Metodi Formali

UniShare

Davide Cozzi
@dlcgold

Indice

1 Introduzione **2**
1.1 Contenuti del Corso 2

2 Sviluppo di Modelli e Sistemi **3**
2.1 Sistemi Elementari 6
2.1.1 Diamond Property 19
2.1.2 Isomorfismo tra Sistemi di Transizione Etichettati . . . 22
2.1.3 Il Problema della Sintesi 23
2.1.4 Contatti 24
2.1.5 Situazioni Fondamentali 26
2.1.6 Sottoreti 32
2.1.7 Operazioni di Composizione per Reti di Petri 34
2.2 Reti Posti e Transizioni 37
2.2.1 I Filosofi a Cena 40
2.2.2 Formalizzazione delle Reti Posti e Transizioni 46
2.2.3 Reti Marcate 55
2.2.4 Proprietà di Comportamento 56
2.2.5 Analisi Strutturale 64

3 Logica PLTL **75**
3.1 Ripasso Logica Proposizionale 75
3.1.1 Sintassi 76
3.1.2 Semantica 78
3.1.3 Equivalenze Logiche 79
3.2 Logica PLTL 81
3.2.1 Sintassi 83
3.2.2 Semantica 85

Capitolo 1

Introduzione

Questi appunti sono presi a lezione. Per quanto sia stata fatta una revisione è altamente probabile (praticamente certo) che possano contenere errori, sia di stampa che di vero e proprio contenuto. Per eventuali proposte di correzione effettuare una pull request. Link: `https://github.com/dlcgold/Appunti`.
Grazie mille e buono studio!

1.1 Contenuti del Corso

Il corso tratta di metodi e tecniche formali per specificare, disegnare e analizzare sistemi complessi, in particolare sistemi concorrenti e distribuiti costituiti da componenti che operano in modo indipendente e che interagiscono tra loro.

Si usa un linguaggio logico che spiega il comportamento di tali sistemi e fa riferimento alla **logica temporale** di tali sistemi, in quanto le proprietà di tali sistemi sono tali per cui evolvono con il cambiamento di stato del sistema e quindi serve una logica che descriva le proprietà dell'evoluzione del comportamento.
Si parlerà delle **Reti di Petri**, ovvero uno strumento per modellare tali sistemi concorrenti e distribuiti. Questo modello ha intrinsechi dei teoremi matematici atti a studiare il comportamento di tali sistemi.
In laboratorio si studieranno algoritmi e strumenti software per la modellazione e l'analisi di tali sistemi.
Si introducono a che sistemi dinamici a tempi discreti, come gli **automi cellulari**.

Capitolo 2

Sviluppo di Modelli e Sistemi

Si hanno diverse fasi di sviluppo di *sistemi complessi* (nel nostro caso **concorrenti** e **distribuiti**). Si hanno 4 grandi fasi (che riprendono le generiche fasi dello sviluppo software), che non seguono una rigida sequenza cronologica tra di loro:

1. specifica del problema e delle proprietà della soluzione
2. modellazione della soluzione
3. implementazione
4. verifica, validazione e collaudo, sia sul modello che implementazione (con eventuali modifiche)

Queste fasi possono alternarsi a vicenda.
I metodi formali possono svolgere una parte rilevante in tutte queste 4 fasi e hanno la prerogativa di sviluppare questi sistemi in maniera corretta e persistente.
Ci si focalizza sulla modellazione e sulla specifica delle proprietà. Si studia inoltre la verifica delle proprietà sul modello costruito. *In questo corso si lascia un attimo da parte l'aspetto implementativo, che comunque seguirebbe alla verifica e alla validazione del metodo.*
Si hanno diversi modelli di sistemi concorrenti e distribuiti, presenti in letteratura:

- **Algebre di Processi**, ovvero una miriade di diversi linguaggi, studiate inizialmente da Milner, che introdusse il calcolo dei sistemi comunicanti, un calcolo algebrico utile alla semantica della concorrenza. Inoltre Hoare ha introdotto i **processi sequenziali comunicanti** come un nucleo di linguaggio di programmazione,

usato come linguaggio macchina per le prime macchine parallele. Queste algebre si basano sul paradigma di avere un forte aspetto della **composizionalità**, in quanto un sistema viene visto come costituito da diverse componenti autonome (sia hardware, che software, che umane) che interagiscono tra loro sincronizzandosi (in modo sincrono, *handshaking*, sfruttando un "canale di comunicazione" che viene modellato come un processo) e scambiandosi messaggi. Questo paradigma è anche alla base dello sviluppo di molti linguaggi di programmazione specificatamente dedicati alla concorrenza.

- **Automi a Stati Finiti**. Un modello concorrente e distribuito viene spesso rappresentato attraverso **sistemi di transizioni etichettati**, che sono una derivazione del modello degli automi a stati finiti, già usati in letteratura per modellare reti neurali, progettare circuiti asincroni, modellare macchine a stati finiti, riconoscere linguaggi regolari (il teorema di Kleene ci ricorda che *ad un automa a stati finiti è possibile associare un'espressione regolare*) e per la modellazione di protocolli di comunicazione.

- **Reti di Petri**, introdotte da Petri con la **teoria generale delle reti di Petri** nella sua tesi di dottorato. Questa teoria parte da una critica al modello a stati finiti dove il focus è su stati globali e trasformazione di stati globali. Petri cercava invece una teoria matematica (fondata sui principi della fisica moderna della relatività e della quantistica) che fosse una teoria dei sistemi in grado di descrivere sistemi complessi in cui mettere al centro il flusso di informazione e che potesse permettere di analizzare l'organizzazione dal punto di vista del flusso di informazione che passa da una componente all'altra. Non si ha il focus, quindi, su "macchine calcolatrici" ma come supporto alla comunicazione in organizzazioni complesse. Si hanno quindi diversi elementi chiave:

  - la comunicazione
  - la sincronizzazione tra componenti
  - il flusso di informazione che passa tra le varie componenti
  - la relazione di concorrenza e l'indipendenza causale tra i vari eventi che comportano i cambiamenti di stato. Ci si concentra su stati locali e non sulla visione di una sequenza di azioni e di uno stato globale

La teoria delle reti di Petri è stata poi sviluppata e ha avuto diverse applicazioni. Sono stati sviluppati diversi linguaggi, ovvero diverse **classi di reti di Petri** per descrivere un sistema complesso a livelli differenti di astrazione.
Sono state anche sviluppate tecniche formali di analisi e di verifica del modello (disegnato mediante reti di Petri), basate sulla teoria dei grafi e sull'algebra lineare.
Le reti di Petri hanno avuto un notevole utilizzo in diversi ambiti applicativi anche estranei all'informatica pura e allo studio della concorrenze, come la modellazione di sistemi biologici o la modellazione di reazioni chimiche. Mediante una classe di reti particolare, le **reti stocastiche** si può valutare le prestazioni di un determinato modello.

Sistemi di Transizioni Etichettati

Definizione 1. *I sistemi di transizione etichettati sono definiti come gli automi a stati finiti ma senza essere visti come riconoscitori di linguaggi infatti un sistema è formato da un insieme, solitamente finito, di stati globali S. Si ha poi un alfabeto delle possibili azioni che può eseguire il sistema. Si hanno anche delle relazioni di transizioni, ovvero delle transizioni che permettono di specificare come, attraverso un'azione, si passa da uno stato ad un altro. Le transizioni si rappresentano con archi etichettati tra i nodi, che rappresentano gli stati. Le etichette degli archi rappresentano le azioni necessarie alla trasformazione. L'insieme delle azioni viene chiamato E mentre $T \subseteq S \times E \times S$ è l'insieme degli archi etichettati. Può essere, opzionalmente, individuato uno stato iniziale s_0. Un sistema non è obbligato a "terminare", quindi non si ha obbligatoriamente uno stato finale.*
Riassumendo quindi un sistema di transizione etichettato è un quadrupla:

$$A = (S, E, T, s_0)$$

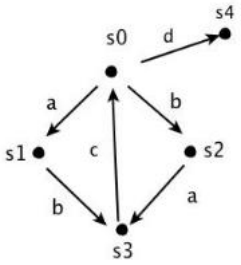


Figura 2.1: Esempio di sistema di transizione etichettato

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 (si possono avere più assi temporali in un sistema distribuito). Quindi la simulazione sequenziale non deterministica (emantica 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.

2.1 Sistemi Elementari

Per introdurre i sistemi elementari delle reti di Petri, ovvero una classe molto semplice e astratta partiamo da un esempio:

Esempio 1. *Vediamo l'esempio del Produttore e del Consumatore.*
Si ha un sistema con una componente Produttore che produce elementi e li deposita in un buffer che ha un'unica posizione (quindi o è pieno o è vuoto) e con un consumatore che preleva dal buffer un elemento per poi consumarlo ed essere pronto a prelevare un altro elemento. Si ha un comportamento ciclico. Usiamo quindi le reti di Petri, col modello dei sistemi elementari, per rappresentare questo modello. Bisogna quindi individuare le proprietà fondamentali locali del sistema.
Partiamo dal produttore, che può avere 2 stati locali:

1. *pronto per produrre*
2. *pronto per depositare*

*Usiamo i **cerchi** per rappresentare condizioni locali che sono associabili a delle proposizioni della logica che possono essere vere o false. Queste preposizioni sono quindi stati locali. Gli eventi locali vengono invece rappresentati con un **rettangolo**. Un evento ha un arco entrante da uno stato che rappresenta le precondizioni di quell'evento (che devono essere vere per permettere l'occorrenza dell'evento). L'occorrenza dell'evento rende false le precondizioni e rende vere le postcondizioni (che sono stati raggiungibili con un arco uscente da un evento). Si ha quindi che il produttore può depositare solo se il buffer non è pieno, quindi le postcondizioni di un evento devono essere false affinché l'evento possa occorrere (oltre alle precondizioni vere).*
Passiamo al consumatore che estrae solo se il buffer è pieno ed è pronto a prelevare. Si procede poi con la stessa logica del produttore di cambiamento tra vero e falso delle varie condizioni locali.

In questo esempio si hanno quindi condizioni che sono preposizioni booleane e rappresentano stati locali.

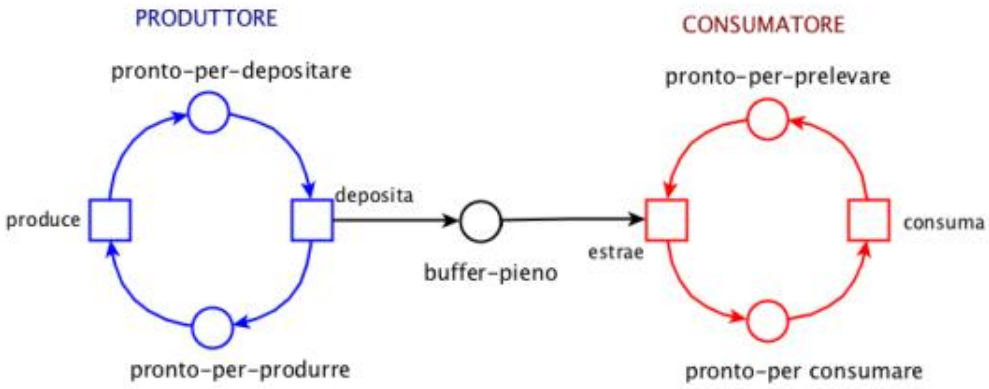


Figura 2.2: Produttore e Consumatore

Lo stato globale del sistema è dato da una collezione di stati locali. Per segnare tali condizioni mettiamo un punto pieno dentro il cerchio e queste condizioni "abilitano" i vari eventi: Si può arrivare ad una configurazione

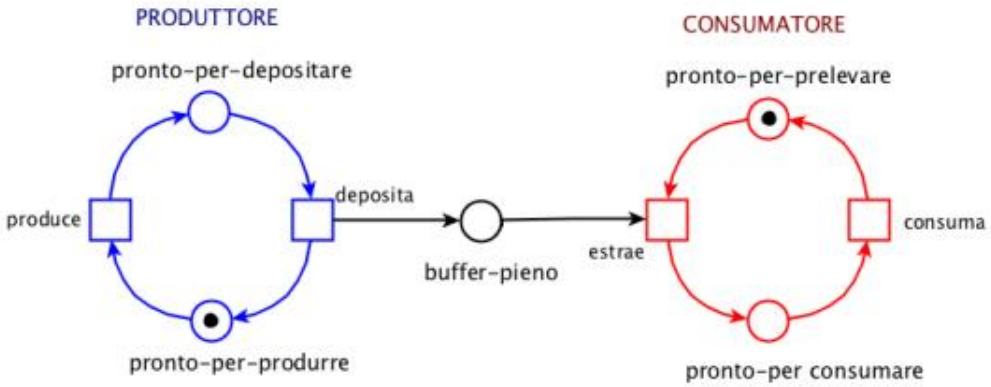


Figura 2.3: Uno stato globale Produttore e Consumatore dove l'evento *produce* è l'unico abilitato

dove, per esempio, sia l'evento produce, del produttore, che l'evento preleva, del consumatore, sono abilitati. Si ha quindi che i due eventi possono

occorrere in modo ***concorrente*** *infatti i due eventi sono* ***indipendenti*** *in quanto condizionati da* ***precondizioni e postcondizioni completamente disgiunte****. Due eventi che occorrono in maniera concorrente lo possono fare in qualsiasi ordine, non si ha infatti una sequenza temporale specifica tra i due.*
In questo sistema quindi siano solo stati locali ed eventi localizzati e non stati ed eventi globali. Un evento dipende solo dalle sue precondizioni e dalle sue postcondizioni.
Se rappresentiamo con delle marche le condizioni vere possiamo simulare il comportamento del sistema con il gioco delle marche che mostra come l'evoluzione delle condizioni avviene all'occorrenza degli eventi.
La simula di un tale sistema può comunque avvenire con un sistema di transizioni etichettato, ovvero con un automa a stati finiti, che rappresenta gli stati globali corrispondenti alle diverse combinazioni di stati locali che di volta in volta sono veri. Gli archi vengono etichettati con gli eventi che comportano un cambiamento di stato globale:

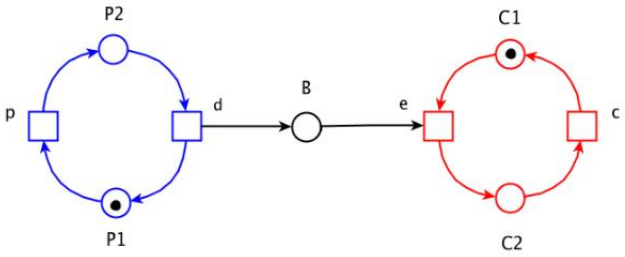


Figura 2.4: Semplificazione della nomenclatura del sistema per praticità

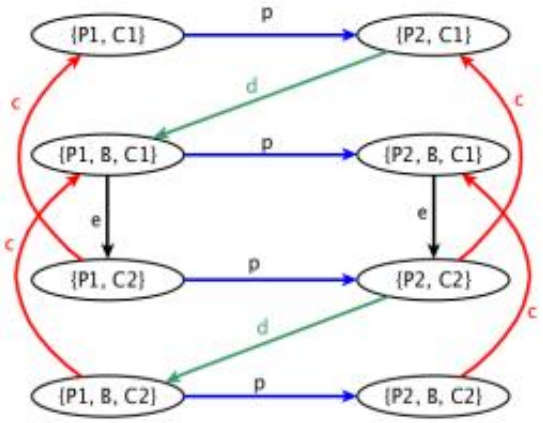


Figura 2.5: Rappresentazione del sistema con un automa a stati finiti che rappresenta stati globali

Passiamo ora alla formalizzazione di questi aspetti.

Definizione 2. *Una **rete elementare** è definita come una tripla:*

$$N = (B, E, F)$$

dove:

- *B è un insieme finito di **condizioni**, ovvero stati locali, preposizioni booleane etc.... Vengono rappresentate con un cerchio*
- *E è un insieme finito di **eventi**, ovvero trasformazioni locali di stato e transizioni locali. Vengono rappresentate con un quadrato*
- *F è una **relazione di flusso** che connette condizioni ad eventi ed eventi a condizioni. Si ha quindi che:*

  $$F \subseteq (B \times E) \cup (E \times B)$$

  *Le relazioni di flusso sono rappresentate da archi orientati. Inoltre la relazione di flusso è tale per cui non esistano **elementi isolati**, in quanto non avrebbero senso, in un tale sistema, eventi isolati (che non modificherebbero mai una condizione) o condizioni isolate (che non verrebbero mai modificate da un evento). Si ha, formalmente, che:*

  $$dom(F) \cup ran(F) = B \cup E$$

  chiedere per formula sopra

Si ha che:

$$B \cap E = \emptyset$$

$$B \cup E \neq \emptyset$$

Ovvero gli insiemi delle condizioni e degli eventi sono tra loro disgiunti e non vuoti.
Sia ora x un elemento qualsiasi della rete, ovvero x può essere o una condizione o un evento, formalmente:

$$x \in B \cup E$$

si ha che:

- ${}^\bullet x = \{y \in X : (y, x)\}$ *rappresenta l'insieme di tutti gli elementi* y *che sono connessi dalla relazione di flusso ad* x*, ovvero si ha un arco da* y *a* x*. Sono quindi i* ***pre-elementi*** *di* x*, ovvero le precondizioni, se* x *è un evento, o i pre-eventi, se* x *è una condizione*

- $x^\bullet = \{y \in X : (x, y)\}$ *rappresenta l'insieme di tutti gli elementi* y *che sono connessi dalla relazione di flusso a partire da* x*, ovvero si ha un arco da* x *a* y*. Sono quindi i* ***post-elementi*** *di* x*, ovvero le postcondizioni, se* x *è un evento, o i post-eventi, se* x *è una condizione*

Posso estendere questa notazione ad insiemi di elementi. Sia A *un insieme qualsiasi di elementi, che possono quindi essere sia condizioni che eventi:*

$$A \subseteq B \cup E$$

Si ha quindi che i pre-elementi dell'insieme A *sono rappresentati con:*

$${}^\bullet A = \cup^\bullet_{x \in A} x$$

ovvero l'unione dei pre-elementi di ogni singolo elemento dell'insieme A*.*
Analogamente si ha che i post-elementi dell'insieme A *sono rappresentati con:*

$$A^\bullet = \cup_{x \in A} x^\bullet$$

ovvero l'unione dei post-elementi di ogni singolo elemento dell'insieme A*. 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 pre-condizioni 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 é definito attraverso le nozioni di **caso (o configurazione)** e di **regola di scatto (o di transizione)**.
Una rete può anche essere suddivisa in sotto-reti, seguendo l'esempio sopra si potrebbe avere una sotto-rete per il produttore, una per il consumatore e anche una per il buffer.

