrazione di formule della logica LTL, dal momento che non vale più il ragionamento dei punti fissi.

Nota 17. *Non possiamo applicare l'algoritmo per le formule della logica CTL per la dimostrazione delle formule della logica LTL.*

Vogliamo quindi trovare un algoritmo che dato un modello di Kripke, definito sulle proposizioni atomiche AP, e una formula della logica LTL ci dice se tale formula è verificata nello stato iniziale.

Tale algoritmo si basa su un automa a stati finiti che riconosce parole infinite su un alfabeto finito Σ, tale automa è anche noto come **automa di Büchi**.

Definizione 80 (**Automa di Büchi**). *Definiamo l'**automa di Büchi** come una quadrupla:*

$$\mathcal{B} = (Q, q_0, \delta, F) \tag{5.44}$$

dove:

- *Q è un insieme finito di stati, anche dette locations.*
- *$q_0 \in Q$ è lo stato iniziale.*
- *$\delta \subseteq Q \times \Sigma \times Q$ è una relazione di transizione.*
- *$F \subseteq Q$ è l'insieme degli stati accettanti.*

Si trasforma il problema in un'accettazione di linguaggi di parole infinite, riconoscibili mediante gli automi sopracitati. Però in questi linguaggi cambia il metodo di accettazione delle parole, dal momento che l'esecuzione dell'automa non può finire in uno stato accettante.

Definizione 81 (**Accettazione di una parola infinita**). *Una parola infinita $w = a_0 a_1 \dots$ è accettata da $\mathcal{B}$ se la sequenza corrispondente di stati $q_0 q_1 \dots$ passa infinite volte per almeno uno stato in F.*

Questi automi condividono molte delle proprietà fondamentali dei linguaggi formali, ma non tutte, infatti si hanno anche le seguenti proprietà:

- le versioni **deterministiche** e **non deterministiche** non sono equivalenti

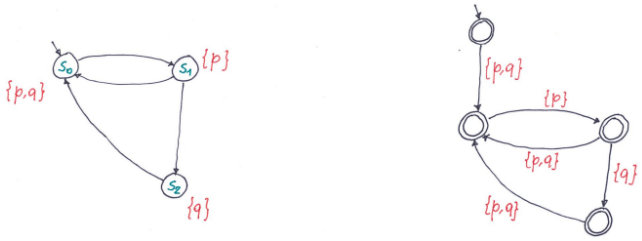


Figura 5.7: Passaggio da modello di Kripke ad automa

- il problema di riconoscere il linguaggio vuoto è un problema **decidibile** ($L(\mathcal{B}) = \emptyset$).

Nota 18. *Il linguaggio composto da parole infinite viene anche chiamato Ω- regolare.*

Sono stati introdotti questi automi perché quando si effettua la dimostrazione di una formula LTL, quello che si fa è analizzare stringhe infinite di preposizioni atomiche della logica dei cammini massimali. Questo implica una correlazione tra i **modelli di Kripke** e gli **automi di Büchi**. Partendo da un modello di Kripke posso ottenere un automa di Büchi corrispondente nel seguente modo:

- Copio la struttura del modello di Kripke.
- Aggiungo uno stato iniziale.
- Etichetto gli archi dell'automa di Büchi con l'insieme delle preposizioni atomiche vere nello stato di arrivo corrispondente nel modello di Kripke.

Esempio 21. *Nella figura 5.7 è riportato un esempio di come è possibile passare da un modello di Kripke ad un automa di Büchi.*

Inoltre, per ogni formula LTL è possibile definire un automa di Büchi corrispondente al modello di Kripke associato alla formula.
Vogliamo ora verificare se una formula $\alpha \in LTL$ è vera nel modello M considerando lo stato q_0 (M, q_0). Per ottenere questo risultato possiamo utilizzare il seguente algoritmo:

1. Costruiamo l'automa $\mathcal{B}_{\neg\alpha}$, ovvero l'automa per la negazione della formula.
2. Trasformiamo il modello di Kripke M nell'automa di Büchi equivalente.
3. Calcoliamo il prodotto sincrono dei due automi ($\mathcal{PS}$,prodotto tra automi).
4. A questo punto se $L(\mathcal{PS}) = \emptyset$, allora $M, q_0 \models \alpha$. Se invece tale linguaggio non è vuoto, vuol dire che abbiamo trovato un cammino per cui è vera la negazione della formula α e quindi α non è vera per quel modello.

