Modelli della Concorrenza

Tommaso Ferrario

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:

$$s_1 : x = 2 \ \ \{x = V\} \ x = 2 \ \{x = 2\}$$

$$s_2 : x = 3 \ \ \{x = V\} \ x = 3 \ \{x = 3\}$$

$$\{x = V\} \ s_1; \ s_2 \ \{x = 3\}$$

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

$$s_1' : \{x = V\} \ x = 1; \ x = x + 1 \ \{x = 2\}$$

$$\{x = V\} \ s_1'; \ s_2 \ \{x = 3\}$$

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 del simbolo $|$:

$$p_1|p_2$$

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:*

$$p_1 = \{x = V\}x = 2\{x = 2\}$$
$$p_2 = \{x = V\}x = 3\{x = 3\}$$

con V che indica un qualunque valore.

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

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

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

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

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

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

Hoare ha introdotto un nuovo paradigma di programmazione, il paradigma **CSP** (*Communicating Sequential Processes*), il linguaggio macchina dei cosiddetti transputer. 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, con lo scambio di informazioni. Viene fatto anche un processo particolare rappresentante la memoria condivisa. Avremo quindi:

$$x \mid p_1 \mid p_2$$

dove x che rappresenta la memoria condivisa dai due processi.

Milner propose infatti il lambda calcolo (λ-calcolo) per passare dal sequenziale al concorrente. 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 lo scambio sincrono di messaggi, con il modello hand-shacking.

Per la gestione di questo scambio di messaggi 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. I processi possono interagire anche con l'ambiente esterno. Dato un sistema P, si scrive:

$$P = p_1 \mid p_2 \mid p_3$$

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.

Grazie alla comunicazione con l'ambiente esterno non si ha più un sistema chiuso.

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 infatti l'**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.

Vedremo quindi:

- Il calcolo "puro" di sistemi comunicanti CCS, definendone la semantica attraverso **LTS** (*Labeled Transition System*), rappresentando i processi come nodi e le azioni come archi etichettati.
- L'equivalenza all'osservazione e la bisimulazione.

2.1 Labeled Transition System (LTS)

I **Labeled Transition System** (LTS) sono molto usati per rappresentare sistemi concorrenti. Questo modello ha origine dal modello degli automi a stati finiti, i quali sono però usati come riconoscitori di linguaggi.

Negli LTS non si ha l'obbligo di avere un insieme finito di stati.

Definizione 1. *Possiamo definire un LTS come una quadrupla:*

$$LTS = (S, Act, T, s_0)$$

dove

- S*: rappresenta un insieme di stati.*
- Act*: è un insieme di nomi di azioni.*
- T*: è una relazione definita come:*

  $$T \subseteq S \times Act \times S$$

  tale che:

  $$\{(s, a, s') \mid s, s' \in S \ \wedge \ a \in Act\}$$

  e può essere rappresentata come:

  $$s \xrightarrow{a} s'$$

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

La transizione $s \xrightarrow{a} s'$ può essere estesa a $w \in Act^*$ avendo più azioni se e solo se:

- se $w = \varepsilon$ allora $s \equiv s'$
- se $w = a \cdot x$ con $a \in Act$ e $x \in Act^*$ se e solo se:

  $$s \xrightarrow{a} s'' \text{ e } s'' \xrightarrow{x} s'$$

Ho che $s \to s'$ se e solo se $\exists a \in Act$ tale che $s \xrightarrow{a} s'$. Quindi ho:

$$\to = \bigcup_{a \in Act} \xrightarrow{a}$$

Ho che $s \xrightarrow{*} s'$ se e solo se $\exists w \in Act^*$ tale che $s \xrightarrow{w} s'$. Si ha che:

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

$$\xrightarrow{*} \subseteq S \times S$$

La relazione $\xrightarrow{*}$ è la chiusura riflessiva e transitiva della relazione $\to$ tale relazione non è simmetrica, avendo sempre $s \xrightarrow{*} s$ ed essendo garantita la transitività.

2.2 Calculus of Communicating Systems (CCS)

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

- *K, ovvero un insieme di nomi di processi che possono anche essere simboli di un alfabeto.*
- *A, ovvero un insieme di nomi di azioni, che sono o azioni di sincronizzazione con l'ambiente o le componenti del sistema.*
- *$\overline{A}$, ovvero l'insieme di nomi delle coazioni contenute in A, $\forall\ a\ \in A\ \exists\ \overline{a}\ \in\ \overline{A}$ quindi:*

  $$\overline{A} = \{\overline{a} \mid a\ \in\ A\}$$

  e ovviamente si ha nel:

  $$\overline{\overline{a}} = a$$

- *$Act = A\ \cup\ \overline{A}\ \cup\ \{\tau\}$ dove $\tau \notin A$ corrisponde all'azione di sincronizzazione tra a e $\overline{a}$, ovvero la sincronizzazione è avvenuta. Le prime due 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.

Definizione 3. *I processi CCS sono di fatto operazioni CCS e un sistema CCS è definito da una collezione di processi $p \in K$ e si avranno:*

$$p = espressione\ CCS$$

e si avrà solo un'equazione $\forall p \in K$.

Definiamo meglio i processi CCS e, ad ogni processo, abbiniamo l'LTS, che sarà una struttura del tipo:

$$(Proc_{CCS}, Act, T, p_0)$$

dove la transizione T viene rappresentata con delle regole di inferenza strutturate nel seguente modo:

$$\frac{Premesse}{Conseguenze}$$

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

Un processo CCS può essere:

- **Nil** o 0 che ha un solo stato senza transazioni.