Definizione 3. *Un* ***caso*** *(o* ***configurazione****) é 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.*
Graficamente le condizioni vere presentano un puntino in mezzo al cerchio mentre le condizioni false solo un cerchio vuoto

Definizione 4. *Sia $N = (B, E, F)$ una rete elementare e sia $c \subseteq B$ una certa configurazione (non serve quindi necessariamente conoscere tutto lo stato del sistema). La* ***regola di scatto*** *mi permette di stabilire quando un evento $e \in E$ è abilitato, ovvero può occorrere, in c sse:*

$$^\bullet e \subseteq c \ e \ e^\bullet \cap c = \emptyset$$

ovvero sse tutte le precondizioni dell'evento sono vere (e quindi sono contenute nella configurazione c) e sse tutte le postcondizioni sono false (quindi non si hanno intersezioni tra le postcondizioni e la configurazione).
L'occorrenza (l'abilitazione) di e in c si denota con la scrittura:

$$c[e >$$

Se un evento e è abilitato in c, ovvero $c[e >$, si ha che quando e occorre in c genera un nuovo caso c' e si usa la notazione:

$$c[e > c'$$

Si ha quindi che c' è così calcolabile:

$$c' = (c -^\bullet e) \cup e^\bullet$$

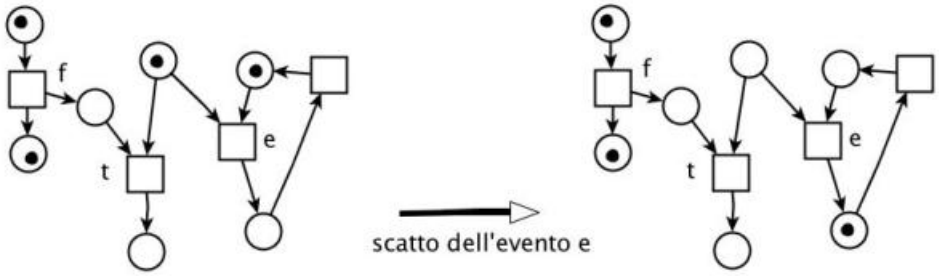
Ovvero togliendo da c tutte le precondizioni dell'evento e e aggiungendo quindi tutte le postcondizioni di e

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.

L'importante è che le precondizioni di un evento siano vere e le postcondizioni false (siamo comunque interessati solo alla validità delle condizioni che riguardano l'evento).

Esempio 2. *Vediamo un esempio esplicativo dove l'evento e è l'unico abilitato, ovvero le sue precondizioni sono vere e le sue postcondizioni sono false. Lo scatto di e rende le precondizioni false e le postcondizioni vere, mentre le altre condizioni rimangono inalterate:*



Si nota quindi che lo scatto dell'evento e riguarda solo le precondizioni e le postcondizioni di quel dato evento, come ci ricorda il principio di estensionalità

Definizione 5. *Sia $N = (B, E, F)$ una rete elementare. Possiamo definire due tipologie di rete:*

1. *N è definita **semplice** sse:*

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

Ovvero per ogni coppia di elementi (che siano quindi eventi o condizioni) se i loro pre-elementi e i loro post-elementi coincidono allora non ha senso distinguere x e y.

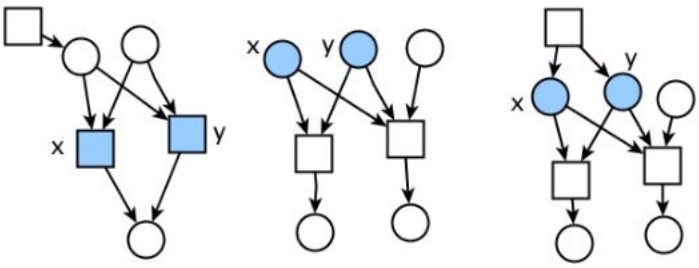


Figura 2.6: Esempi di reti **non** semplici

2. *N è definita **pura** sse:*

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

*Ovvero se per ogni evento non esiste una precondizione che sia anche postcondizioni. Si ha quindi un **cappio** (detto anche **side condition**) tra un evento e una condizione. Avere questa situazione comporta che l'evento non può scattare in quanto la condizione che per lui è sia una precondizione che una postcondizioni non può essere contemporaneamente vera e falsa, l'evento non potrà mai scattare e quindi non potrà mai essere osservato. Non avrebbe quindi senso modellarlo*

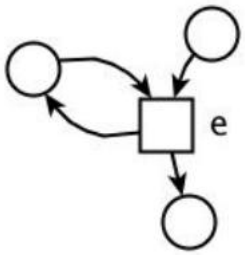


Figura 2.7: Esempio di rete **non** pura

Definizione 6. *Data una rete elementare $N = (B, E, F)$ e sia $U \subseteq E$ un sottoinsieme di eventi e siano $c, c_1, c_2 \in B$ tre configurazioni. Si ha che:*

- *U è un **insieme di eventi indipendenti** sse:*

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

ovvero per ogni coppia distinta di eventi nell'insieme U si ha che le precondizioni e le postcondizioni dei due eventi sono completamente disgiunte.

- *U è un **passo abilitato**, ovvero un insieme di eventi concorrenti in una certa configurazione c, che si indica con:*

$$c[U >$$

sse:

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

U quindi deve essere un insieme di eventi indipendenti e ogni evento in U è abilitato in c, quindi le sue precondizioni sono vere e le sue postcondizioni sono false. Si ha quindi che U è un insieme di eventi abitati in maniera concorrente in c

- *U è un **passo** dalla configurazione c_1 alla configurazione c_2, che si indica con:*

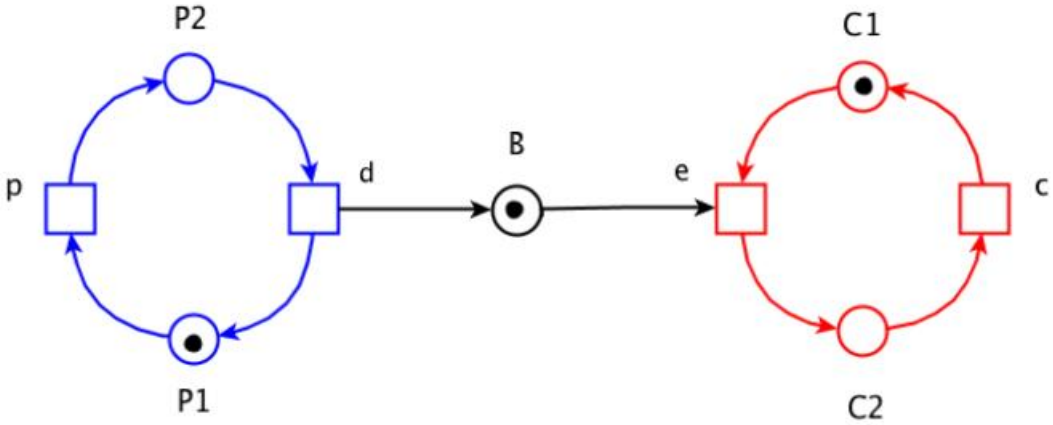
$$c_1[U > c_2$$

sse:

$$(c_1[U) \wedge \left(c_2 = (c_1 -^\bullet U) \cup U^\bullet\right)$$

ovvero sse U è un passo abilitato in c_1 e lo scatto degli eventi in U porta alla configurazione c_2 che si ottiene togliendo da c_1 l'insieme delle precondizioni degli eventi in U e aggiungendo quindi l'insieme delle postcondizioni degli eventi in U

Esempio 3. *Riprendiamo l'esempio del produttore e del consumatore. Sia dato il sistema Σ che modella produttore e consumatore*



Si hanno:

- *$\{p, e\}, \{p, c\}, \{d, c\}$ esempi di insiemi di eventi indipendenti*
- *$\{p, e\}$ che è un passo abilitato in $\{P_1, B, C_1\}$*
- *$\{P_1, B, C_1\}[\{p, w\} > \{P_2, C_2\}$ ovvero lo scatto del passo $\{p, e\}$ ci porta in $\{P_2, C_2\}$*

Diamo ora una definizione formale di **sistema elementare**.

Definizione 7. *Un **sistema elementare** $\Sigma = (B, E, F; c_{in})$ è definito come una rete $N = (B, E, F)$ e a cui è associato un caso iniziale, una configurazione iniziale, ovvero un sottoinsieme di condizioni che rappresentano lo stato iniziale da cui inizia la computazione e l'evoluzione del sistema. Formalmente il caso iniziale si indica con $C_{in} \in B$*

Definizione 8. *Dato un sistema elementare $\Sigma = (B, E, F; c_{in})$ si indica con C_Σ l'insieme dei **casi raggiungibili** da tale sistema a partire dal caso iniziale c_{in}.*
Formalmente l'insieme dei casi raggiungibili è il di piccolo sottoinsieme dell'insieme delle parti di B, ovvero 2^B, tale che:

- *$c_{in} \in C_\Sigma$, ovvero sicuramente il caso iniziale appartiene all'insieme dei casi raggiungibili*
- *se $c \in C_\Sigma$, $U \subseteq E$ e $c' \subseteq B$ sono tali che $c[U > c'$ allora $c' \in C_\Sigma$, ovvero se ho un generico caso c che appartiene ai casi raggiungibili, se ho un insieme di eventi U tale che questo insieme di eventi (che abbiamo visto essere indipendenti, per la definizione di passo abilitato) è abilitato in c in un unico passo e la sua occorrenza mi porta in c', allora anche c' appartiene a C_Σ.*

*Questa è una definizione data per **induzione strutturale**, nel primo punto si ha la base, nel secondo l'ipotesi e la conseguenza*

Definizione 9. *Dato un sistema elementare $\Sigma = (B, E, F; c_{in})$ si indica con U_Σ l'**insieme dei passi** di Σ, ovvero di tutti i possibili insiemi di eventi indipendenti che possono occorrere in qualche caso. Formalmente:*

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

Ovvero l'insieme dei sottoinsiemi di eventi tali per cui esistano due casi raggiungibili in C_Σ e U è abilitato in c e il suo scatto mi porta in c'.

Definiamo ora il comportamento dei sistemi elementari.

Definizione 10. *Sia $\Sigma = (B, E, F; c_{in})$ un sistema elementare e siano $c_i \in C_\Sigma$ ed $e_i \in E$.*
Definiamo;

- *un **comportamento sequenziale** come una sequenza di eventi che possono occorrere dal caso iniziale. Facendo scattare in maniera sequenziale gli eventi uno alla volta in c_n:*

$$c_{in}[e_1 > c_1[e_2 > \ldots [e_n > c_n$$

Scrittura che può essere alleggerita in:

$$c_{in}[e_1 e_2 \dots e_n > c_n$$

Possiamo dire di avere a che fare con una ***simulazione sequenziale non deterministica, detta anche semantica a interleaving****, infatti ho più eventi abilitati da prendere uno alla volta*

- *un* ***comportamento non sequenziale****, in quanto possiamo anche considerare insiemi di eventi, ovvero passi. Considero quindi sequenze di passi, avendo a che fare con la* ***step semantics****. Non ho quindi una simulazione sequenziale non deterministica in quanto dal caso iniziale faccio scattare un insieme di eventi, in maniera concorrente (e quindi senza ordine specificato), per poi far scattare un altro insieme di eventi fino ad arrivare a* c_n*:*

  $$c_{in}[U_1 > c_1[U_2 > \dots [U_n > c_n$$

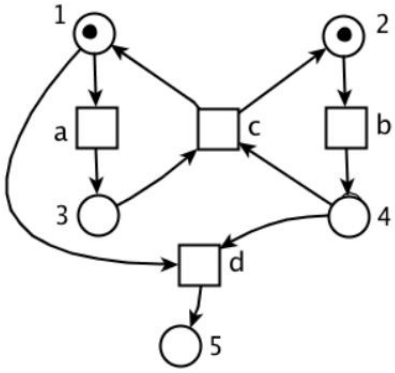
  Scrittura che può essere alleggerita in:

  $$c_{in}[U_1 U_2 \dots U_n > c_n$$

  Gli insiemi U_i *non sono insiemi massimali abilitati ma sottoinsiemi indipendenti e abilitati in* c_{in}*.*
  Posso avere anche un altro tipo di ***comportamento non sequenziale****, definito da Petri stesso, in una* ***semantica ad ordini parziali*** *in cui si definiscono processi non sequenziali. Il comportamento di tale sistema viene registrato in una rete di Petri*

In ogni caso si considerano sia sequenze finite che infinite (con cicli) di eventi o passi.

Esempio 4. *Dato il sistema elementare* Σ*:*



si ha, per esempio, la seguente sequenza di occorrenza di eventi:

$$\{1,2\}[a>\{3,2\}[b>\{3,4\}[c>\{1,2\}[b>\{1,4\}[d>\{5\}$$

arrivati in "5" abbiamo un caso finale, ovvero una situazione di ***deadlock****, in quanto il sistema non può evolvere ulteriormente.*
Vediamo anche la seguente possibile sequenza di passi. In "1" e "2" sia "a" che "b" sono indipendenti e sono entrambi abilitati (scattano in maniera concorrente in un unico passo... ovviamente posso avere passi con lo scatto di un solo evento):

$$\{1,2\}[\{a,b\}>\{3,4\}[\{c\}>\{1,2\}[\{b\}>\{1,4\}$$

Come ricordato posso finire in una sequenza infinita.

Vediamo ora come modellare e registrare il comportamento del sistema. Un modo è usando il **grafo dei casi raggiungibili**.

Definizione 11. *Il* ***grafo dei casi raggiungibili*** *di un sistema elementare* $\Sigma = (B, E, F; c_{in})$ *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, ovvero gli stati globali sono i casi raggiungibili dal sistema* Σ
- U_Σ*, è l'alfabeto, ovvero i passi del sistema rappresentano l'alfabeto*

- *A è l'insieme di archi etichettati, formalmente definito come:*

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

ovvero sono archi che connettono uno caso c con un caso c′ e sono etichettati con un passo U sse U è abilitato in c e porta in c′. Ovviamente c e c′ sono devono essere raggiungibili e U deve appartenere all'insieme dei passi di Σ

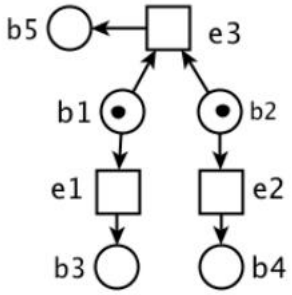


Figura 2.8: il sistema Σ

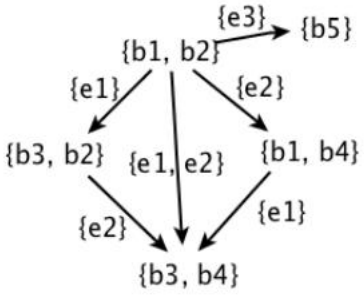


Figura 2.9: Grafo dei casi del sistema Σ

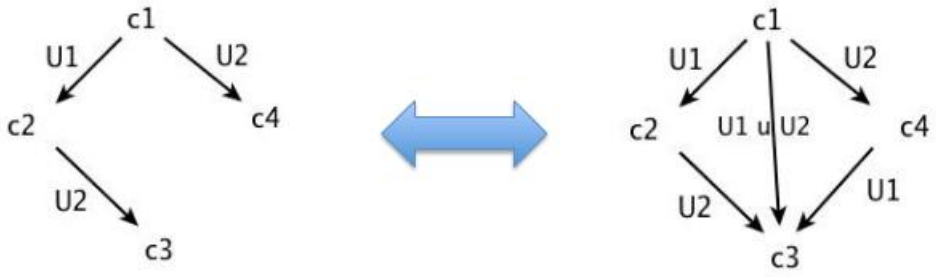
2.1.1 Diamond Property

Dato un sistema elementare $\Sigma = (B, E, F; c_{in})$ e il suo grafo dei casi $CG_\Sigma = (C_\Sigma, U_\Sigma, A, c_{in})$ si ha che il grafo soddisfa una particolare proprietà, detta **diamond property**, tipica solo dei sistemi elementari.