Nota 19. *Ricorda che Il prodotto sincrono tra automi corrisponde ad un automa che riconosce il linguaggio intersezione dei linguaggi accettati dagli automi*

5.6 μ calcolo

Il μ calcolo è un linguaggio logico che permette di definire delle formule ricorsive.

Esempio 22. *Supponiamo di avere un solo operatore temporale X. Come possiamo esprimere le proprietà $EF\alpha$?*

$$\begin{aligned} EF\alpha \quad &\equiv \alpha \vee EX\alpha \vee EXEX\alpha \vee \cdots \equiv \\ &\equiv \alpha \vee EX(\alpha \vee EX\alpha \vee EXEX\alpha \vee \dots) \equiv \\ &\equiv \alpha \vee EX(EF\alpha) \end{aligned} \tag{5.45}$$

La semantica di questo linguaggio è definita su modelli di Kripke attraverso gli operatori di punto fisso.
Questo linguaggio ha massima potenza espressiva, ma ha lo svantaggio di avere un elevata complessità. Questo ci permette di affermare che:

$$CTL^* \subset \mu - \text{calcolo} \tag{5.46}$$

In generale questo linguaggio ha un'applicabilità minore vista la sua esprissività e complessità.

5.7 Complessità

Sia M un modello di Kripke e f una formula, e siano $|M|$ il numero di stati del modello e $|f|$ il numero di simboli nella formula. Allora possiamo definire la complessità delle logiche definite come:

- CTL: $\mathcal{O}(|M| \times |f|)$
- LTL: $\mathcal{O}(|M| \times 2^{|f|})$

Anche se dalle formule sembrano complessità completamente diverse, il valore di maggiore peso è composto dalla dimensione del modello, la quale risulta esponenziale rispetto al numero di stati. Inoltre, le formule LTL sono solitamente composte da meno simboli rispetto alle formule CTL.

Nota 20. *Ricorda che nei sistemi concorrenti gli stati esplodono sempre in un esponenziale*

Esistono anche altri algoritmi per la dimostrazione di formule:

- OBDD: sfruttano i diagrammi di decisione binari ordinati che rappresentano le formule in modo simbolico e risparmiano l'esplosione esponenziale degli stati
- ordine parziale: si rappresentano i sistemi concorrenti secondo un ordine parziale
- SAT: si riduce il problema di dimostrazione a SAT e quindi si sfruttano gli algoritmi esistenti per la dimostrazione.

5.8 Fairness

Un altro problema dei sistemi concorrenti è che la rappresentazione utilizzata sfrutta la **semantica interleaving** che trasforma le relazioni di dipendenza in scelte.

Definizione 82 (**Esecuzione unfair**). *Un'esecuzione è **unfair** (ingiusta) se un evento rimane abilitato da un istante in poi, ma non scatta mai.*

La possibilità di avere esecuzioni unfair porta a un nuovo problema, ovvero si vuole limitare la valutazione di una formula all'esecuzione fair.

Possiamo distinguere un esecuzione fair in:

- **Strong fair**: un'esecuzione si dice fortemente fair se:

$$GF(t \text{ abilitata}) \to GF(t \text{ scatta}) \equiv GF(c[t >) \to GF(c[t > c') \tag{5.47}$$

  ovvero, se un evento t è abilitato infinite volte, allora t scatta infinite volte.

- **Weak fair**: un'esecuzione si dice debolmente fair se un evento t non è sempre abilitato e le volte che è abilitato non scatta.