Figura 2.1: Rappresentazione grafica dell'operazione Nil

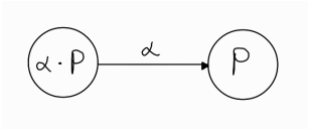

Figura 2.2: Rappresentazione grafica dell'operatore prefisso

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

$$\frac{}{\alpha \cdot P \xrightarrow{\alpha} P}$$

- **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 $p_1', p_2' \in Proc_{CCS}$.

$$\frac{p_1 + p_2 \xrightarrow{\alpha} p_1'}{p_1 + p_2 \xrightarrow{\alpha} p_1'}$$

oppure

$$\frac{p_1 + p2 \xrightarrow{\beta} p_2'}{p_1 + p_2 \xrightarrow{\beta} p_2'}$$

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

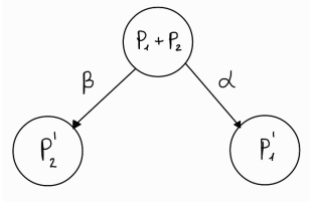

Figura 2.3: Rappresentazione grafica dell'operatore prefisso

processi:

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

Nel caso di $I = \emptyset$ avrò $\sum_{i \in I} p_i = Nil$

Nell'applicazione di questa operazione possono presentarsi situazioni di non determinismo, ad esempio avendo $p_1 = \alpha \cdot p_1'$ e $p_2 = \alpha \cdot p_2'$.

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

$$\frac{p_1 \xrightarrow{\alpha} p_1'}{p_1 | p_2 \xrightarrow{\alpha} p_1' | p_2}$$

oppure

$$\frac{p_2 \xrightarrow{\alpha} p_2'}{p_1 | p_2 \xrightarrow{\alpha} p_1 | p_2'}$$

oppure

$$\frac{p_1 \xrightarrow{\alpha} p_1' \wedge p_2 \xrightarrow{\overline{\alpha}} p_2'}{p_1 | p_2 \xrightarrow{\tau} p_1' | p_2'}$$

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

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

$$p_{\setminus L}$$

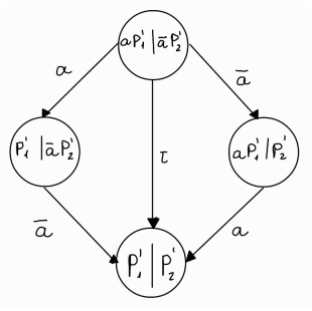

Figura 2.4: Rappresentazione grafica dell'operatore prefisso

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.

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

- **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:

$$f : Act \to Act \tag{2.1}$$

inoltre, deve sempre essere garantito che:

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

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

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

- $K = P$ con K uguale al nome di processo e $P \in Proc_{CCS}$.

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

Quindi dato un CCS posso associare un LTS per quanto riguarda la semantica. Si ha una precedenza degli operatori, da quello con più precedenza a quello con meno precedenza:

1. Restrizione
2. Rietichettatura
3. Prefisso
4. Composizione parallela
5. Somma

Esempio 2 (**Priorità degli operatori**). *Avendo:*

$$R + a \cdot p | b \cdot Q_{\setminus L}$$

sarebbe:

$$R + ((a \cdot p) | b \cdot (Q_{\setminus L}))$$

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

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

tale che sia:

- Riflessiva.
- Simmetrica.
- Transitiva.

Bisognerà inoltre astrarre:

- Gli stati e considerare le azioni Act
- Dalle sincronizzazioni interne, ovvero dalle τ
- Rispetto al non determinismo

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

Definizione 4. *Una relazione di equivalenza R è una **congruenza** se e solo se:*

$$\forall p, q \in Proc_{CCS} \wedge \forall c[\cdot] \text{ contesto } CCS$$

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

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

Sia $LTS_1 = (Q_1, Act_1, T_1, q_{01})$ e $LTS_1 = (Q_2, Act_2, T_2, q_{02})$ posso affermare che LTS_1 è **isomorfo** a LTS_2 se e solo se: $\alpha : Q_1 \to Q_2$ e $\beta : Act_1 \to Act_2$ sono corrispondenze biunivoche. Se valgono le seguenti assunzioni:

- $\alpha(q_{01}) = q_{02}$
- $(q_1, a, q_1') \in T_1$ allora $(\alpha(q_1), \beta(a), \alpha(q_1')) \in T_2$

Teorema 1. *Dati due processi p_1 e p_2 a cui assegniamo i due LTS LTS_1 e LTS_2. I due processi sono **equivalenti** se LTS_1 e LTS_2 sono isomorfi.*

Oltre all'isomorfismo, due processi sono equivalenti se i due LTS ammettono le stesse sequenze di operazioni, prendendo l'equivalenza forte tra automi a stati finiti. Questa è detta **equivalenza rispetto alle tracce**, ovvero due programmi sono equivalenti se implementano la stessa sequenza di istruzioni.

Preso $p \in Proc_{CCS}$ posso definire l'insieme delle tracce di p come:

$$tracce(p) = \{w \in Act^* \mid \exists p' \in Proc_{CCS} \ p \xrightarrow{w} p'\} \tag{2.2}$$

Suppongo, per ora, di non considerare le sincronizzazioni (quindi senza τ).

- Se $w = \varepsilon$ allora $p = p'$
- Se $w = x_1 \cdot x_2 \cdot \dots \cdot x_n$ con $x_i \in Act$ $\exists p_1', p_2', \dots, p_n' \in Proc_{CCS}$ tale che:

$$p \xrightarrow{x_1} p_1' \xrightarrow{x_2} \dots \xrightarrow{x_n} p_n' = p'$$

Teorema 2. *Preso $p' \in Proc_{CCS}$ è **equivalente rispetto alle tracce** a p'', indicato come:*

$$p' \overset{T}{\sim} p''$$

se e solo se:

$$tracce(p') = tracce(p'') \tag{2.3}$$

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

2.2.1 Bisimulazione forte

Dato che le tracce non sono sufficienti nel caso di processi concorrenti, si definisce il concetto di **bisimulazione**.