Definizione 12. *La **diamond property** stabilisce una proprietà della struttura del grafo della rete elementare, ovvero, dati $U_1, U_2 \in U_\Sigma$ tali che:*

- $U_1 \cap U_2 = \emptyset$
- $U_1 \neq \emptyset$
- $U_2 \neq \emptyset$

e dati $c_i \in C_\Sigma$ allora vale, per esempio:



ovvero se posso rilevare come sottografo una struttura come quella a sinistra nell'immagine allora sicuramente tale sottografo contiene anche l'arco gli archi per ottenere l'immagine di destra. Il discorso vale anche all'opposto.

Si possono fare delle prove:

1. **prima prova:**
   Dimostriamo che possiamo passare all'immagine di destra da quella di sinistra aggiungendo i due archi mancanti.
   Per semplicità diciamo che U_i è un singolo evento e_i, con $i = 1, 2$. Siano inoltre $c_1, c_2 \in C_\Sigma$, ovvero sono casi raggiungibili, ed $e_1, e_2 \in E$ tali che $c_1[e_1 > c_2[e_2 >$ e $c_1[e_2 >$, i due eventi quindi sono abilitati in sequenza e da c_1 è anche abilitato c_2 . Si vuole dimostrare che:

   $$({}^\bullet e_1 \cup e_1^\bullet) \cap ({}^\bullet e_2 \cup e_2^\bullet) = \emptyset$$

   ovvero che i due eventi sono indipendenti, che sono entrambi abilitati e che sono eseguibili in qualsiasi ordine.

Da $c_1[e_1 >$ e $c_1[e_2 >$ segue che:

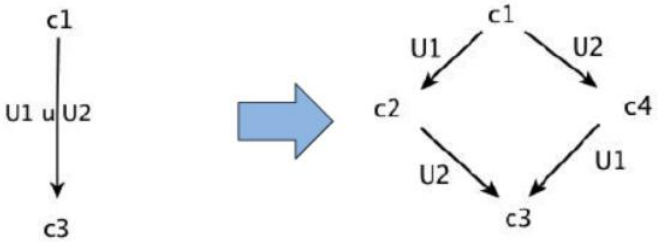
- ${}^\bullet e_1 \cap e_2^\bullet = \emptyset$
- ${}^\bullet e_2 \cap e_1^\bullet = \emptyset$

infatti se e_1 e e_2 sono entrambi abilitati in c_1, le loro pre-condizioni sono vere e le post-condizioni false, e quindi non è possibile che una condizione sia contemporaneamente precondizione di e_1 (vera) e anche postcondizione di e_2 (falsa), e viceversa. Quindi le precondizioni di un evento sono disgiunte dalle postcondizioni dell'altro. Inoltre dal fatto che ho $c_1[e_1 > c_2[e_2$, ovvero che da c_1 è abilitato e_1 e che dopo lo scatto di e_1 è ancora abilitato e_2 possiamo dire che:

- $e_1^\bullet \cap e_2^\bullet = \emptyset$
- ${}^\bullet e_1 \cap {}^\bullet e_2 = \emptyset$

in c_2, infatti, le pre-condizioni 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 postcondizioni 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. Quindi le precondizioni dei due eventi sono disgiunte, come del resto anche le postcondizioni, in quanto i due eventi sono sequenziali.
Si è quindi dimostrato che i due eventi hanno precondizioni e post-condizioni completamente disgiunte e quindi la tesi è verificata

2. **seconda prova:**
   Analizzando la situazione:



Si supponga che $U_1 \cup U_2 \in U_\Sigma$ e che si abbiano:

- $U_1 \cap U_2 = \emptyset$, ovvero sono disgiunti
- $U_1 \neq \emptyset$
- $U_2 \neq \emptyset$

allora se $c_1[(U_1 \cup U_2) > c_3$, quindi è abilitato il passo $U_1 \cup U_2$ in c_1, sicuramente si ha che sono abilitati anche i singoli passi:

- $c_1[U_1 >$
- $c_1[U_2 >$

resta da dimostrare che dopo lo scatto di U_1 è ancora abilitato U_2 in c_2. Ma se $U_1 \cup U_2$ è un passo abilitato significa che posso eseguirli in qualsiasi ordine, quindi anche prima U_1 e poi U_2, e questo comporta sicuramente che U_2 è abilitato e che porta a c_3. Analogamente invertendo U_1 e U_2, formalmente:

- $c_1[U_1 > c_2[U_2 > c_3$
- $c_1[U_2 > c_4[U_1 > c_3$

Si dimostra così che l'immagine di sinistra comporta quella di destra.

Grazie alla diamond property possiamo non considerare il grafo dei casi raggiungibili ma solo il **grafo dei casi sequenziale**:

Definizione 13. *Un **grafo dei casi sequenziale** del sistema elementare $\Sigma = (B, E, F; c_{in})$ è una quadrupla:*

$$SCG_\Sigma = (C_\Sigma, E, A, c_{in})$$

dove le etichette sono i singoli eventi (mentre il resto rimane definito come nel grafo dei casi raggiungibili). Formalmente si ha quindi che:

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

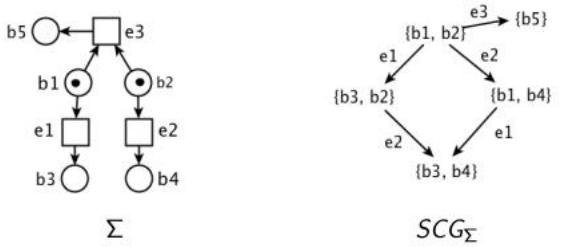


Figura 2.10: Esempio di grafo dei casi sequenziale

Si registra quindi l'occorrenza di un evento alla volta. Il grafo dei casi sequenziale è quindi il sistema di transizione con gli archi etichettati dai singoli eventi.

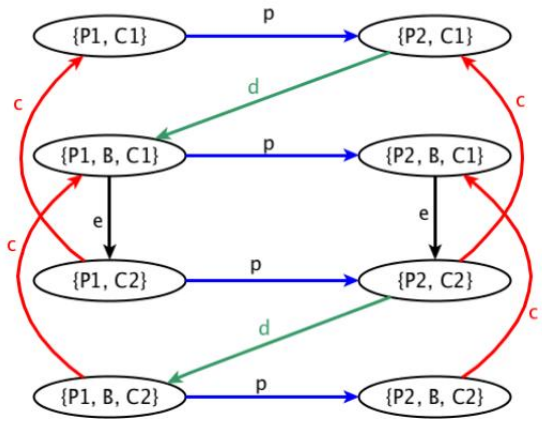


Figura 2.11: Esempio di grafo dei casi sequenziale dell'esempio c0on produttore e consumatore

Riprendendo l'immagine precedente si ha che per la diamond property possono aggiungere l'arco "centrale" che trasformerebbe nuovamente il grafo dei casi sequenziale in quello dei casi raggiungibili quindi: *per la diamond property, nei sistemi elementari il grafo dei casi e il grafo dei casi sequenziale sono* ***sintatticamente equivalenti****, ovvero possono essere ricavati a vicenda.*
Questo implica il fatto che due sistemi elementari hanno grafi dei casi ***isomorfi*** *sse hanno grafi dei casi sequenziali isomorfi.*

2.1.2 Isomorfismo tra Sistemi di Transizione Etichettati

Si ricorda che:

Si parla di isomorfismo quando due strutture complesse si possono applicare l'una sull'altra, cioè far corrispondere l'una all'altra, in modo tale che per ogni parte di una delle strutture ci sia una parte corrispondente nell'altra struttura; in questo contesto diciamo che due parti sono corrispondenti se hanno un ruolo simile nelle rispettive strutture.

Diamo ora una definizione formale di isomorfismo tra sistemi di transizione etichettati, che possono quindi essere grafi dei casi o grafi dei casi sequenziali.

Definizione 14. *Siano dati due sistemi di transizione etichettati:*
$A_1 = (S_1, E_1, T_1, s_{01})$ *e* $A_2 = (S_2, E_2, T_2, s_{02})$.
e siano date due ***mappe biunivoche****:*

1. $\alpha : S_1 \to S_2$, *ovvero che passa dagli stati del primo sistema a quelli del secondo*
2. $\beta : E_1 \to E_2$, *ovvero che passa dagli eventi del primo sistema a quelli del secondo*

allora:

$$\langle \alpha, \beta \rangle : A_1 = (S_1, E_1, T_1, s_{01}) \to A_2 = (S_2, E_2, T_2, s_{02})$$

è un ***isomorfismo*** *sse:*

- $\alpha(s_{01}) = s_{02}$, *ovvero l'immagine dello stato iniziale del primo sistema coincide con lo stato iniziale del secondo*
- $\forall s, s' \in S_1, \forall e \in E_1 : (s, e, s') \in T_1 \Leftrightarrow (\alpha(s), \beta(e), \alpha(s')) \in T_2$ *ovvero per ogni coppia di stati del primo sistema, tra cui esiste un arco etichettato e, vale che esiste un arco, etichettato con l'immagine di e, nel secondo sistema che va dall'immagine del primo stato considerato del primo sistema all'immagine del secondo stato considerato del secondo sistema, e viceversa*

Definizione 15. *Si definiscono due* ***sistemi equivalenti*** *sse hanno grafi dei casi sequenziali, e quindi di conseguenza anche grafi dei casi,* isomorfi.
Due sistemi equivalenti accettano ed eseguono le stesse sequenze di eventi

2.1.3 Il Problema della Sintesi

Si presenta ora un problema tipico dell'informatica, il **problema della sintesi**, ovvero dato un comportamento, o meglio una sua specifica, decidere se esiste un'implementazione di tale specifica, ovvero un modello, che abbia esattamente quel comportamento.
In questo caso dato un sistema di transizioni etichettato $A = (S, E, T, s_0)$, con:

- S insieme degli stati
- E insieme delle etichette, ovvero degli eventi
- T insieme delle transizioni
- s_0 stato iniziale

ci si propone di stabilire se esiste un sistema elementare $\Sigma = (B, E, F; c_{in})$, tale che l'insieme degli eventi del sistema esattamente l'insieme delle etichette di A e tale che il suo grafo dei casi SCG_Σ sia isomorfo ad A. Ci si propone anche di costruirlo.
Il problema è stato risolto mediante la cosiddetta **teoria delle regioni** (che però non verrà trattato nel corso). Si può però dire che A dovrà soddisfare la diamond property, in quanto altrimenti non sarebbe un sistema di transizioni che potrebbe corrispondere al comportamento di un sistema elementare.

2.1.4 Contatti

Definizione 16. *Sia $\Sigma = (B, E, F; c_{in})$ un sistema elementare e siano $e \in E$ un evento e $c \in C_\Sigma$ un caso raggiungibile dal caso iniziale. Allora si ha che (e, c) è un **contatto** sse:*

$$^\bullet e \subseteq c \wedge e^\bullet \cap c \neq \emptyset$$

Ovvero, in termini pratici, siamo nel caso in cui un evento e ha le precondizioni vere, si ha quindi che $^\bullet e \subseteq c$, e l'evento non ha tutte le postcondizioni false, quindi $e^\bullet \cap c \neq \emptyset$, allora si dice che l'evento e è in una situazione di contatto e quindi non può scattare

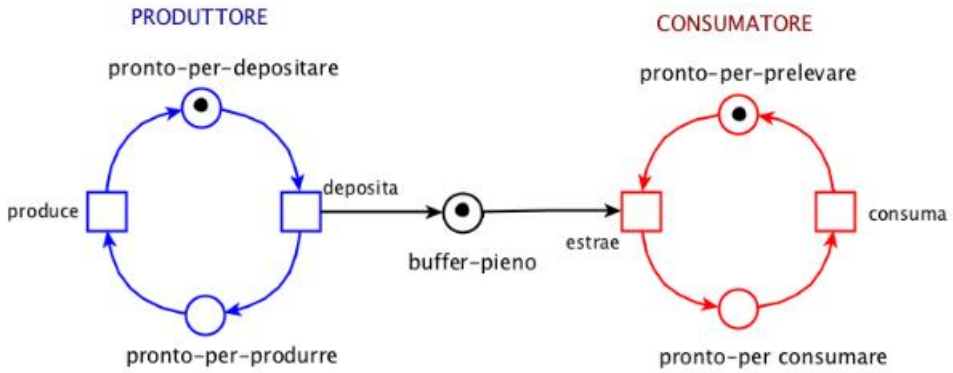


Figura 2.12: Esempio dove l'evento *deposita* è in una situazione di contatto

Definizione 17. *Sia $\Sigma = (B, E, F; c_{in})$ un sistema elementare. Si dice che il sistema è **senza contatti** sse:*

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

ovvero per ogni evento e per ogni caso raggiungibile dal caso iniziale succede sempre che se le precondizioni sono vere, ovvero ${}^\bullet e \subseteq c$*, allora le postcondizioni sono false, ovvero disgiunte dal caso considerato* ($e^\bullet \cap c = \emptyset$)

Ci si chiede se sia possibile trasformare un sistema elementare Σ, con contatti, in uno Σ', senza contatti, senza però modificarne il comportamento.
La risposta a questo quesito è affermativa e la procedura consiste nell'aggiungere a Σ il complemento di ogni condizione che crea situazione di contatto, ottenendo così un sistema Σ' con grafo dei casi isomorfo a quello di Σ.
Per aggiungere il complemento, data la condizione x, si aggiunge la condizione *not* x che sarà vera tutte le volte che x è falsa e viceversa. Per ottenere questo risultato la nuova condizione avrà come pre-eventi i post-eventi di x e come post-eventi i pre-eventi di x. Ovvero connetto la nuova condizione agli stessi eventi di quella vecchia ma con archi orientati in senso opposto. Ovviamente le inizializzazioni delle due condizioni dovranno essere opposte (una vera e l'altra falsa).

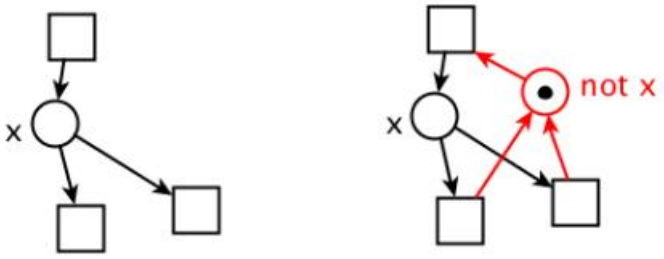


Figura 2.13: Esempio con l'ottenimento del complemento di un sistema

Se un sistema è senza contatti si ha una regola di contatto semplificata:

Definizione 18. *Sia* $\Sigma = (B, E, F; c_{in})$ *un sistema elementare* ***senza contatti****. Sapendo che se le precondizioni di un evento sono vere allora sicuramente le postcondizioni di quell'evento sono false in quel caso. Dato che questo avviene per ogni evento e per ogni caso raggiungibile dal caso iniziale per verificare che un evento e sia abilitato in un caso raggiungibile c è sufficiente verificare che le precondizioni di e siano vere (in quanto automaticamente le postcondizioni saranno false). In maniera formale quindi si ha che:*

$$c[e \text{ sse } {}^\bullet e \subseteq c, \quad \text{con } e \in E, c \in C_\Sigma$$

semplificando di molto la ***regola di scatto***

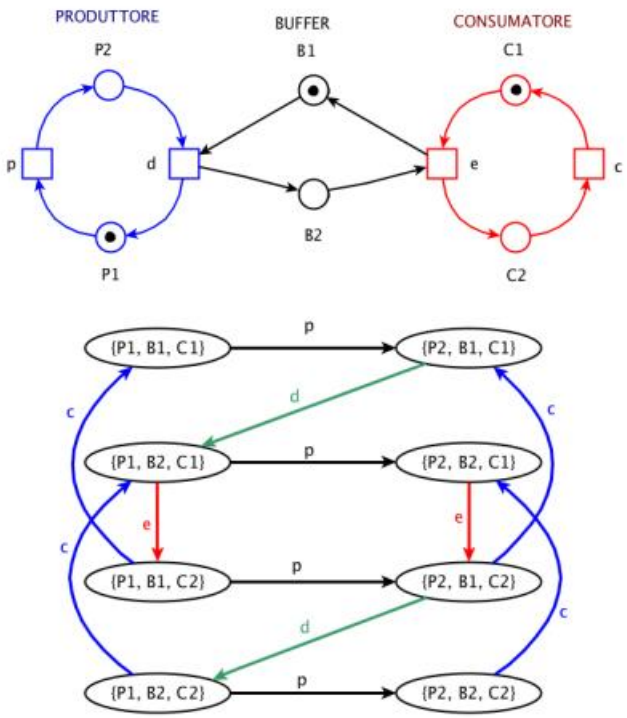


Figura 2.14: Esempio con il complemento del sistema produttore-consumatore (dove è stato aggiunto solo il complemento di "buffer-pieno", ottenendo così sia $B1$ che $B2$, in quanto le altre condizioni avevano già il loro complemento). In aggiunta si ha anche il grafo dei casi sequenziale corrispondente al nuovo sistema senza contatti (grafo che è isomorfo a quello ottenibile al sistema con contatti)

2.1.5 Situazioni Fondamentali

Sequenza

Definizione 19. *Sia $\Sigma = (B, E, F; c_{in})$ un sistema elementare, con contatti o meno e siano $c \in C_\Sigma$ un caso raggiungibile dal caso iniziale e $e_1, e_2 \in E$ due eventi.*
Si ha che e_1 ed e_2 sono ***in sequenza*** *nel caso raggiungibile c sse:*

$$c[e_1 > \wedge \neg c[e_2 \wedge c[e_1 e_2 >$$

ovvero in c è abilitato e_1 ma non e_2 ma, dopo lo scatto di e_1, e_2 diventa abilitato. Quindi in c è possibile attivare prima e_1 e poi e_2 in sequenza.
Si ha quindi una relazione di ***dipendenza causale tra*** *e_1* ***ed*** *e_2, ovvero*

qualche postcondizione di e_1 è precondizione di e_2 (che quindi può occorrere solo se precedentemente è occorso e_1).

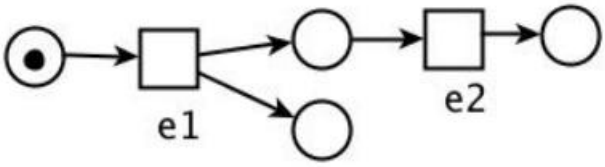


Figura 2.15: Esempio di sequenza tra e_1 ed e_2

Concorrenza

Definizione 20. *Sia $\Sigma = (B, E, F; c_{in})$ un sistema elementare, con contatti o meno e siano $c \in C_\Sigma$ un caso raggiungibile dal caso iniziale e $e_1, e_2 \in E$ due eventi.*
*Si ha che i due eventi sono **concorrenti** nel caso raggiungibile c sse:*

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

ovvero se possono essere abilitati in unico passo o, detto in maniera diversa, se sono indipendenti ed entrambi abilitati in c

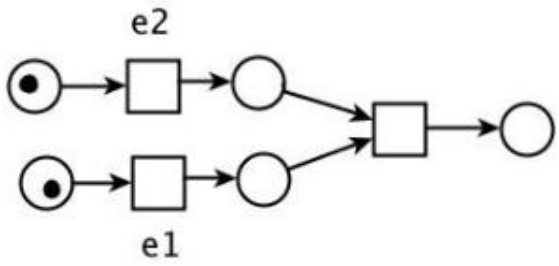


Figura 2.16: Esempio di concorrenza tra e_1 ed e_2

Conflitto

Definizione 21. *Sia $\Sigma = (B, E, F; c_{in})$ un sistema elementare, con contatti o meno e siano $c \in C_\Sigma$ un caso raggiungibile dal caso iniziale e $e_1, e_2 \in E$ due eventi.*
Si ha che e_1 ed e_2 sono in conflitto sse:

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

ovvero i due eventi sono entrambi abilitati (quindi le precondizioni sono vere mentre le postcondizioni son false) ma l'occorrenza di uno disabilità l'altro, quindi non possono essere abilitati in un unico passo, in quanto non sono indipendenti. Ci sono due casi:

1. *i due eventi hanno una precondizione in comune, e in tal caso si parla di* ***conflitto forward*** *(ovvero in avanti)*

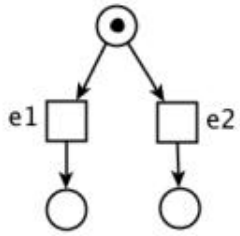


Figura 2.17: Esempio di conflitto forward tra e_1 ed e_2

2. *i due eventi hanno una postcondizione in comune, e in tal caso si parla di* ***conflitto backward*** *(ovvero all'indietro)*

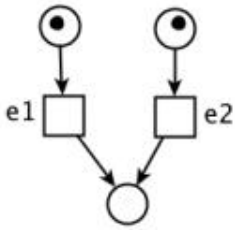


Figura 2.18: Esempio di conflitto backward tra e_1 ed e_2

Si ha quindi una situazione di ***non determinismo****, non essendo specificato quale dei due eventi scatterà prima (e lo scatto di uno impedisce lo scatto dell'altro).*

Posso ritrovarmi nel caso in cui effettivamente un evento scatta, cambiando lo stato del sistema. In tal caso, in un'ottica completamente deterministica, si deve assumere che ***l'ambiente*** *abbia fornito un'informazione riguardo il conflitto, ovvero c'è stato qualcosa di esterno che ha permesso ad uno dei due eventi di scattare ugualmente. Ho quindi guadagnato dell'informazione.*

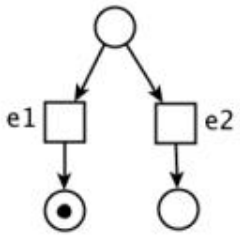


Figura 2.19: Esempio di conflitto con l'intervento dell'ambiente tra e_1 ed e_2 (con conseguente guadagno di informazione)

Ci sono però casi in cui in ogni caso non si può avere informazione su quale esempio sia scattato. Si può ipotizzare che tale informazione fosse presente nello stato precedente del sistema (una sola condizione attiva, per esempio). Quindi l'informazione, finita nell'ambiente (ricevuta dall'ambiente), si è persa.

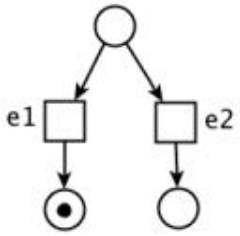


Figura 2.20: Esempio di conflitto tra e_1 ed e_2 con conseguente perdita di informazione (si può ipotizzare, per esempio, che nello stato precedente la precondizione di e_1 fosse attiva mentre quella dio e_2 fosse inattiva)

Il modello, nell'ottica di Petri, non è quindi un ***modello chiuso*** *ma è in grado di comunicare con l'ambiente (in sintonia con le teorie della fisica).*

Confusione

Definizione 22. *La situazione di* ***confusione*** *è una mistura di situazioni di concorrenza e di conflitto. Si hanno 2 tipi di confusione, entrambe ammissibili:*

1. *detta* ***confusione asimmetrica*** *considera il fatto di avere un caso raggiungibile e due eventi abilitati, nella figura e_1 ed e_2, in maniera concorrente in c, che nella figura consiste nel caso $\{b_1, b_2, b_3\}$. I due eventi sono quindi indipendenti. Nella figura lo scatto dei due eventi porterebbe allo stato $c' = \{b_4, b_5\}$. Bisogna analizzare però nel dettaglio il sistema. Se prima occorre e_1 non si ha alcun conflitto mentre se occorre prima e_2 (che porterebbe in $\{b_1, b_3, b_4\}$) si crea un conflitto tra e_1 ed e_3, che viene risolto a favore di e_1 e a sfavore di e_3.* ***Non è possibile stabilire oggettivamente se è stato sciolto un conflitto****. Sono quindi in una situazione di confusione in quanto non so se è stata effettuata o meno una scelta.*

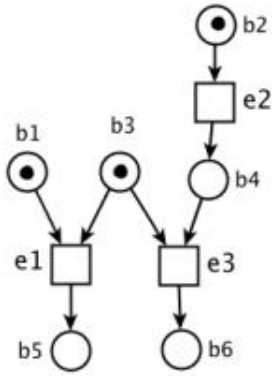


Figura 2.21: Esempio di confusione asimmetrica tra e_1 ed e_2

Una soluzione che vedremo sarà la scomposizione in più componenti del sistema

2. *detta* ***confusione simmetrica****, nome dovuto al fatto che la rete risulta disegnata in modo simmetrico, comportando delle problematiche. Prendiamo nell'immagine il caso raggiungibile $c = \{b_1, b_2\}$ e i due eventi e_1 ed e_3, abilitati in maniera concorrente. Si ha che $c[\{e_1, e_3\} > c'$, con $c' = \{b_3, b_5\}$. Anche in questo caso si hanno dei conflitti, infatti sia e_1 che e_3 sono in conflitto con e_2*

(in quanto se uno dei due viene eseguito e_2 non può più occorrere). Anche in questo caso non posso stabilire se il conflitto è stato risolto nel momento in cui arrivo in c' (ovvero se si è deciso di fare e_1 piuttosto che e_2 o e_3 piuttosto che e_2).
Anche in questo caso potremo dividere in componenti (una che esegue e_1 ed e_2 e un'altra che esegue e_1 ed e_3, con e_2 che è una sincronizzazione tra le due componenti).
Non si può dire chi ha deciso e chi ha la responsabilità di decidere quale evento deve occorrere, se alla componente di b_1 o a quella di b_2.

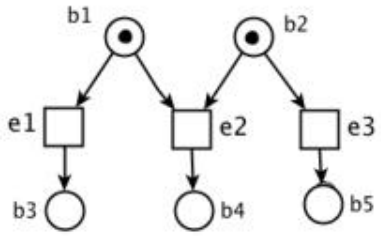
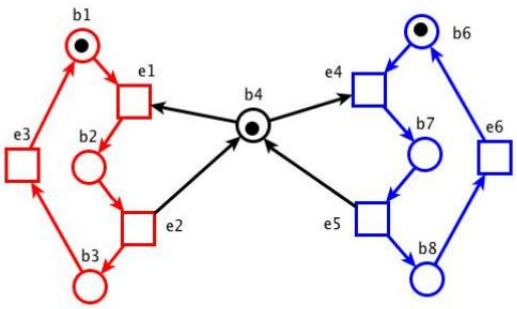


Figura 2.22: Esempio di confusione simmetrica tra e_1 ed e_2

Petri era convinto, nella sua visione deterministica e senza conflitti, che l'avere una situazione di confusione nel modello fosse dovuto al non aver esplicitato alcuni aspetti o di aver costruito male il modello, con informazioni parziali e non complete sul modello e sull'ambiente.
Un altro studioso, Einar Smith, molto vicino a Petri ha invece dimostrato come la confusione sia inevitabile

Vediamo un esempio famoso, detto della **mutua esclusione**, portato da Smith per spiegare come la confusione sia inevitabile nella realtà.

Esempio 5. *Si analizza il seguente sistema:*



In rosso e in Blu abbiamo specificate le due componenti del sistema, che condividono una risorsa, ovvero la condizione b_4, che rappresenta che la risorsa è libera e a disposizione. L'evento e_1 e evento e_4 rappresentano eventi di acquisizione della risorsa (rispettivamente per la prima e per la seconda componente) e quindi le loro precondizioni rappresentano la necessità di acquisirla. Tra questi due eventi c'è una ***situazione di conflitto****. Le condizioni b_2 e b_7, ovvero le rispettive postcondizioni dei due eventi, rappresentano che la risorsa è in uso per la rispettiva componente mentre gli eventi e_2 ed e_5 rappresentano il rilascio della risorsa condivisa, sempre per la rispettiva componente, arrivando rispettivamente nella componente b_3 e b_8. Ovviamente la risorsa non può essere contemporaneamente in uso da entrambe le risorse, quindi b_2 e b_7 non possono essere contemporaneamente marcate, ovvero vere, e per questo si parla di mutua esclusione (se una delle due è vera l'altra deve essere necessariamente falsa). D'altro canto gli eventi e_3 ed e_6 possono invece occorrere in modo concorrente senza conflitti.*
Scrivendo formalmente si ha che, nel caso che la risorsa sia stata acquisita dalla componente rossa e successivamente rilasciata:

$$\{b_3, b_4, b_6\}[\{e_3, e_4\} > \{b_1, b_7\}$$

ma se scatta prima e_3 ho il conflitto tra e_1 ed e_4, se scatta prima e_4 non ho conflitti con e_1. Quindi non ho informazioni sulla risoluzione del conflitto.
Capire se è confusione simmetrica

2.1.6 Sottoreti

Partiamo subito con una definizione formale:

Definizione 23. *Siano $N = (B, E, F)$ e $N_1 = (B_1, E_1, F_1)$ due reti elementari.*
Si dice che N_1 è ***sottorete*** *di N sse:*

- *$B_1 \subseteq B$, quindi l'insieme delle condizioni della rete N_1 è sottoinsieme di quello della rete N*
- *$E_1 \subseteq E$, quindi l'insieme degli eventi della rete N_1 è sottoinsieme di quello della rete N*
- *$F_1 = F \cap [(B_1 \times E_1) \cup (E_1 \times B_1)]$, ovvero la relazione di flusso di N_1 è definita come la restrizione della relazione di flusso di N rispetto alle condizioni e B_1 e agli eventi E_1 (tengo quindi solo gli archi di N che connettono eventi e condizioni di N_1)*

Definizione 24. *Siano $N = (B, E, F)$ e $N_1 = (B_1, E_1, F_1)$ due reti elementari.*
*Si dice che N_1 è **sottorete generata da** B_1 di N (ovvero di sottorete generata da un insieme di condizioni) sse:*

- *$B_1 \subseteq B$, quindi l'insieme delle condizioni della rete N_1 è sottoinsieme di quello della rete N*
- *$E_1 = {}^\bullet B_1 \cup B_1^\bullet$, ovvero come eventi si hanno tutti quegli eventi che sono collegati in N alle condizioni incluse nell'insieme di condizioni B_1, prendendo quindi tutti i pre-eventi e i post-eventi delle condizioni dell'insieme B_1*
- *$F_1 = F \cap [(B_1 \times E_1) \cup (E_1 \times B_1)]$, ovvero la relazione di flusso di N_1 è definita come la restrizione della relazione di flusso di N rispetto alle condizioni B_1 e agli eventi E_1*

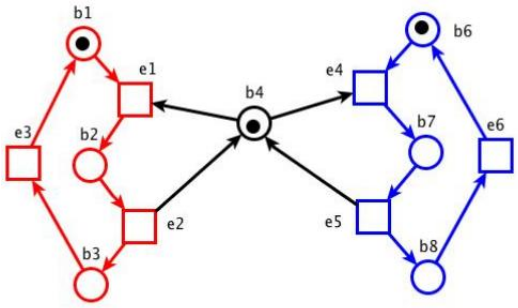
Non ho quindi una sottorete generata da un insieme arbitrario di condizioni ed eventi ma questi ultimi sono direttamente presi in relazione all'insieme delle condizioni scelto

Definizione 25. *Siano $N = (B, E, F)$ e $N_1 = (B_1, E_1, F_1)$ due reti elementari.*
*Si dice che N_1 è **sottorete generata da** E_1 di N (ovvero di sottorete generata da un insieme di condizioni) sse:*

- *$B_1 = {}^\bullet E_1 \cup E_1^\bullet$, ovvero come condizioni si hanno tutte quelle condizioni che sono collegati in N agli eventi inclusi nell'insieme di eventi E_1, prendendo quindi tutte e precondizioni e le postcondizioni degli eventi dell'insieme E_1*
- *$E_1 \subseteq E$, quindi l'insieme degli eventi della rete N_1 è sottoinsieme di quello della rete N*
- *$F_1 = F \cap [(B_1 \times E_1) \cup (E_1 \times B_1)]$, ovvero la relazione di flusso di N_1 è definita come la restrizione della relazione di flusso di N rispetto alle condizioni B_1 e agli eventi E_1*

Non ho quindi una sottorete generata da un insieme arbitrario di condizioni ed eventi ma le prime sono direttamente prese in relazione all'insieme degli eventi scelto

Esempio 6. *Tornando all'esempio della mutua esclusione:*



si ha, per esempio:

- *in rosso si ha la sottorete* $N' = (\{b_1, b_2, b_3\}, \{e_1, e_2, e_3\}, F')$*, che è la sottorete generata dall'insieme di condizioni* $B' = \{b_1, b_2, b_3\}$
- *in blu si ha la sottorete* $N' = (\{b_6, b_7, b_8\}, \{e_4, e_5, e_6\}, F')$*, che è la sottorete generata dall'insieme di condizioni* $B' = \{b_6, b_7, b_8\}$
- *si ha la sottorete* $N' = (\{b_2, b_4, b_7\}, \{e_1, e_2, e_4, e_5\}, F')$*, che è la sottorete generata dall'insieme di condizioni* $B' = \{b_2, b_4, b_7\}$
- *si ha la sottorete* $N' = (\{b_1, b_2, b_3, b_4\}, \{e_1, e_2, e_3\}, F')$*, che è la sottorete generata da dall'insieme di eventi* $E' = \{e_1, e_2, e_3\}$

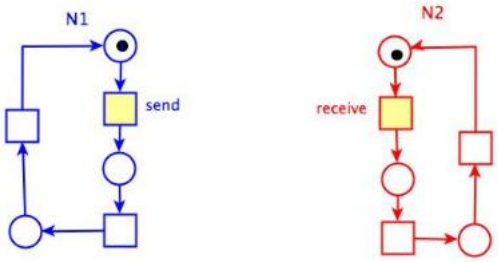
2.1.7 Operazioni di Composizione per Reti di Petri

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:

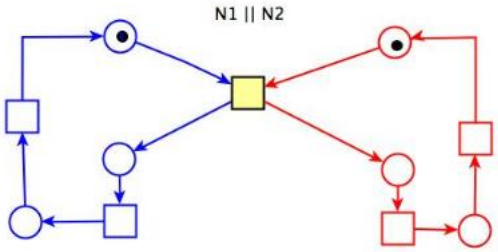
1. la **composizione sincrona**
2. la **composizione asincrona**
3. la **composizione mista, tra sincrona e asincrona**

Iniziamo informalmente a vedere degli esempi pratici.