Definizione 5 (**Relazione di Bisimulazione Forte**). *Data una relazione binaria $R \subseteq Proc_{CCS} \times Proc_{CCS}$ è una relazione di **bisimulazione (forte)** 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*

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

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

$$\exists p' \text{ tale che } p \xrightarrow{\alpha} p' \text{ e si ha } p'Rq'$$

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:

$$pRq$$

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

Teorema 3. *Se prendo $\overset{Bis}{\sim} \subseteq Proc_{CCS} \times Proc_{CCS}$ si dimostra che è:*

- *Riflessiva.*
- *Simmetrica.*
- *Transitiva.*

e quindi è una relazione di equivalenza. Quindi:

$$p \overset{Bis}{\sim} q \iff \forall \alpha \in Act \text{ se } p \xrightarrow{\alpha} p'$$

allora

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

e se:

$$q \xrightarrow{\alpha} q'$$

allora

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

Teorema 4. *Se due processi sono fortemente bisimili allora sono sicuramente equivalenti rispetto alle tracce. Non vale il viceversa.*

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 3. *Consideriamo i processi $p_1 = a \cdot b \cdot Nil + a \cdot c \cdot Nil$ e $p_2 = a \cdot (b \cdot Nil + c \cdot Nil)$*
Osserviamo subito che i due processi sono equivalenti rispetto alle tracce, difatti:

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

Vediamo se i due processi sono anche bisimili.

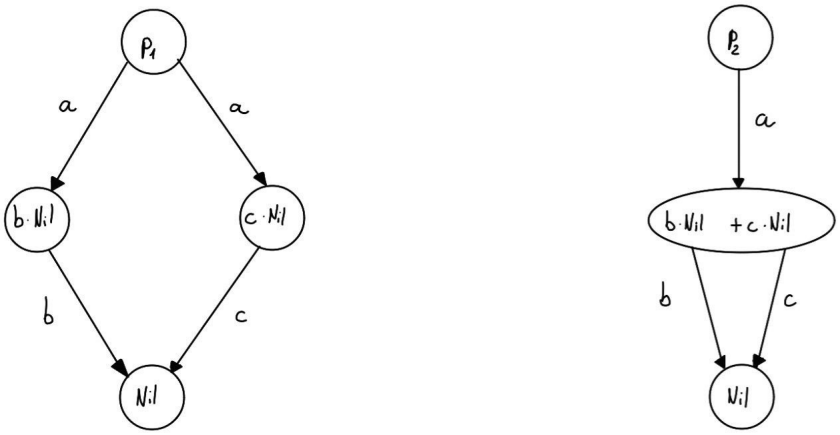

Figura 2.5: LTS dei processi p_1 e p_2

- *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 $b \cdot Nil$ e $b \cdot Nil + c \cdot Nil$: 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:

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

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

Esempio 4. *Consideriamo due processi che simulano dei buffer che possono contenere due elementi:*

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

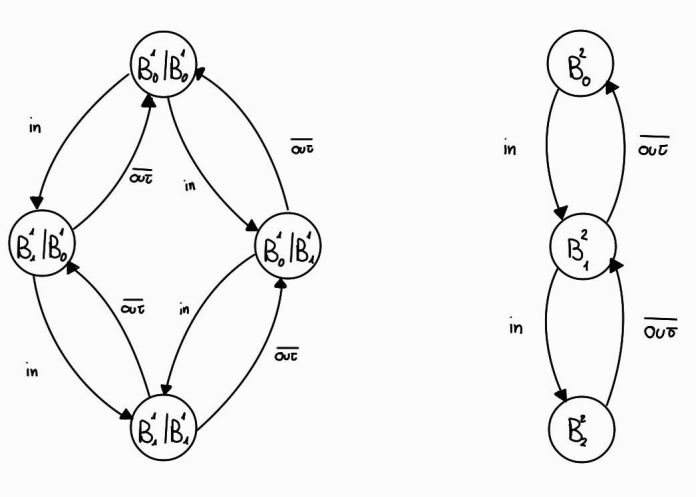

Figura 2.6: LTS dei processi $B_0^1|B_0^1$ e B_0^2

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 $B_0^1|B_0^1$.

Cerco quindi di astrarre dalle interazioni interne (τ), introducendo l'equivalenza debole, volendo astrarre rispetto alle azioni τ , introducendo il concetto di bisimulazione debole.

Vediamo quindi le proprietà della bisimulazione forte:

- La bisimulazione forte è una congruenza rispetto agli operatori del CCS. Quindi se ho $p \overset{Bis}{\sim} q$ con $p, q \in Proc_{CCS}$ vale:
$$\alpha \cdot p \overset{Bis}{\sim} \alpha \cdot q \; \forall \; \alpha \in Act$$
- Vale la proprietà commutativa rispetto alla composizione $+$:
$$p + r \overset{Bis}{\sim} q + r \;\wedge\; r + p \overset{Bis}{\sim} r + q, \quad \forall r \in Proc_{CCS}$$
- Vale la proprietà commutativa rispetto alla composizione $|$:
$$p|r \overset{Bis}{\sim} q|r \;\wedge\; r|p \overset{Bis}{\sim} r|q, \quad \forall r \in Proc_{CCS}$$
- Per ogni funzione di rietichettatura $f : Act \to Act$ vale:
$$p_{[f]} \overset{Bis}{\sim} q_{[f]}$$
- $p_{\backslash L} \overset{Bis}{\sim} q_{\backslash L}$