Esempio 7. *Supponiamo di avere i modelli di due componenti, N_1, con un evento che corrisponde ad un'azione di invio, ed N_2, con un evento che corrisponde ad un'azione di ricezione:*



Supponiamo che invio e ricezione siano eventi corrispondenti all'hand-shacking, ovvero l'invio avviene solo se può avvenire la ricezione.
Vado quindi a sincronizzare questi due eventi, che diventano quindi un'unico evento nella rete composta, che è abilitato se le due precondizioni, nelle due componenti sono entrambe vere:

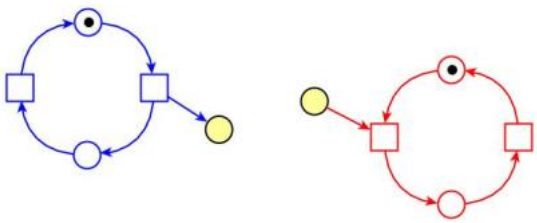


La composizione viene indicata con:

$$N_1||N_2$$

Lo scatto dell'evento, in maniera sincrona, rende vere le postcondizioni nelle due componenti e false le due precondizioni.
Abbiamo appena visto un esempio di ***composizione sincrona***

Esempio 8. *Supponiamo di avere i modelli di due componenti, N_1, che invia in un canale un messaggio (per esempio in un buffer), e N_2, che riceverà il messaggio solo quando esso sarà disponibile:*



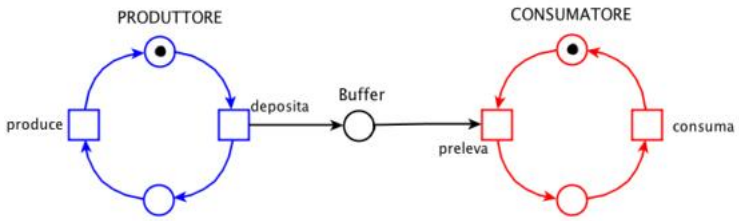
In questo caso, a differenza dell'esempio precedente, non identifichiamo eventi ma condizioni. Identifico quindi il canale (le due condizioni) come uno solo, che avrà il pre-evento in una componente e il post-evento nell'altra. Quindi il pre-evento, nella componente N_1, può scattare solo se questa nuova condizione condivisa, il canale, è libera, indipendentemente dalla componente N_2. D'altro canto l'evento in N_2 può scattare solo se la condizione condivisa è marcata, indipendentemente dallo stato della prima componente, liberando il canale di comunicazione. Avvio e ricezione (dopo che il messaggio è stato inviato può essere letto in un qualsiasi futuro) non sono sincronizzati e si ha quindi a che fare con un esempio di ***composizione asincrona***

Sarà interessante studiare come la composizione di due componenti, per esempio, senza deadlock non comporta, in generale, l'ottenimento di una rete priva di deadlock.

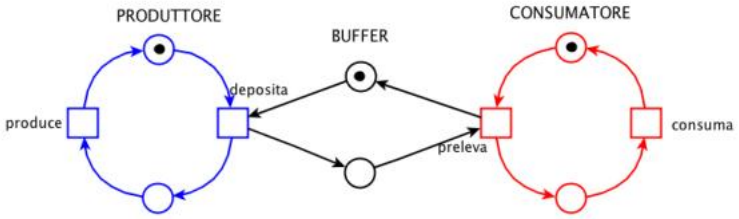
2.2 Reti Posti e Transizioni

Vediamo ora una nuova classe delle *reti di Petri*, detta **reti Posti e Transizioni**, permette di rappresentare un sistema in modo più compatto, sono una sorta di *ripiegamento* dei sistemi elementari. Partiamo quindi da un esempio:

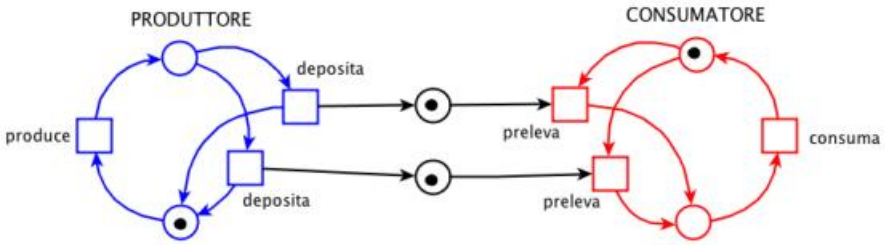
Esempio 9. *prendiamo in studio sempre il sistema produttore-consumatore: Sia nella versione con contatto:*



che senza:

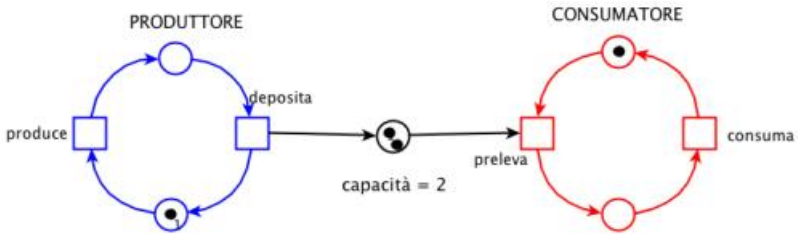


Suppongo di voler modellare il sistema in modo che il buffer possa avere un numero determinato di posizioni, per esempio 2 (il buffer può quindi depositare fino a due elementi). Si ottiene quindi:

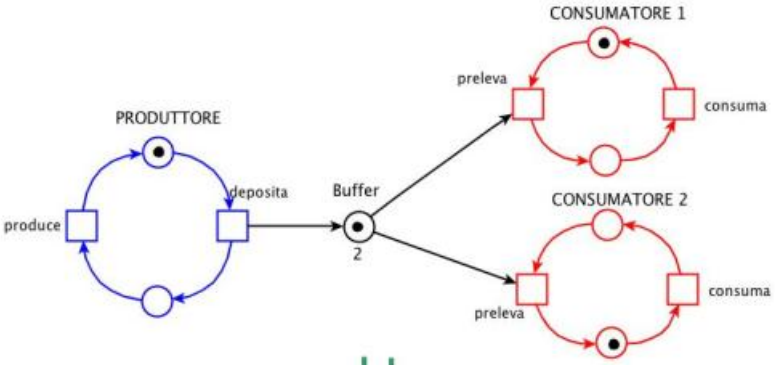


Dove si hanno due posizioni del buffer (e quindi basta che una sia vuota per permettere al produttore di depositare) a disposizione del sistema. Si nota come l'aumento dei buffer complica drasticamente la modellazione del

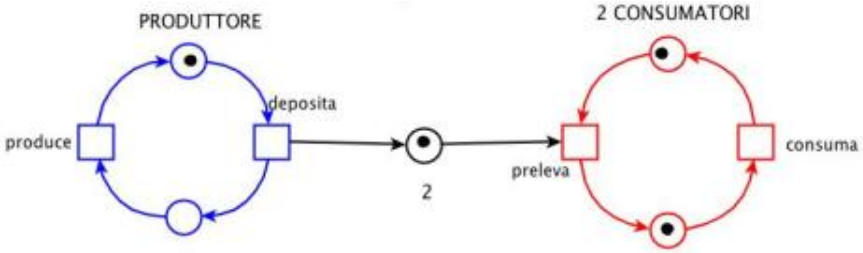
sistema. Si cerca quindi una soluzione più compatta, compattando gli eventi deposita e preleva e dando nuova notazione al buffer. Si ottiene:



con il buffer che diventa anche un ***contatore*** *del numero di elementi presenti al suo interno. Un buffer a due posizioni diventa quindi una condizione non più booleana, detta* ***Posto****, con una* ***capacità*** *pari a 2. Il produttore non può produrre oltre la capacità.*
Con questa rappresentazione non ho più alcuna difficoltà nel rappresentare più posizioni del buffer in quanto basta aumentare la capacità. Ho però perso delle informazioni infatti non ho più una relazione di dipendenza tra dove si deposita (se nella prima posizione o nella seconda) e dove si preleva.
Aggiungiamo un'altra complicanza: aggiungiamo un secondo consumatore:



Possiamo comprimere nella stessa maniera, ottenendo, in quanto i due consumatori hanno lo stesso modello di comportamento:



Anche qui perdo l'informazione riguardo quale dei due consumatori ha effettivamente prelevato, riguardo quale dei due è pronto a prelevare e quale a consumare, continuando ad ignorare anche la posizione del buffer nei confronti della quale stanno agendo.
Si può arrivare ad avere due componenti identiche che sono concorrenti tra loro.

Le condizioni non sono più, in generale, booleane ma sono **posti** dotati di **contatori**, gli elementi all'interno sono detti **marche**. Ai posti si assegna anche una **capacità**.
Si ha inoltre che lo scatto di una transizione dipende dalla disponibilità delle risorse, per esempio un produttore può produrre n elementi alla volta e il consumatore consumarne m. Si usano quindi archi pesati (se non indicato ovviamente ha peso 1, quello del normale check booleano di condizione attiva) tali che una transizione possa scattare sse i pesi vengono rispettati:

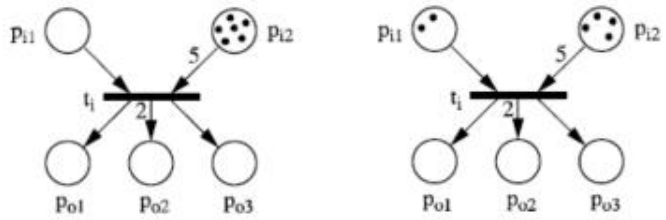


Figura 2.23: Esempi con la transizione t_i non abilitata

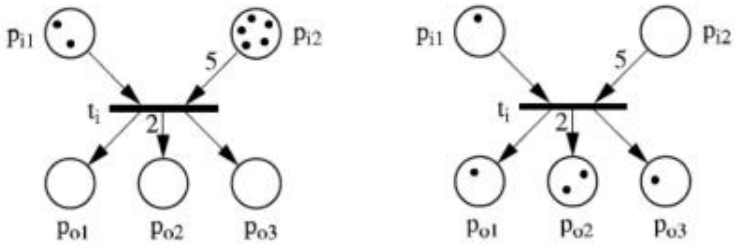


Figura 2.24: Esempio con la transizione t_i abilitata, a sinistra il "prima" e a destra il "dopo" lo scatto di t_i

Esempio 10. *Petri aveva introdotto l'uso delle reti anche per le reazioni chimiche, per esempio:*

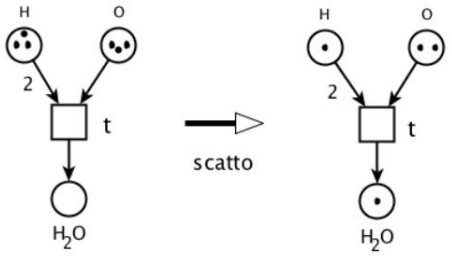


Figura 2.25: Esempio con la modellazione della produzione di H_2O

2.2.1 I Filosofi a Cena

Questo è un classico problema di sincronizzazione, introdotto da Dijkstra. Si ha un tavolo rotondo con 5 filosofi ($p_i, \ = 0, \ldots 4$), ciascuno ha davanti un piatto di spaghetti e si hanno solo 5 forchette ($f_i, \ = 0, \ldots 4$):

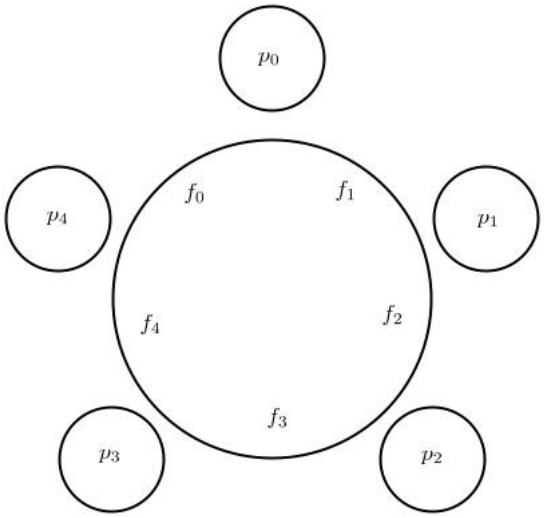
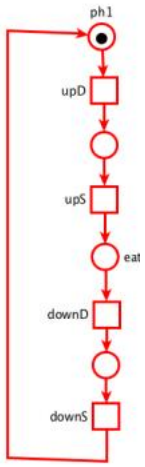


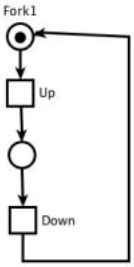
Figura 2.26: Rappresentazione schematica del problema dei 5 filosofi

Ogni filosofo ha lo stesso comportamento, un po' pensa e un po' mangia, prendendo prima la forchetta alla sua destra e poi quella alla sua sinistra (perché necessitano di due forchette per mangiare). Si ha quindi il *deadlock* se tutti vogliono mangiare, prendendo tutti in primis una forchetta, impedendosi tutti a vicenda di mangiare (non potendo prendere due forchette) o

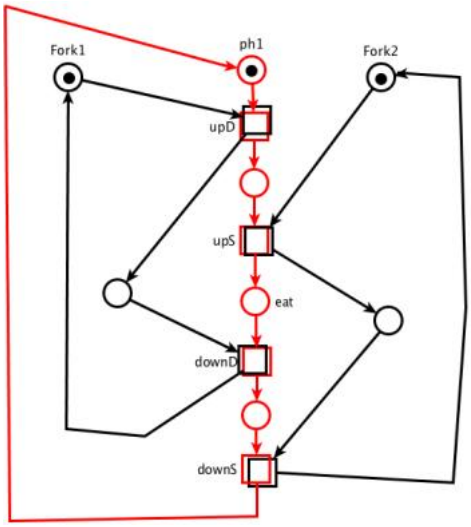
si ha la *starvation* in quanto potrebbero accordarsi per mangiare in quattro, impedendo a uno di mangiare.
Si vuole modellare tale schema. Ogni filosofo può essere modellato come una rete elementare, che farà da componente al modello finale:



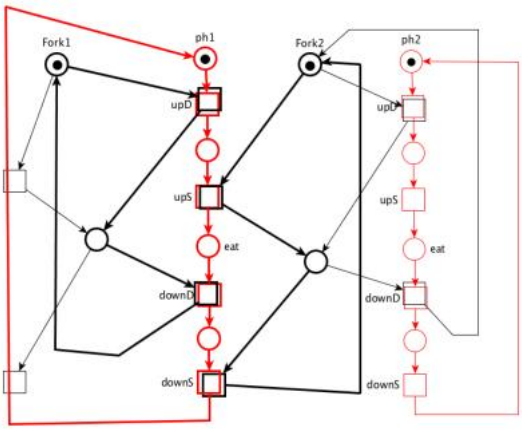
con gli eventi *upD* e *upS*, dove il filosofo prende la forchetta destra e sinistra, e rispettivamente *downD* e *downS*, dove le mette giù. Si ha la condizione che specifica che il filosofo sta pensando e quella che mi segnala l'azione del mangiare, oltre alle due condizioni intermedie che separano le azioni tra forchetta sinistra e destra.
Si modella anche la componente della forchetta, con gli eventi che segnalano se è depositata, *down*, o meno, *up*:



Combino quindi le due componenti per specificare il filosofo che prende la sua forchetta destra e poi la sinistra, quindi per ogni componente *filosofo* si hanno due componenti *forchetta*:

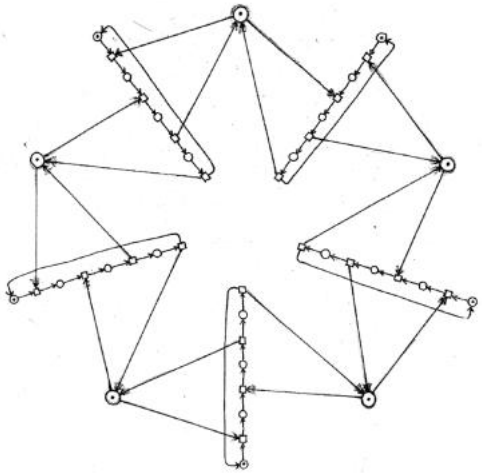


con la *forchetta 1 (fork 1)* che sarà la forchetta destra e la *forchetta 2 (fork 2)* che sarà la forchetta sinistra. Si hanno quindi diverse transizioni di sincronizzazione tra le due componenti (ogni volta che il filosofo interagisce con la forchetta).
Bisogna aggiungere che ogni forchetta può essere presa da due filosofi, ognuna a destra di un filosofo e a sinistra di un altro:

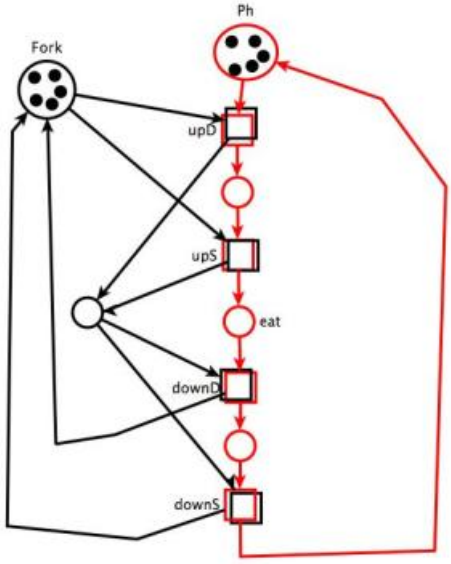


Ogni forchetta quindi si sincronizza alternativamente tra due filosofi, venendo presa da uno dei due filosofi. Si è arrivati quindi ad una **situazione di conflitto**, o meglio ad una **situazione di confusione**.

Per ora la soluzione di questo problema viene lasciato da parte per valutare il modello dello stesso.
Si ha quindi una possibile rappresentazione del modello completo:



Ma ogni filosofo, come del resto ogni forchetta, si comporta nella stessa maniera. Si tenda quindi di *ripiegare* il modello, modellando un un'unica componente filosofo con 5 marche distribuite nei vari stati locali del filosofo. Stesso discorso per le forchette. Ottengo quindi il seguente modello:



Con le due componenti, *filosofo* e *forchetta*, entrambe inizializzate con 5 marche ciascuno. Questo modello però perde informazione sulla forchetta che

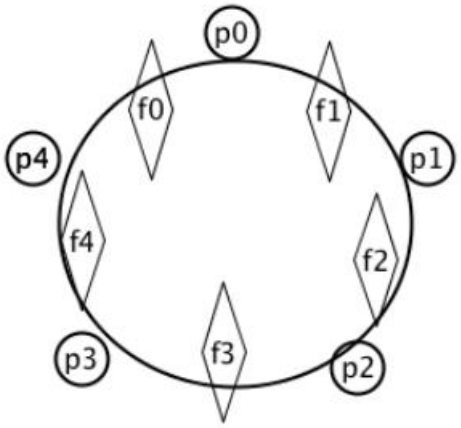
un filosofo può prendere, si ha un radicale **cambio di protocollo**. Un filosofo può prendere una qualsiasi forchetta e non più quella alla sua destra/sinistra. Non posso quindi usare le *reti Posti e Transizioni* in quanto perdo troppe informazioni.
L'unica soluzione possibile è quindi quella di recuperare le informazioni perse inserendole nelle marche, alle quali viene aggiunta una struttura dati. Si ha così modo di distinguere i vari filosofi e le forchette. Per compattare il sistema devo quindi rendere più complessa l'essenza della marca, viene arricchita con una struttura dati.
Si arriva così ad avere una **Rete di Alto Livello**, come, per esempio, una *rete colorata*, a partire da un rete elementare.
Distinguo quindi filosofi e forchette nei due insiemi di strutture dati:

1. $Phil = \{p_0, \ldots, p_4\}$, insieme dei filosofi
2. $Fork = \{f_0, \ldots, f_4\}$, insieme delle forchette

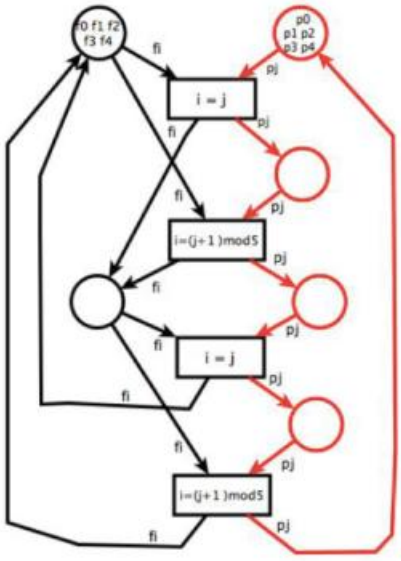
Si nota che per gli indici si ha una somma *modulo 5*, ovvero gli indici rispondono alla regola:

$$(i + 1) \bmod 5$$

In modo che gli indici siano ordinati avendo inoltre lo 0 che segue il 4, in modo circolare.
Si ottiene quindi, mantenendo il protocollo iniziale:



Si mantiene quindi un modello simile a quello descritto sopra ma al posto di marche non strutturate si ha nel posto delle 5 forchette le marche dell'insieme *Fork* e al posto dei 5 filosofi le marche dell'insieme *Phil*:



Le transizioni scattano in determinate condizioni. Per esempio la prima transizione, relativa al fatto che il filosofo prende la forchetta alla sua destra, scatta sse con l'istanza filosofo p_j e forchetta f_i si ha che $i = j$ (ho quindi almeno una forchetta, almeno un filosofo e la forchetta deve essere quella alla sua destra, che, per come abbiamo modellato il problema, è quella con lo stesso indice). Sugli archi si hanno quindi annotate determinate variabili che denotano istanze. Per la forchetta a sinistra si usa il modulo 5. Si procede quindi per i vari filosofi.

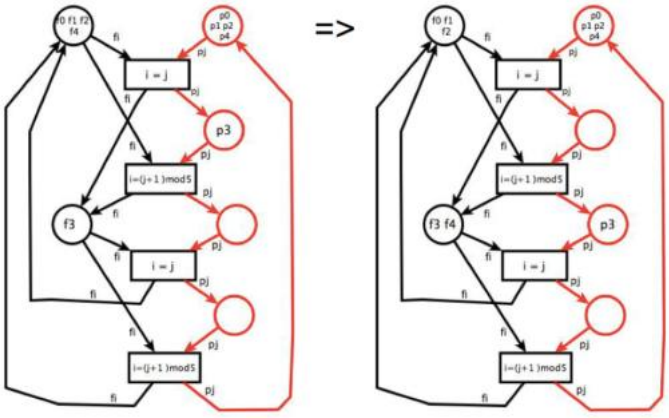
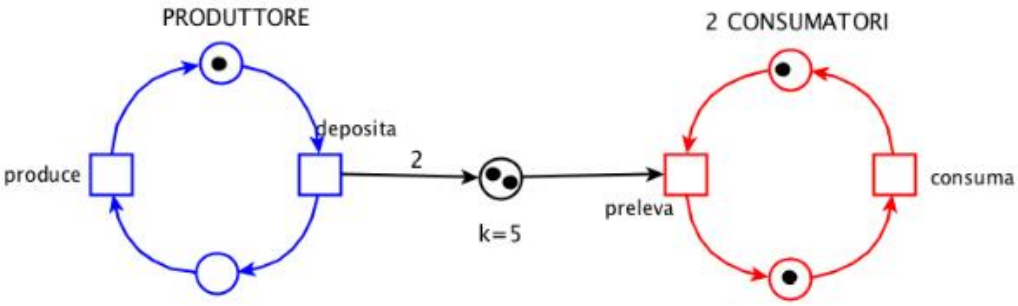


Figura 2.27: Esempio con il filosofo p_3 che ha già preso la forchetta alla sua destra, f_3, e prende quella a sinistra, f_4

Si ha quindi un prezzo per rappresentare in maniera compatta un sistema elementare, mediante una rete di alto livello, ovvero l'arricchimento della struttura dati.

2.2.2 Formalizzazione delle Reti Posti e Transizioni

Nonostante la sezione si occupi di formalismi ci appoggiamo ad un esempio per avere un confronto diretto tra teoria e pratica. Questo esempio è un'n-sima versione del sistema produttore-consumatore, con un produttore che deposita a due elementi alla volta, in un buffer che ha capacità massima apri a cinque, che vengono consumati da due consumatori:



quindi al massimo ho cinque marche nel buffer e 2 in uno degli stati del consumatore (in quanto rappresenta due consumatori).
Passiamo ora alla formalizzazione:

Definizione 26. *Si definisce un* ***sistema Posti e Transizioni (sistema P/T)*** *la sestupla:*

$$\Sigma = (S, T, F, K, W; M_0)$$

dove:

- (S, T, F) *è una rete con:*
  - S *che rappresenta l'insieme dei posti*
  - T *che rappresenta l'insieme delle transizioni*
  - F *che rappresenta la relazione di flusso che lega posti e transizioni tramite archi*
- $K : S \to \mathbb{N}^+ \cup \{\infty\}$ *che rappresenta una funzione che ad ogni posto in* S *assegna un valore, che può essere un naturale strettamente*

positivo o anche infinito (ovvero in quel posto posso avere un numero qualsiasi di marche). Il valore zero non è ammesso in quanto implicherebbe che il posto non potrebbe mai essere occupato, rendendo la modellazione di un tale posto inutile. K è detta ***funzione capacità dei posti****.*

- $W : F \to \mathbb{N} \cup \{\infty\}$ *che rappresenta una funzione che assegna ad ogni arco un peso mediante un valore naturale che stavolta può essere, oltre che infinito, anche zero. W è detta* ***funzione peso degli archi***

- $M_0 : S \to \mathbb{N} \cup \{\infty\} : \forall s \in S$ *t.c.* $M_0(s) \leq K(s)$ *che rappresenta la* ***marcatura iniziale*** *del sistema, ovvero una funzione che assegna ad ogni posto un naturale, eventualmente nullo o infinito, minore o uguale alla capacità massima di tale posto (capacità espressa dalla funzione K) indicante il numero di marche allo stato iniziale.*

Bisogna ora definire la regola di scatto, ovvero il ***gioco delle marche****, la regola per cui le marche si spostano sulla rete. Si ha quindi, dati:*

$$M : S \to \mathbb{N} \cup \{\infty\} \ e \ t \in T$$

ovvero data una marcatura e una qualsiasi transizione t si ha che:

$$M[t >$$

ovvero una transizione è abilitata in una certa marcatura,

sse:

$$\forall s \in S, \ M(s) \geq W(s,t) \wedge M(s) + W(t,s) \leq K(s)$$

ovvero per ogni posto si ha che ci sono abbastanza marche nei posti perché possa scattare la transizione, ovvero c'è un arco di peso corretto che collega quel posto con la transizione, avendo peso dell'arco minore o uguale al numero di marche del posto, e, inoltre, si deve verificare che la transizione non metta troppe marche in quel posto, quindi la marcatura del posto (ovvero il numero di marche già presenti in esso) più il numero di marche che si aggiungono con lo scatto della transizione non deve superare la capacità del posto.

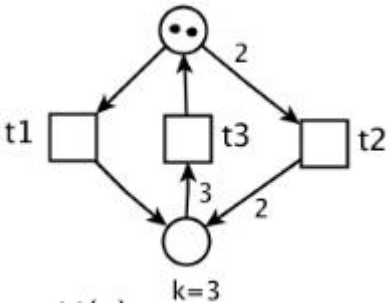


Figura 2.28: Nell'esempio si ha che la transizione t_2 può scattare sse nella posto precedente ho almeno due marche, in quanto l'arco tra i due ha peso due. Inoltre ho il posto che segue vuoto ma con capacità massima pari a tre, quindi la transizione può scattare in quanto l'arco tra i due pesa due, assicurandomi che dopo la transizione, nel posto che la segue, non verrà superata la capienza massima, arrivando infatti ad avere marcatura pari a 2

Lo scatto della transizione, se questa è abilitata nella marcatura, mi genera una nuova marcatura che viene ottenuta da quella precedente togliendo tante marche dal posto che è di input alla transizione quanto il peso dell'arco che connette tale posto alla transizione e aggiungendo tante marche al posto in output quante il peso dell'arco che connette la transizione a tale posto, ovvero, formalmente:

$$M[t > M'$$

sse

$$M[t > \wedge \forall s \in S,\ M'(s) = M(s) - W(s,t) + W(t,s)$$

quindi il nuovo posto avrà marcatura pari a quella precedente al più dei due contributi, il primo negativo e il secondo positivo, dei pesi dei due archi.

Definizione 27. *Dato un sistema P/T* $\Sigma = (S, T, F, K, W; M_0)$ *si definisce l'**insieme delle marcature raggiungibili**, dalla marcatura iniziale di* Σ *come:*

$$[M_0 >$$

ed esso è il più piccolo insieme tale che:

- $M_0 \in [M_0 >$*, ovvero ola marcatura iniziale appartiene all'insieme delle marcature raggiungibili*
- *se* $M \in [M_0 > \wedge \exists t \in T : M[t > M'$ *allora* $M' \in [M_0$*, ovvero se* M *appartiene all'insieme delle marcature raggiungibili ed esiste una transizione una transizione tale per cui* M *va in* M' *allora, di*

conseguenza si ha che M' appartiene all'insieme delle marcature raggiungibili dallo stato iniziale

Anche questa è una definizione per induzione

Esempio 11. *Vediamo qualche esempio particolare (se la capacità non è specificata si ha che essa è infinita):*

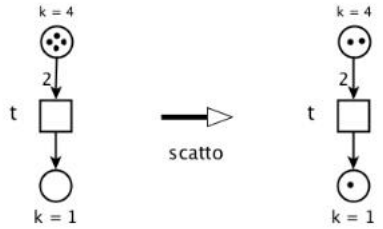


Figura 2.29: Lo scatto toglie due marche dall'input e ne mette una nell'output ma la seconda volta t non sarebbe più abilitata in quanto il posto in output si satura alla prima transizione, è un caso di **contatto**

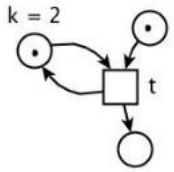


Figura 2.30: La transizione t è abilitata avendo i posti in ingresso col giusto numero di marche e in uscita ha un posto vuoto che può riempire a piacere e lo stesso porto che prima aveva in input che può riempire con una marca rispettando le regole di capacità

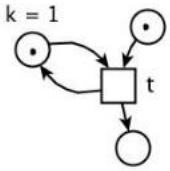


Figura 2.31: La transizione t non è abilitata, e non lo sarà mai in M in quanto la regola di scatto prevede che ci sia a priori una capacità sufficiente nei posti di output, cosa che qui non accade avendo uno di essi capacità uno, non si ragiona in modo sequenziale "prima tolgo e poi metto", lo spazio deve essere disponibile a priori per far scattare la transizione

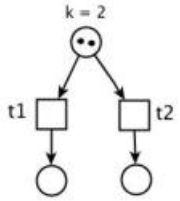


Figura 2.32: In questa rete si ha che t_1 e t_2 sono abilitate in M ed esse sono **concorrenti** in quanto lo scatto dell'una non disabilita lo scatto dell'altra dovendo entrambe prelevare una marca da un posto che ne contiene due per riversarne una ciascuna in un posto libero. Si ha quindi concorrenza anche in corrispondenza di transizioni non indipendenti

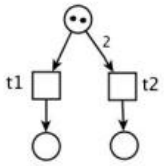


Figura 2.33: In questa rete si ha che t_1 e t_2 sono in **conflitto** e **non concorrenti** tra loro. Si ha infatti che lo scatto di una disabilita quella dell'altra, visto che da sola t_2 svuoterebbe il posto in input (se scatta prima t_1 poi ho una sola marca nello stato in input disabilitando t_2 che ne richiede due, viceversa lo scatto di t_2 lascerebbe vuoto lo stato in input impedendo a t_1 di scattare). Il conflitto è dato non dalla struttura della rete ma dalla sua marcatura iniziale. Quindi in questa rete con questa marcatura può scattare una sola delle due transizioni, inoltre t_1 potrebbe occorrere due volte senza errori, essendo quindi, se occorre contemporaneamente per quelle due volte, **concorrente con se stessa**. La doppia occorrenza di t_1, indicatat con $2t_1$ è un **passo**

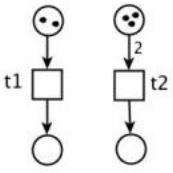


Figura 2.34: In questa rete si ha che t_1 è abilitata due volte, avendo quindi $2t_1$ e lo è anche t_2, che può scattare solo una volta. Si ha quindi che t_1 occorre concorrente con se stessa e contemporaneamente concorrente con t_2. Si ha quindi che il *multiset* $2t_1 + t_2$ è un **passo abilitato**

Si da ora la definizione formale di *multiset di transizioni abilitate* in un sistema:

Definizione 28. *Dato un sistema $P/T\ \Sigma = (S, T, F, K, W; M_0)$ si definisce un* ***multiset*** $U : T \to \mathbb{N}$ *come una funzione che assegna ad una transizione un numero naturale. Si ha inoltre che:*

- *un multiset è detto* ***concorrentemente abilitato*** *in $M \in [M_0 >$, quindi con M marcatura raggiungibile. Detto in maniera diversa U è un passo $M[U >$. Un multiset è concorrentemente abilitato in M sse:*

  $$\forall s \in S,\ \sum_{t \in T} U(t) \cdot W(s,t) \leq M(s)\ \wedge\ M(s) + \sum_{t \in T} U(t) \cdot W(t,s) \leq K$$

  ovvero per ogni posto si devono avere un numero di marche nella marcatura iniziale superiore al numero che posso togliere mediante le varie transizioni nel multiset U. Inoltre le marche che devo aggiungere più il numero di marche già presenti nel posto in output deve essere inferiore alla capacità dell'output.
  Questa condizione può essere riscritta come:

  $$\sum_{t \in T} U(t) \cdot W(s,t) \leq M(s) \leq K(s) - \sum_{t \in T} U(t) \cdot W(t,s)$$

- *un multiset U abilitato in M può occorrere generando M', avendo quindi:*

  $$M[U > M'$$

  sse:

  $$\forall s \in S\ M'(s) = M(s) - \sum_{t \in T} U(t) \cdot W(s,t) + \sum_{t \in T} U(t) \cdot W(t,s)$$

  quindi la marcatura M' in un certo posto è uguale alla marcatura del posto presente prima del passo al più dei contributi, positivi e negativi, delle varie transizioni (positivi se si ha tale posto come output e negativi se lo si ha come input)