A queste proprietà possiamo aggiungere delle *leggi* legate al fatto che gli operatori hanno delle proprietà. Dati $p, q, r \in Proc_{CCS}$ valgono le seguenti proprietà:

- **Commutatività**:
$$p + q \overset{Bis}{\sim} q + p \;\wedge\; p|q \overset{Bis}{\sim} q|p$$
- **Distributività**:
$$(p + q) + r \overset{Bis}{\sim} p + (q + r) \;\wedge\; (p|q)|r \overset{Bis}{\sim} p|(q|r)$$
- **Leggi di assorbimento**:
$$p + Nil \overset{Bis}{\sim} p \;\wedge\; p|Nil \overset{Bis}{\sim} p$$

2.2.2 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 mi obbliga a modificare la definizione della funzione di transizione. La relazione di transizione debole è definita come:

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

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

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

Come fatto per la relazione forte, definiamo la relazione di transizione per sequenze di azioni $w \in Act^*$:

$$p \overset{w}{\Rightarrow} p'$$

se e solo se:

- Se $w = \varepsilon$ oppure $w = \tau^*$ allora ho:
$$p \xrightarrow{\tau}{}^{*} p'$$
- Se $w = a_1 \dots a_n$ con $a_i \in A \cup \overline{A}$ allora:
$$p \overset{a_1}{\Rightarrow} p_1 \overset{a_2}{\Rightarrow} \dots \overset{a_n}{\Rightarrow} p'$$
dove ogni a_i può essere preceduto/seguito da una qualsiasi sequenza di τ.

Definizione 6. *Definiamo l'**equivalenza debole rispetto alle tracce**, la quale è rappresentata come:*

$$p \overset{T}{\approx} q$$

se e solo se:

$$Tracce_{\Rightarrow}(p) = Tracce_{\Rightarrow}(q)$$

ovvero:

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

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

Posso definire le tracce come:

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

Definizione 7. *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:*

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

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

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

Esempio 5. *Consideriamo i processi $r = a \cdot (b \cdot Nil + \tau \cdot c \cdot Nil)$ e $k = a \cdot (b \cdot Nil + \tau \cdot c \cdot Nil) + a \cdot c \cdot Nil$.*

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

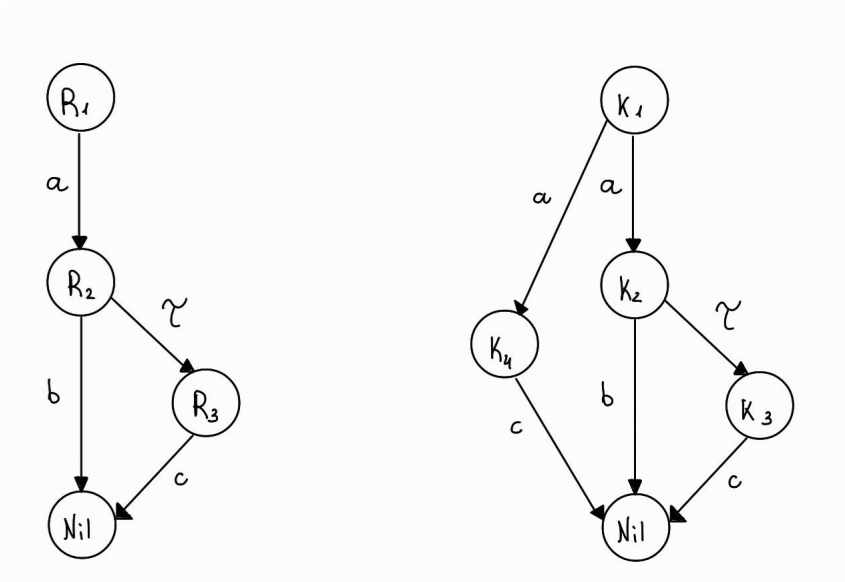

Figura 2.7: LTS di r e k

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

Teorema 6.

- *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.

2.3 Proprietà

Siano $p, q \in Proc_{CCS}$:

- Se $LTS(p)$ è isomorfo a $LTS(q)$ allora $p \simeq q$.
- Deve astrarre dagli stati.
- Se $p \simeq q$ allora le tracce di p e q sono equivalenti:

$$Tracce(p) = Tracce(q)$$

- Se $p \simeq q$ 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: deve essere possibile sostituire un sotto-processo con un suo equivalente senza modificare il comportamento complessivo del sistema.

Siano $p, q \in Proc_{CCS}$ l'equivalenza rispetto alle tracce forte gode delle seguenti proprietà:

- Se $LTS(p)$ è isomorfo a $LTS(q)$ allora $p \overset{T}{\sim} q$.
- Astrae dagli stati.
- Se $p \overset{T}{\sim} q$ se e solo se le tracce di p e q sono equivalenti:

$$Tracce(p) = Tracce(q)$$

- $\overset{T}{\sim}$ è una congruenza rispetto agli operatori CCS.
- Non garantisce di preservare il deadlock o l'assenza di deadlock, nell'interazione con l'ambiente.

Siano $p, q \in Proc_{CCS}$ la bisimulazione forte gode delle seguenti proprietà:

- Se $LTS(p)$ è isomorfo a $LTS(q)$ allora $p \overset{Bis}{\sim} q$.
- Astrae dagli stati.
- Se $p \overset{Bis}{\sim} q$ se le tracce di p e q sono equivalenti:

$$Tracce(p) = Tracce(q)$$

- $\overset{Bis}{\sim}$ è una congruenza rispetto agli operatori CCS.
- Preservare il deadlock o l'assenza di deadlock, nell'interazione con l'ambiente.

L'equivalenza rispetto alle Tracce forte è più restrittiva dell'equivalenza rispetto alle Tracce debole. Siano $p, q \in Proc_{CCS}$ l'equivalenza rispetto alle tracce debole gode delle seguenti proprietà:

- Se $LTS(p)$ è isomorfo a $LTS(q)$ allora $p \overset{T}{\approx} q$.
- Astrae dagli stati.
- Se $p \overset{T}{\approx} q$ se e solo se le tracce di p e q sono equivalenti:

$$Tracce(p) = Tracce(q)$$

- $\overset{T}{\approx}$ è una congruenza rispetto agli operatori CCS.

- Non garantisce di preservare il deadlock o l'assenza di deadlock, nell'interazione con l'ambiente.

La bisimulazione forte è più restrittiva della bisimulazione debole:

$$p \overset{Bis}{\sim} q \Rightarrow p \overset{Bis}{\approx} q \ (\overset{Bis}{\sim} \subseteq \overset{Bis}{\approx})$$

La bisimulazione forte (debole) è più restrittiva dell'equivalenza rispetto alle Tracce forte (debole)

$$p \overset{Bis}{\sim} q \Rightarrow p \overset{T}{\sim} q \ e \ p \overset{Bis}{\approx} q \Rightarrow p \overset{T}{\approx} q$$

$$(\overset{Bis}{\sim} \subseteq \overset{T}{\sim}) \ e \ (\overset{Bis}{\approx} \subseteq \overset{T}{\approx})$$

Definizione 8. *$p \in Proc_{CCS}$ è un processo deterministico se e solo se vale che:*

$$\forall x \in Act = A \cup \overline{A} \cup \{\tau\}, \ \textit{se } p \xrightarrow{*} p' \ \textit{e } p \xrightarrow{*} p'' \ \textit{allora } p' = p''$$

Siano $p, q \in Proc_{CCS}$ se p e q sono deterministici e $p \overset{T}{\sim} q$ ($p \overset{T}{\approx} q$) allora $p \overset{Bis}{\sim} q$ ($p \overset{Bis}{\approx} q$).
Siano $p, q \in Proc_{CCS}$ la bisimulazione debole gode delle seguenti proprietà:

- È un'equivalenza, la più grande relazione di Bisimulazione debole.
- Astrae da azioni non osservabili (τ) e dai cicli inosservabili (τ *loop*).
- Preservare il deadlock o l'assenza di deadlock, nell'interazione con l'ambiente.

Teorema 7. *Se $p, q \in Proc_{CCS}$ tale che $p \overset{Bis}{\approx} q$, allora:*

- $\alpha \cdot p \overset{Bis}{\approx} \alpha \cdot q \quad \forall \alpha \in A_{CCS} = A \cup \overline{A} \cup \tau$
- $p|r \overset{Bis}{\approx} q|r \ \wedge \ r|p \overset{Bis}{\approx} r|q \ \forall r \in Proc_{CCS}$
- $p_{[f]} \overset{Bis}{\approx} q_{[f]} \ \forall f$ *funzione di etichettatura.*
- $p_{\backslash L} \overset{Bis}{\approx} q_{\backslash L} \ \forall L \subseteq A$

La Bisimulazione debole è una congruenza rispetto agli operatori del CCS diversi da + e ricorsione.
Posso quindi affermare che la Bisimulazione debole non è una congruenza per il CCS.

2.4 Congruenza

Si definisce quindi, tramite assiomi, la più grande relazione di congruenza $\overset{C}{\approx}$ (per il CCS puro e senza ricorsione, con agenti finiti) che è contenuta nella relazione di Bisimulazione $\overset{Bis}{\approx}$.

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

insieme finito di Assiomi Ax:

- 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. $p + (q + r) \overset{C}{\approx} (p + q) + r$ e $p|(q|r) \overset{C}{\approx} (p|q)|r$
2. $p + q \overset{C}{\approx} q + p$ e $p|q \overset{C}{\approx} q|p$
3. $p + p \overset{C}{\approx} p$ (ma $p|p \overset{C}{\not\approx} p$)
4. $p + Nil \overset{C}{\approx} p$ e $p|Nil \overset{C}{\approx} p$
5. $p + \tau \cdot p \overset{C}{\approx} \tau \cdot p$

6. $\mu \cdot \tau p \overset{C}{\approx} \mu \cdot p$
7. $\mu \cdot (p + \tau \cdot q) \overset{C}{\approx} \mu \cdot (p + \tau \cdot q) + \mu \cdot q$
8. Se p e q sono delle somme: $p = \sum_i \alpha_i \cdot p_i$ e $q = \sum_j \beta_j \cdot q_j$, $\alpha, \beta \in Act$:

$$p|q \overset{C}{\approx} \sum_i \alpha_i(p_i|q) + \sum_j \beta_j \cdot (p|q_j) + \sum_{\alpha_i = \overline{\beta}_j} \tau \cdot (p_i|q_j)$$

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

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 9 (**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:*

  $\square$

- *$F \subseteq (B \times E) \cup (E \times B)$ 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, in un tale sistema, eventi isolati o condizioni isolate. Si ha, formalmente, che:

  $$dom(F) \cup ran(F) = B \cup E \tag{3.2}$$

  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 10 (**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:*

$$ {}^\bullet e \subseteq c \ \wedge \ e^\bullet \cap c = \emptyset \tag{3.3} $$

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

$$ c' = (c -^\bullet e) \cup e^\bullet \tag{3.4} $$

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

- *N è **semplice** se e solo se:*

$$ \forall x, y \in B \cup E, \ {}^\bullet x = {}^\bullet y \ \wedge \ x^\bullet = y^\bullet \Rightarrow x = y \tag{3.5} $$

- *N è **pura** se e solo se:*

$$ \forall e \in E : \ {}^\bullet e \cap e^\bullet = \emptyset \tag{3.6} $$

Definizione 12. *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:*

$$ \forall e_1, e_2 \in U : \ e_1 \neq e_2 \Rightarrow ({}^\bullet e_1 \cup e_1^\bullet) \cap ({}^\bullet e_2 \cup e_2^\bullet) = \emptyset \tag{3.7} $$

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

$$ U \text{ insieme di eventi indipendenti } \wedge \ \forall e \in U : \ c[e > \tag{3.8} $$

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

$$ c_1[U > \ \wedge \ c_2 = (c_1 - {}^\bullet U) \cup U^\bullet \tag{3.9} $$

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.