- *U_Σ è l'**insieme dei passi** di Σ e viene definito come:*

  $$U_\Sigma = \{U : T \to \mathbb{N} |\ \exists M, M' \in [M_0 >:\ M[U > M'\}$$

  ovvero l'insieme dei passi possibili tali per cui passo da una marcatura raggiungibile ad un'altra attivando il passo

Definizione 29. *Dato un sistema P/T* $\Sigma = (S, T, F, K, W; M_0)$*, tale che* $\forall s \in SK(s) < \infty$*, si definisce il* ***grafo di raggiungibilità****, indicato con* $RG(\Sigma)$*, costruito solo con posti di capacità finita in quanto avere capacità infinite comporterebbe avere un grafo infinito, come la quadrupla:*

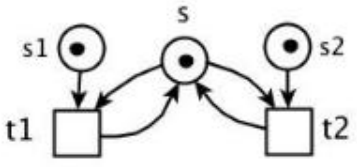
$$RG(\Sigma) = ([M_0 >, U_\Sigma, A, M_0)$$

dove, per rappresentare questo sistema di transizioni, si ha:

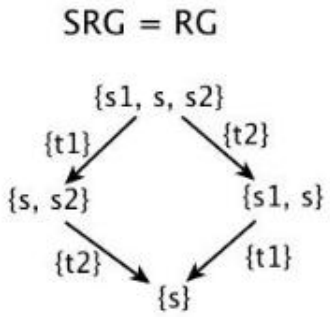
- $[M_0 >$ *che rappresenta le marcature raggiungibili dallo stato iniziale del sistema* Σ*. Sono così rappresentati gli stati del sistema di transizioni*
- U_Σ *rappresenta l'insieme dei passi che rappresenta l'alfabeto delle etichette degli archi del sistema di transizioni*
- $A = \{(M, U, M') : M, M' \in [M_0 > \wedge U \in U_\Sigma \wedge M[U > M'\}$ *dove, quindi dalla marcatura* M *ho un arco, etichettato* U*, alla marcatura* M' *sse le due marcature sono raggiungibili dallo stato iniziale, ed esiste un passo, nell'insieme dei passi, abilitato in* M *tale per cui posso passare dalla marcatura* M *alla marcatura* M'*. È quindi la regola che definisce come connettere mediante archi i vari stati*
- M_0 *che rappresenta la marcatura iniziale del sistema nonché lo stato iniziale del sistema di transizioni*

Se U *è una singola transizione si ha il grafo di raggiungibilità sequenziale, detto* $SGR(\Sigma)$*.*
La diamond property non è più valida in questa classe di reti a causa dei* self loop *che posso avere tra una transizione e un posto

Esempio 12. *Si prenda il sistema P/T* Σ*:*



Si ha che lo scatto di t_1 e lo scatto di t_2 non può essere contemporaneo in quanto si richiederebbero due marche nel posto s, devono quindi scattare in sequenza. SI ottiene che il grafo dei casi raggiungibili è uguale a quello dei casi raggiungibili sequenziale e quindi non vale la diamond property (non potendo aggiungere alcun arco tra $\{s_1, s, s_2\}$ e $\{s\}$):



Definizione 30. *Dato un sistema P/T si ha che una transizione t è in una* ***situazione di contatto*** *nella marcatura M sse:*

$$\forall s \in S, W(s,t) \leq M(s) \ \wedge M(s) + W(t,s) > K(s)$$

ovvero una transizione è in una situazione di contatto in una certa marcatura M sse ha abbastanza marche nei posti in input ma non abbastnza nei posti in output. Formalmente si ha che l'arco tra il posto in input e la transizioni ha peso inferiore al numero di marche dell'input e la marcatura del posto output più le marche che vengono aggiunte è maggiore della capacità del posto di output stesso .
Un esempio:

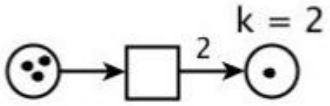


Figura 2.35: Esempio di situazione di contatto

Si cerca di capire se è possibile trasformare la rete da una con contatti ad una senza modificarne il comportamento. Questo è possibile con la **complementazione**. Preso un posto s se ne fa il complemento $\overline{s}$, ovvero si fa un altro posto che ha archi in output verso transizioni che sono di input a quelle del posto considerato e viceversa per gli archi in input. Il complemento è quindi collegato alle stesse transizioni del posto considerato ma con archi direzionati all'inverso. Per quanto riguarda il numero di marche del complemento bisogna avere che il numero di marche del posto e del suo complemento siano di somma pari alla capacità del posto (che è pari a quella del complemento), ovvero:

$$M_0(\overline{s}) = K(s) - M_0(s)$$

Si può inoltre dimostrare che per ogni marcatura raggiungibile M la somma delle marcature tra un posto e il suo complemento è sempre la medesima, comunque evolva il sistema:

$$M(\overline{s}) + M(s) = K(s)$$

Definizione 31. *Dato un sistema P/T* $\Sigma = (S, T, F, K, W; M_0)$ *è* ***senza contatti*** *sse:*

$$\forall M \in [M_0 >, \forall t \in T, \forall s \in S$$

si ha:

$$M(s) \geq W(s,t) \Rightarrow M(s) + W(t,s) \leq K(s)$$

ovvero ogni volta che ho abbastanza marche in un posto di input ad una transizione allora sicuramente non ne ho troppe nel posto di output

Definizione 32. *Dato un sistema P/T* $\Sigma = (S, T, F, K, W; M_0)$ *senza contatti allora si ha che una transizione* $t \in T$ *è abilitata in* $M \in [M_0 >$*, che si indica con* $M[t >$*, sse:*

$$\forall s \in S,\ M(s) \geq W(s,t)$$

ovvero sse nei posti in input ho abbastanza marche (trascurando di studiare i posti in output essendo una rete senza contatti).
Si ha quindi che in assenza di contatti la capacità dei posti non gioca più alcun ruolo nella regola di scatto in quanto controllo solo di avere sufficienti marche in input per far scattare una transizione

2.2.3 Reti Marcate

Si cerca di identificare i sistemi elementari come casi particolari dei sistemi P/T.

Definizione 33. *Dato un sistema P/T* $\Sigma = (S, T, F, K, W; M_0)$ *si ha che* Σ *è una* ***Rete Marcata*** *sse:*

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

ovvero per ogni posto la marcatura iniziale assegna un valore finito al posto che però ha capacità infinita (che quindi non gioca nessun ruolo, non vincolando lo scatto della transizione). Inoltre il peso di ogni arco deve essere minore o uguale a uno, imponendo quindi che abbia o peso uno o peso nullo (che segnala l'assenza dell'arco).
Questa sottoclasse delle reti P/T può essere denotata con la sola quadrupla:

$$\Sigma = (S, T, F; M_0)$$

in quanto la capacità, denotata dalla funzione K*, e il peso degli archi, denotato dalla funzione* W*, diventano ridondanti ed eliminabili dallo studio della rete.*

Definizione 34. *Una rete marcata è definita* ***safe (sicura)*** *sse per ogni marcatura raggiungibile, compresa quella iniziale, ogni posto contiene al massimo una marca (tornando quindi ad una definizione simil booleana dello stato del posto che o è vuoto o contiene una marca). Formalmente si ha che una rete marcata è safe sse:*

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

In una rete marcata i self-loop possono essere abilitati*:*

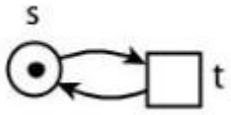


Figura 2.36: Esempio di loop (o cappio) su una rete marcata

Si ricorda che in una rete elementare in ogni caso il self-loop non è abilitato e questa è una delle differenze principale tra un sistema elementare e una rete marcata, differenza alle quali si aggiungono anche il fatto che in una rete marcata ho capacità infinita e non pari a uno e marcature di un posto intere e non booleane. Sia nei sistemi elementari che nelle reti marcate

ho solo archi di peso 1.
Si ha di conseguenza che un sistema elementare puro*, ovvero un sistema elementare che non presenta mai cappi,* coincide*,* $\cong$*, con una* rete marcata pura e safe*, quindi senza cappi e con al più una marca per posto (quindi o una marca o zero marche).*
Poter vedere un sistema elementare come sottoclasse delle reti P/T permette varie possibilità dal punto di vista pratico.

2.2.4 Proprietà di Comportamento

Si considerano sistemi P/T del tipo $\Sigma = (S, T, F, K, W; M_0)$ tali che $\forall s \in S$ si ha $K(s) = \infty$ quindi con capacità dei posti illimitata.

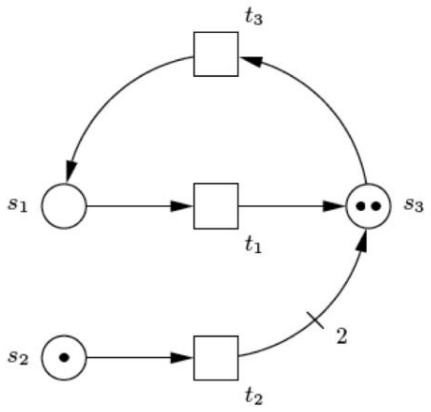


Figura 2.37: Esempio di sistema P/T

Abbiamo già visto la teoria dietro il grafo di raggiungibilità ma a questa vanno aggiunte le tecniche di rappresentazione. La differenza con i grafi di sistemi elementari si ritrova nel fatto che le marcature possono essere rappresentate da un vettore colonna, con tanti elementi quanti i posti della rete, contenente nell'i-sima posizione il numero di marche dell'i-simo posto in una data marcatura.
Solitamente il grafo di raggiungibilità viene usato per indagare le proprietà di comportamento.

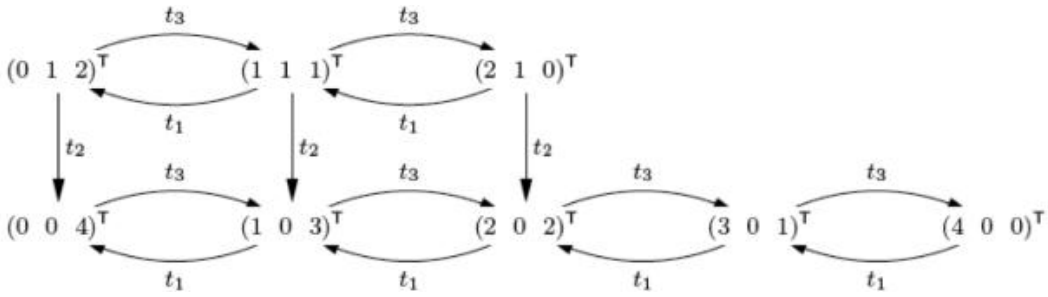


Figura 2.38: Grafo di raggiungibilità del sistema precedente, si nota, per esempio, che con $(0, 1, 2)$ si indica che in quello stato $M(s_1) = 0$,$M(s_2) = 1$ e $M(s_3) = 2$

Limitatezza

Vediamo innanzitutto un problema famoso, detto **problema di raggiungibilità**.
In tale problema si ha una rete e due marcature, M e M', e ci si domanda se dalla prima marcatura, raggiungibile dalla marcatura iniziale M_0, si può raggiungere la seconda. Formalmente si ha che:

$$RP = \{\langle (S, T, F, W), M, M' \rangle | (S, T, F, W) \text{ è una rete P/T, } M, M' : S \to \mathbb{N} \text{ e } M' \in [M >\}$$

Questo problema è stato dimostrato decidibile da Mayr, anche se esponenziale nello spazio. È un problema hard.
Si ha quindi l'importanza del fatto che il grafo delle marcature sia limitato e viene quindi studiato il **problema della limitatezza**.

Definizione 35. *Sia $\Sigma = (S, T, F, K, W; M_0)$ un sistema P/T con $K(s) = \infty, \forall s \in S$. Sia inoltre definito $n \in \mathbb{N},\ n \geq 1$. Si ha quindi che il sistema in analisi è:*

- ***n-bounded*** *(n-limitato) sse $\forall s \in S$ e $\forall M \in M_0[>$ si ha:*

$$M(s) \leq n$$

ovvero sse per ogni posto e per ogni marcatura raggiungibile da quella iniziale non succede che su tale posto si accumulino più di n marche

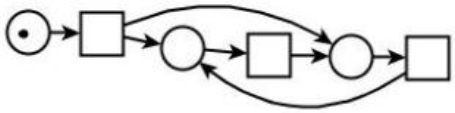


Figura 2.39: Esempio di sistema *2-bounded* (ma non 1-bounded)

- ***bounded*** *(limitato) sse* $\exists n \in \mathbb{N}$ *tale per cui* $\forall s \in S$ *e* $\forall M \in M_0[>$ *si ha:*

  $$M(s) \leq n$$

  ovvero se esiste un limite per il numero di marche

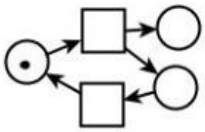


Figura 2.40: Esempio di sistema *non bounded*, si andrà in *overflow*

Si ha di conseguenza una proposizione: se Sigma è ***limitato*** *allora il numero delle marcature raggiungibili dalla marcatura iniziale è un insieme* ***finito****, ovvero* $[M_0 >$ *è un insieme finito, e quindi il grafo di raggiungibilità (il grafo delle marcature) è finito sia nel caso sia standard che in quello sia sequenziale:*

  - $SG(\Sigma)$ *è finito*
  - $SGR(\Sigma)$ *è finito*

- ***safe*** *(1-safe, sicuro) sse* $\forall s \in S$ *e* $\forall M \in M_0[>$ *si ha:*

  $$M(s) \leq 1$$

  ovvero ho al più una marca per posto comunque evolva il sistema

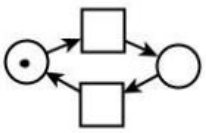


Figura 2.41: Esempio di sistema *safe* (ovvero 1-bounded)

Un sistema safe con n posti ha al più 2^n *marcature raggiungibili*

Terminazione

Un'altra proprietà interessante è quella della **terminazione**, in quanto è spesso necessario che un sistema sequenziale termini (a differenza di uno concorrente che spesso non ha termine).

Definizione 36. *Sia* $\Sigma = (S, T, F, K, W; M_0)$ *un sistema P/T. Si ha che:*

- Σ *è detto* ***terminante*** *sse non ammette sequenze infinite.*
  Inoltre $M \in [M_0 >$ *è una* ***marcatura di deadlock*** *sse* $\forall t \in T$ *non si ha* $M[t >$*, ovvero una marcatura in cui non è abilitata alcuna transizione.*
  Ne segue che un sistema è terminante se arriva in una marcatura di deadlock.

- Σ *è* ***deadlock-free*** *sse* $\forall M \in [M_0 >$*:*

  $$\exists t \in T : \ M[t >$$

  ovvero sse non esiste una marcatura di deadlock raggiungibile da quella iniziale. Scritto diversamente:

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

- Σ *è* ***1-live*** *(1-vivo) sse* $\forall t \in T$*:*

  $$\exists M \in [M_0 >: \ M[t >$$

  ovvero sse ogni transizione può essere abilitata almeno una volta (ho sempre una marcatura raggiungibile che abilita una transizione).
  Non avere un sistema 1-live implica un errore nella modellazione dello stesso in quanto si ha una transizione mai abilitata superflua

- Σ *è* ***live*** *(vivo) sse* $\forall t \in T$ *e* $\forall M \in [M_0 >$*:*

  $$\exists M' \in [M >: \ M'[t >$$

  ovvero sse per ogni transizione e per ogni marcatura raggiungibile esiste sempre un'altra marcatura raggiungibile che abilita la transizione.
  Si ha che un sistema live *comporta che tale sistema sia anche* deadlock-free *mentre il fatto che un sistema sia* deadlock-free *non implica necessariamente che sia anche* live

Esempio 13. *Vediamo qualche esempio:*

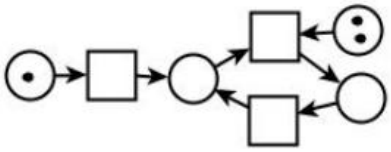


Figura 2.42: Esempio di sistema *terminante* (quindi non deadlock-free), in quanto scatta al più due volte

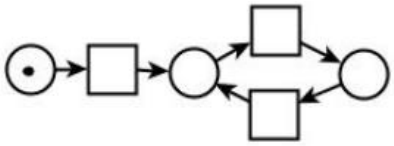


Figura 2.43: Esempio di sistema *deadlock-free*, *non terminante* e *1-live* ma non live, in quanto può continuare a cicalare, ogni transizione scatta almeno una volta ma la prima può scattare solo una volta e non può più essere attivata

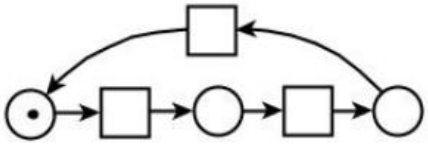


Figura 2.44: Esempio di sistema *live*, *non terminante* e *deadlock-free*, in quanto si ha un ciclo infinito lungo tutto il sistema

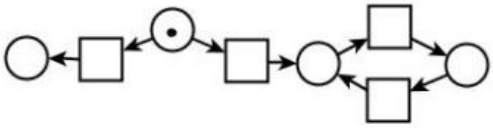


Figura 2.45: Esempio di sistema *1-live* ma non *live*, *non terminante* e *non deadlock-free*, in quanto potrebbe sia terminare che non terminare non potendo permettere la riattivazione di tutte le transizioni

Reversibilità

Un'altra proprietà di comportamento è quella della **reversibilità**, detta anche della **ciclicità**

Definizione 37. *Sia* $\Sigma = (S, T, F, K, W; M_0)$ *un sistema P/T. Si ha che* Σ *è* ***reversible*** *(reversibile sse:*

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

ovvero se da una marcatura raggiungile da quella iniziale posso tornare alla marcatura iniziale stessa.