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 $c' \in C_\Sigma$

U_Σ è l'**insieme dei passi** di Σ:

$$U_\Sigma = \{U \subseteq E | \exists c, c' \in C_\Sigma : c[U > c'\} \tag{3.10}$$

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:

$$c_{in}[e_1 > c_1[e_2 > \ldots [e_n > c_n \text{ oppure } c_{in}[e_1 e_2 \ldots e_n > c_n \tag{3.11}$$

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

$$c_{in}[U_1 > c_1[U_2 > \ldots [U_n > c_n \text{ oppure} c_{in}[U_1 U_2 \ldots U_n > c_n \tag{3.12}$$

- **Comportamento non sequenziale**: processi non sequenziali ("partial order semantics" - "true concurrency"). Si definiscono processi non sequenziali. 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.

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:

$$A = \{(c, U, c') | c, c' \in C_\Sigma, U \in U_\Sigma, c[U > c'\} \tag{3.13}$$

3.1.1 Diamond property

Definizione 13 (**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:*

$$^\bullet e_1 \cap e_2^\bullet = \emptyset \ \wedge \ ^\bullet e_2 \cap e_1^\bullet = \emptyset \tag{3.14}$$

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:

$$^\bullet e_1 \cap ^\bullet e_2 = \emptyset \ \wedge \ e_1^\bullet \cap e_2^\bullet = \emptyset \tag{3.15}$$

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.

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.*

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.

Figura 3.1: Diamond property 1

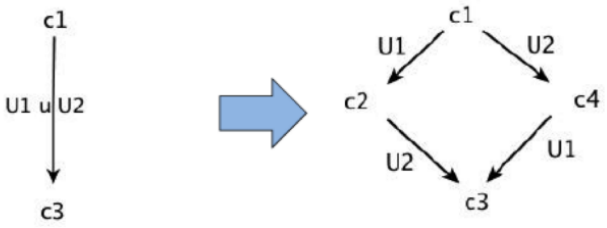

Figura 3.2: Diamond property 2

Definizione 14 (**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 15 (*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 16 (**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:*

$$^\bullet e \subseteq c \ \wedge \ e^\bullet \cap c \neq \emptyset \tag{3.16}$$

Definizione 17 (**Sistema senza contatti**). *Un sistema elementare $\Sigma = (B, E, F; c_{in})$ è **senza contatti** se e solo se:*

$$\forall e \in E, \ \forall c \in C_\Sigma \text{ si ha } {}^\bullet e \subseteq c \Rightarrow e^\bullet \cap c = \emptyset \tag{3.17}$$

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

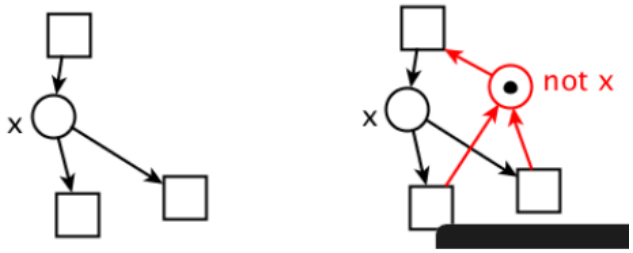

Figura 3.3: Complemento dell'operazione x.

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

$$c[e > \text{ se e solo se } {}^\bullet e \subseteq c \tag{3.18}$$

Definizione 18 (**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:*

$$c[e_1 > \ \wedge \neg c[e_2 > \ \wedge \ c[e_1 e_2 > \ \ (c[e_1 > c'[e_2 >) \tag{3.19}$$

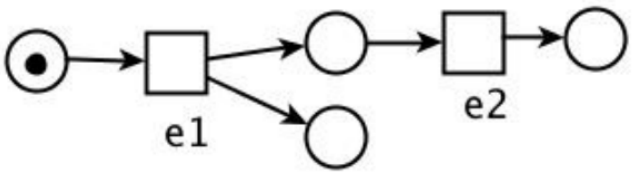

Figura 3.4: Rappresentazione della sequenza

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

Definizione 19 (**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:*

$$c[\{e_1, e_2\} > \tag{3.20}$$

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

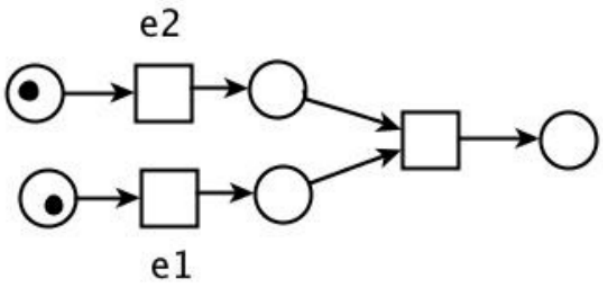

Figura 3.5: Rappresentazione della concorrenza

Definizione 20 (**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:*

$$c[e_1 > \ \wedge\ c[e_2 > \ \wedge\ \neg c[\{e_1, e_2\} > \tag{3.21}$$

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

Figura 3.6: Rappresentazione di un conflitto

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

Definizione 21 (**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 sincrona.
2. La composizione asincrona.
3. La composizione mista, tra sincrona e asincrona.

3.2 Processi non sequenziali

Definizione 22 (**Rete causale**)**.** *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).*
- $\forall x, y \in B \cup E : (x, y) \in F^+ \Rightarrow (y, x \notin F^+)$ *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 essere quindi si 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**:

$$(X, \leq) = (B \cup E, F^*) \tag{3.22}$$

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

Definizione 23. *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.*

Definizione 24. *Data una rete causale $N = (B, E, F)$ e dato un ordine parziale $(X, \leq)$ con $X = B \cup E$ definiamo:*

$$C \subseteq X \tag{3.23}$$

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* $\forall y \in X \cap C$ *si ha che:*

$$\exists c \in C : y \not\textbf{co}\ c \tag{3.24}$$

Quindi in C *definito o come* ***co-set*** *o come* ***taglio*** *si ha che vale la transitività. Definiamo:*

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

come:

- ***li-set*** *se e solo se* $\forall x, y \in L :\ x$ ***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:*

$$\exists l \in L :\ y \not\textbf{li}\ l \tag{3.26}$$

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 25. *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 26 (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:*

$$\forall h \in Linee(N),\ \forall c \in Tagli(N) :\ |h \cap c| = 1 \tag{3.27}$$

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.*

Definizione 27 (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 si* 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
- $\forall K \in B$, K B-taglio di N è tale che: K è finito e $\exists c \in C_\Sigma :\ \phi(K) = c$

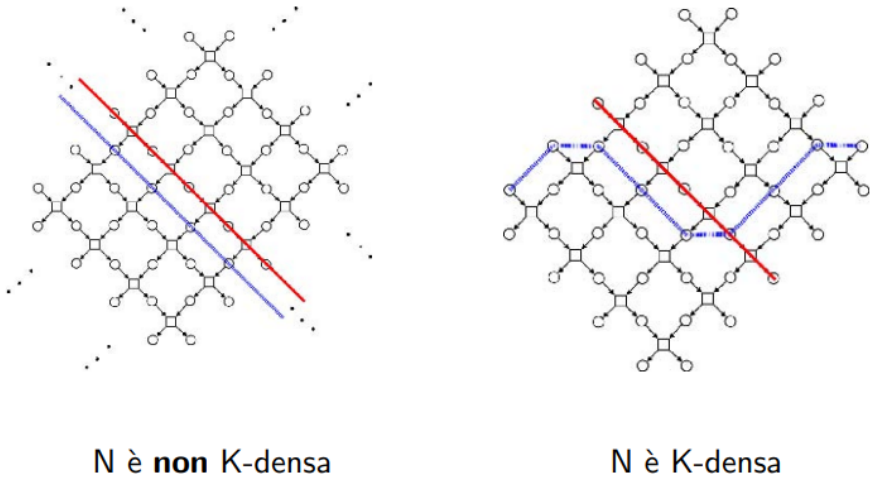

Figura 3.7: Esempio di rete K-densa

3.2.1 Reti di Occorrenze

Definizione 28 (**Rete di occorrenze**). *$N = (B, E, F)$ è una **rete di occorrenze** se e solo se:*

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

  $\#\subseteq X \times X$ *dove* $X = B \cup E$ *è definita come* : $x \# y$ *se e solo se* $\exists e_1, e_2 \in E: \ {}^\bullet e_1 \cap {}^\bullet e_2 \neq \emptyset \wedge e_1 \leq x \wedge e_2 \leq y$ (3.28)

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

Definizione 29 (**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 ${}^\bullet$ 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}.*

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 3. *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 30 (**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).*

Definizione 31. *Σ 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 32. *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 1. *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 33 (**Sistema Posti e Transizioni**)**.** *Formalmente un sistema posti e transizioni è definito come:*

$$\sigma = (S, T, F, K, W, M_0) \tag{3.29}$$

dove:

- (S, T, F) *è una rete.*
- $K : S \to \mathbb{N}^+ \cup \infty$ *è la funzione che assegna ai posti le capacità*
- $W : F \to \mathbb{N}$ *è la funzione peso degli archi*
- $M_0 : S \to \mathbb{N} : \forall s \in S \; M_0(s) \leq K(s)$ *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 34 (**Regola di scatto**)**.**

$$M[t > \text{ se e solo se } \forall s \in S \; M(s) \geq W(s,t) \; \wedge \; M(s) \text{ è } W(t,s) \leq K(s) \tag{3.30}$$

$$M[t > M' \text{ se e solo se } M[t > \; \wedge \forall s \in S \; M'(s) = M(s) - W(s,t) + W(t,s) \tag{3.31}$$

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.

Definizione 35 (**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.*

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.

Da completare perché mancano le slide

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.
- $\neg,\ \wedge,\ \vee,\ \rightarrow,\ \leftrightarrow$ sono i connettivi logici, utilizzati per creare formule complesse.
- $(,\ )$ sono i demilitatori, utilizzati per la creazione di formule complesse.

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

Definizione 36 (**Formule ben formate**). *Definiamo l'insieme delle formule ben formate F_p come:*

- $\bot,\ \top \in F_p$ *le costanti logiche sono delle formule ben formate.*
- $\forall p_i \in PA,\ p_i \in F_p$ *le proposizioni atomiche sono formule ben formate.*
- *Se A, B inF_p allora:*
$$(\neg A),\ (A \wedge B),\ (A \vee B),\ (A \rightarrow B),\ (A \leftrightarrow B)\ \in F_p \tag{4.1}$$
- *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	$A \wedge B$	$A \vee B$	$\neg A$	$A \to B$	$A \longleftrightarrow B$
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:

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

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 **tautlogia** se $I_v(A) = 1$ per ogni I_v
- A è una **contraddizione** se $I_v(A) = 0$ per ogni I_v

Definizione 37 (Logicamente equivalenti). *Due formule ben formate A e B sono logicamente equivalenti, indicato con $A \equiv B$, se e solo se:*

$$I_v(A) = I_v(B) \quad \forall I_v \tag{4.6}$$

Definizione 38. ***[Modello]*** *Definiamo ora i modelli, ovvero delle interpretazioni delle proposizioni atomiche, cioè una scelta di valori di verità per tutte le proposizioni atomiche.*

Un modello è un sottoinsieme delle proposizioni atomiche $M \subseteq PA$ a cui è associata un'interpretazione definita come $I_M : F_p \to \{0, 1\}$ 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 39. *Se $M \models A$ per tutti gli M, allora A è una tautologia e si indica $\models A$.*

Definizione 40. *Se $M \models A$ per qualche M, allora A è soddisfacibile.*

Definizione 41. *Se $M \models A$ non è soddisfatta da nessun M, allora A è in soddisfacibile.*

Definizione 6. Definizione 7.

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 $A_i \in F_p$ sono le premesse, mentre $B \in F_p$ 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:

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

- Modus Tollens:

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

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

Definizione 42 (**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 8 (**(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 9 (**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 vera 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 postcondizioni che sono delle 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 43 (**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 pre-cndizioni.
- 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:

$$\forall z \in \mathbb{Z}\ z \in E\ \wedge\ \forall e_1, e_2 \in E\ (e_1 + e_2), (e_1 - e_2), -e_1, (e_1 * e_2), (e_1/e_2), (e_1 \% e_2) \in E \tag{4.16}$$

- Espressioni logiche B definite come:
- Comandi C definiti come:
  - Assegnamento $x := e\ \in C$ 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 $B \in C$ è un espressione logica e C è un comando.
  - skip $\in C$ questa istruzione non modifica la memoria.

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

$$\frac{T_1, \ldots, T_m, f_1, \ldots, f_n}{T} \tag{4.17}$$

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. Per il comando **skip** definiamo la seguente regola di derivazione:

$$\frac{}{\{p\}\ \text{skip}\ \{p\}} \tag{4.18}$$

   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. Per il comando **sequenza** definiamo la seguente regola di derivazione:

$$\frac{\{p\}\ \text{C}\ \{p'\}\quad \{p'\}\ \text{D}\ \{q\}}{\{p\}\ \text{C; D}\ \{p\}} \tag{4.19}$$

   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. Per il comando **scelta** definiamo la seguente regola di derivazione:

$$\frac{\{p \wedge B\}\ \text{C}\ \{q\} \quad \{p \wedge \neg B\}\ \text{D}\ \{q\}}{\{p\}\ \text{if B then C else D endif}\ \{q\}} \tag{4.20}$$

   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 C 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. Per il comando **implicazione** definiamo la seguente regola di derivazione:

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

   inoltre:

$$\frac{\{p\}\ \text{C}\ \{q\} \quad q \to q'}{\{p\}\ \text{C}\ \{q'\}} \tag{4.22}$$

   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. Per il comando **assegnamento** definiamo la seguente regola di derivazione:

$$\frac{}{\{q[E/q]\}\ \text{x}\ :=\ \text{E}\ \{q\}} \tag{4.23}$$

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

6. Per il comando **iterazione** (correttezza parziale) definiamo la seguente regola di derivazione:

$$\frac{\{inv \wedge B\}\ \text{C}\ \{inv\}}{\{inv\}\ \text{while B do C endwhile}\ \{inv \wedge \neg B\}} \tag{4.24}$$

   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 violata guardando lo stato della memoria durante l'esecuzione del blocco, ma viene ripristinata al termine del blocco.

Definizione 44 (**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:

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

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

$$\models \to \vdash \tag{4.26}$$

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:

$$\vdash \{p\}\ \mathrm{C}\ \{q\} \tag{4.27}$$

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

Usiamo anche:

$$\models \{p\}\ \mathrm{C}\ \{q\} \tag{4.28}$$

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 ha 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

$\vdash$ può avere a pedice una sigla per la regola rappresentata.

4.2.4 Correttezza totale

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

$$\{p\}\ \text{while B do C endwhile}\ \{q\} \tag{4.29}$$

Definizione 45 (**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"*

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

Definizione 46 (**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"*

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

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

Allora:

$$\overset{tot}{\vdash}\ \{inv\}\ W\ \{inv \wedge \neg B\} \tag{4.30}$$

Nota 4. *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 generale 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.

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

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 $(i \wedge \neg B) \to s\ (\{i\}\ W;\ Z\ \{q\})$. Infine, cerchiamo una formula u tale che $\{p\} \vee \{u\}$ è $u \to i\ (\{p\}\ V;\ W;\ Z\ \{q\})$.

4.2.6 Proprietà

La logica di Hoare gode delle proprietà di:

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

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

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

Per fare ciò definiamo:

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

$$\sigma \models p \tag{4.31}$$

  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:

- $m(\neg q) = \Sigma \backslash m(q)$
- $m(p \vee q) = m(p) \cup m(q)$
- $m(p \wedge q) = m(p) \cap m(q)$
- $m(p \rightarrow q) = m(\neg p) \cup m(q)$, 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.

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

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

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 10. $\models \{p\}C\{q\}$ *se e solo se* $p \rightarrow wp(S, q)$

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