Figura 2.46: A sinistra un esempio di sistema *reversible* e a destra uno *non reversible*

Si ha che avere un sistema reversible *e* 1-live *implica avere un sistema* live *ma non si ha per forza il contrario*

Tecniche di Verifica delle Proprietà

La prima tecnica usata per studiare le proprietà sopra descritte è quello dell'analisi del grafo di raggiungibilità $RG(\Sigma)$ e di quello di raggiungibilità sequenziale ($SGR(\Sigma)$).
Si ha innanzitutto che: *Se* $RG(\Sigma)$ *è* ***finito****, allora esistono algoritmi per decidere le seguenti proprietà:*

- un posto è *safe, m-bounded, bounded*
- il sistema Σ è *safe, n-bounded, limitato*
- una transizione è ***dead*** *(morta)*, *1-live*, *live* (*live* se per ogni marcatura raggiungibile ho un cammino che contiene tale transizione)
- il sistema Σ va in *deadlock*, è *deadlock-free, 1-live e live*
- il sistema Σ è *reversible*

Si hanno quindi due risultati interessati:

1. *un sistema* Σ ***terminante****, ovvero che va in* ***deadlock****, implica che il grafo di raggiungibilità* $RG(\Sigma)$ *ha almeno un* ***nodo terminante****, ovvero un nodo da cui non escono archi, e viceversa.*

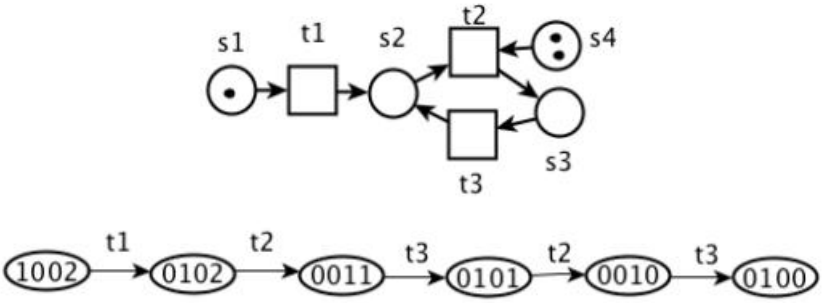


Figura 2.47: Esempio di un sistema terminante, col suo grafo di raggiungibilità che presenta, appunto, un nodo terminante, il nodo etichettato con "0100"

2. *un sistema* Σ ***reversible*** *implica che il grafo di raggiungibilità* $RG(\Sigma)$*, come del resto anche quello sequenziale* $SRG(\Sigma)$*, è* ***fortemente connesso****, e viceversa.*
   Si ricorda che un grafo orientato è fortemente connesso se per ogni coppia di nodi esiste un cammino orientato dal primo al secondo nodo.

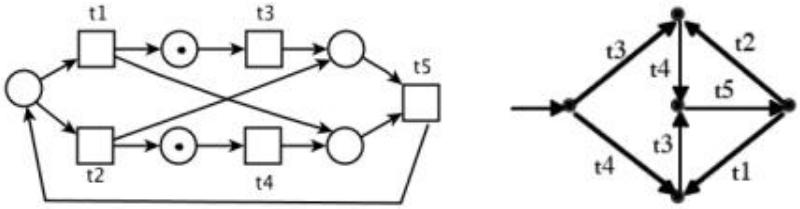


Figura 2.48: Esempio di sistema *live*, *1-safe* e *non reversible* con il grafo di raggiungibilità, quindi, *non fortemente connesso*

Oltre allo studio del grafo di raggiungibilità si hanno altre tecniche di verifica delle proprietà, tra cui l'**analisi strutturale del grafo della rete** che sfrutta:

- **tecniche di algebra lineare**, che sfruttano la rappresentazione algebrica della rete, mediante:
  - *equazioni di stato* che descrivono la dinamica attraverso operazioni algebriche tra matrici
  - *S-invarianti* e *T-invarianti*
- **studio del grafo della rete, di particolari sottoinsiemi di nodi e altre caratteristiche**
- **condizioni necessarie e sufficienti per garantire proprietà di comportamento di particolari sottoclassi di reti**

Safety, Liveness e Fairness

Le proprietà che specificano il comportamento dei **sistemi concorrenti e/o distribuiti** (detti anche **reattivi**) sono classificate in tre principali categorie, a seconda del tipo di comportamento che descrivono:

1. **proprietà di safety**, che descrivono proprietà che non devono mai accadere, dichiarano che "mai accadranno comportamenti indesiderati" (ad esempio: mai un semaforo avrà accese contemporaneamente le luci verde e rossa; mai due semafori ad un incrocio saranno contemporaneamente verdi).
   Nota: questa proprietà non ha nulla a che vedere con la 1-safeness *(la* safeness*) delle reti P/T*
2. **proprietà di liveness**, che descrivono proprietà che devono essere verificate da tutte le esecuzioni, dichiarano che "prima o poi un certo fatto deve accadere" (ad esempio, prima o poi il semaforo diventa verde)
3. **proprietà di fairness**, che descrivono proprietà che "descrivono fatti che devono accadere infinitamente spesso" (ad esempio, la luce verde si accende infinitamente spesso)

2.2.5 Analisi Strutturale

Spesso studiare il grafo di raggiungibilità comporta un costo esponenziale nel numero dei posti della rete, le marcature sono almeno dell'ordine di 2^{posti}. Si passa quindi dallo studio del grafo di raggiungibilità allo studio della rete stessa mediante la cosiddetta **analisi strutturale**, che studia il **grafo della rete**

Definizione 38. *Sia $\Sigma = (S, T, F; M_0)$ una **rete marcata**, reti con capacità dei posti illimitata e peso degli archi $\leq$ 1 (quindi o 0 o 1). Possiamo quindi studiare alcune proprietà strutturali che danno indicazioni sulle proprietà di comportamento:*

- *se un sistema Σ è* safe *e* bounded *allora:*

  $$\forall x \in S \cup T,\ {}^\bullet x \neq \emptyset \wedge x^\bullet \neq \emptyset$$

  ovvero per ogni nodo del grafo della rete non si ha mai che tale nodo abbia un insieme di pre-elementi vuoto (una transizione senza archi entranti sarebbe sempre abilitata permettendole di scattare infinite volte, rendendo il sistema non bounded*) e un insieme di post-elementi vuoto (una transizione senza archi uscenti, al suo scatto, non svuota le marche prese in input comportando il* non bounded *del sistema). Analogamente si ragiona per i posti.*
  *Questa è una **condizione necessaria***

- *se un sistema Σ è* safe *e* bounded *allora sicuramente il grafo della rete (S, T, F) è **strettamente connesso** (si ricorda che in un grafo strettamente connesso, presi due nodi, esiste sempre un cammino tra essi)*

- *se un sistema Σ è* safe *e* bounded *allora:*

  $$\exists M \in [M_o >, \exists \sigma \in T^* :\ M[\sigma > M$$

  tale che tutte le transizioni in T occorrono in σ

  ovvero esiste una marcatura raggiungibile ed esiste una sequenza di transizioni tale che a partire da quella marcatura, con quella sequenza, si torna nella stessa marcatura e in questa sequenza compaiono tutte le transizioni del mio sistema.
  Questa proprietà può essere analizzata sul grafo di raggiungibilità e non sul grafo della rete

Si hanno anche proprietà strutturali legate al fatto che il grafo della rete $N = (S, T, F)$ *sia connesso. Tale rete infatti è:*

- ***debolmente connessa*** *(detto anche solo* ***connessa****) sse:*

  $$\forall x, y \in S \cup T,\ (x, y) \in (F \cup F^{-1})^*$$

  ovvero presi due nodi qualunque tra gli insiemi dei posti e delle transizioni esiste un cammino non orientato ($(F \cup F^{-1})^*$*, dove lo star indica la chiusura transitiva) tra i due nodi*

- ***strettamente connessa*** *sse:*

  $$\forall x, y \in S \cup T,\ (x, y) \in F^*$$

  ovvero presi due nodi qualunque tra gli insiemi dei posti e delle transizioni esiste un cammino orientato (F^**) tra i due nodi*

Inoltre si ha che:

- *un* ***cammino semplice*** *di* N *è una sequenza*

  $$x_1 f_1 x_2 f_2 \dots f_{n-1} x_n$$

  che non passa mai due volte per uno stesso nodo e per lo stesso arco, ovvero, formalmente:

  $$x_i \in S \cup T \ e\ f_i \in F$$

- *un* ***ciclo semplice*** *è un cammino semplice tale che* $x_1 = x_n$*, ovvero il primo e l'ultimo nodo coincidono*

- *una rete* $N = (S, T, F)$ *è* ***coperta da cicli*** *sse,* $\forall f \in F$*, ovvero ogni arco, appartiene a qualche ciclo*

Si ha la seguente proposizione:

Data una rete $N = (S, T, F)$ si ha che essa è **strettamente connessa** sse N è **debolmente connessa** e **coperta da cicli**

Si hanno poi delle proprietà strutturali per reti senza marcature:

Definizione 39. *Data una rete $N = (P, T, F)$ senza marcature (con capacità illimitata e peso degli archi ≤ 1) si ha che può essere definita:*

- ***strutturalmente limitata*** *se:*

  $$\forall M_0 : \ (P, T, F; M_0) \ \text{è limitata}$$

  ovvero per ogni possibile marcatura iniziale il sistema è limitato

- ***strutturalmente viva*** *se:*

  $$\exists M_0 : \ (P, T, F; M_0) \ \text{è viva}$$

  ovvero se esiste almeno una marcatura iniziale tale per cui il sistema è vivo

- ***ben formata (WF, well formed)*** *se:*

  $$\exists M_0 : \ (N, M_0) \ \text{è viva e limitata}$$

  ovvero esiste almeno una marcatura tale per cui il sistema risulta essere sia vivo che limitato

Esempio 14. *Vediamo degli esempi per chiarire come la struttura della rete possa dare indicazioni sul comportamento.*

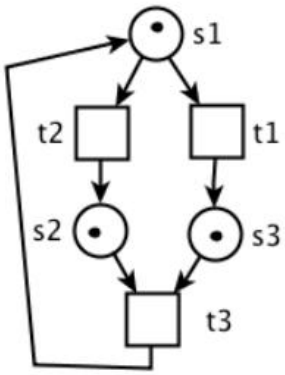


Figura 2.49: Esempio di rete *strutturalmente limitata non strutturalmente viva* e *non WF*, infatti t_3 scatta fino a svuotare i due pre-posti ma dopo lo scatto di una delle due transizioni tra t_1 e t_2 impedisce lo scatto dell'altra, portando ad uno stato di *deadlock*

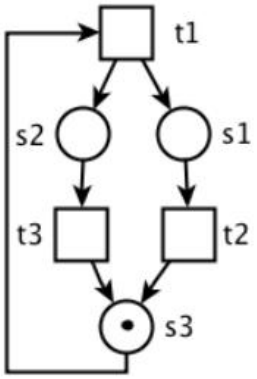


Figura 2.50: Esempio di rete *non strutturalmente limitata*, *strutturalmente viva* e *non WF*, infatti facendo scattare t_1 accumulo una marca in s_1 e una in s_2. Facendo poi scattare le altre due transizioni accumulo marche in s_2, in modo illimitato. D'altro canto la rete è *viva* in quanto può sempre scattare una transizione

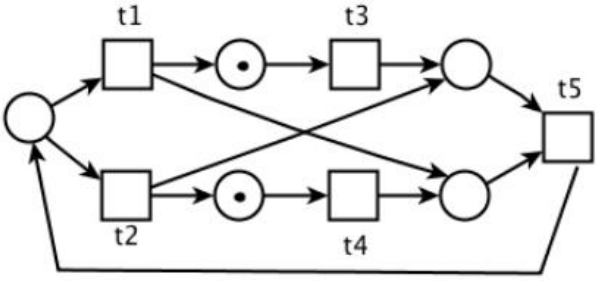


Figura 2.51: Esempio di rete *strutturalmente limitata*, *strutturalmente viva*, *WF* e *non reversible*, infatti in ogni posto si ha sempre o una o zero marche e può sempre scattare almeno una transizione

Si hanno anche altre proposizioni interessanti:

- se una rete N senza marcatura è **well formed** allora si ha che N è **coperta di cicli**.
  È una condizione necessaria
- *ricordando che N è **strettamente connessa** sse **debolmente connessa** e **coperta da cicli**,* si ha che se N è **well formed** e **debolmente connessa** allora N è **strettamente connessa**.
  È una condizione necessaria

quindi se una rete debolmente connessa *non è anche* strettamente connessa*, o* coperta da cicli*, non esiste una marcatura iniziale tale che il sistema così ottenuto sia* vivo *e* limitato

Rappresentazione Algebrica

Per poter fare analisi strutturale sulla rete è utile avere una **rappresentazione algebrica** del grafo della rete.

Definizione 40. *Sia* $\Sigma = (S, T, F, K, W; M_0)$ *un sistema P/T tale che* $\forall s \in S,\ K(S) = \infty$ *(quindi con capacità illimitata).*
Σ *può essere rappresentato da* ***due matrici*** *con* $|S|$ *(numero dei posti) righe e* $|T|$ *(numero delle transizioni) colonne:*

1. *una* ***matrice backward****,* $\underline{B} : S \times T \to \mathbb{N}$*, che contiene in posizione* i, j *il peso dell'arco che collega il posto* i*-simo alla transizione* j*-sima*

2. *una* ***matrice forward****,* $\underline{F} : S \times T \to \mathbb{N}$*, che contiene in posizione* i, j *il peso dell'arco che collega la transizione* j*-sima al posto* i*-simo*

La marcatura M_0 *può essere rappresentata da un vettore colonna* M_0 *di* $|S|$ *elementi.*

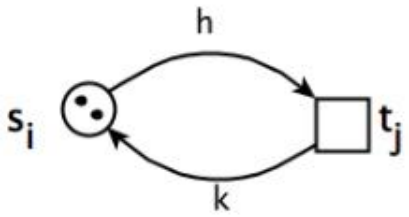


Figura 2.52: Esempio di una rete che comporta: $\underline{B}_{i,j} = W(s_i, t_j) = h$, $\underline{F}_{i,j} = W(t_i, s_j) = k$ e $M_{0_i} = M_0(s_i) = 2$

Con questa rappresentazione doppia si può facilmente rappresentare una rete a livello computazionale

Vediamo ora come rappresentare la regola di scatto usando le matrici.

Definizione 41. *Sia $M : S \to \mathbb{N}$ una marcatura e sia $t \in T$ una certa transizione. Si ha che la transizione è abilitata in tale marcatura ($M[t >$) sse:*

$$\forall s \in S,\ M(s) \geq W(s,t)$$

(ovvero se il numero di marche in ogni posto che funge da pre-posto alla transizione è maggiore o uguale al peso ddell'arco che collega il posto alla transizione)
ovvero sse:

$$\underline{M} \geq \underline{B}(t)$$

(ovvero nella matrice backward, nella colonna di tale transizione, ho pesi degli archi minori o uguali al numero di marche di ogni posto che precede tale transizione. Posso quindi confrontare la colonna che descrive la marcatura e la colonna nella matrice backward relativa alla transizione)
Inoltre si ha che tale transizione, se abilitata mi porta nella marcatura M' ($M[t > M'$) sse:

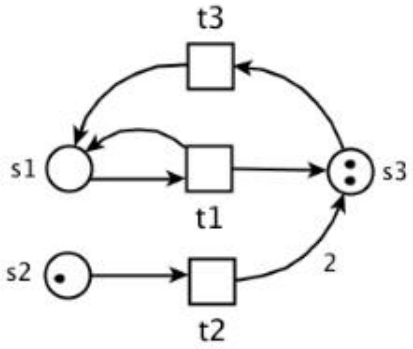
$$M[t > \wedge\ \forall s \in S, M'(s) = M(s) - W(s,t) + W(t,s)$$

(ovvero ottengo la nuova marcatura partendo dalla marcatura precedente sommando/sottraendo i contributi di archi entranti e uscenti) ovvero sse:

$$M[t > \wedge\ \underline{M'} = \underline{M} - \underline{B}(t) + \underline{F}(t) = \underline{M} + \underline{F}(t) - \underline{B}(t)$$

(ovvero usando le due matrici posso dire che se una transizione è abilitata allora lo scatto mi porta in una marcatura M' calcolabile prendendo la colonna M, togliendo i valori della colonna di t nella matrice backward e aggiungendo quelli della matrice forward (ovviamente nell'ordine che si preferisce essendo operazioni commutative))

Esempio 15. *Vediamo un esempio più completo sulla rete:*



dove si ha (i posti vuoti equivalgono a 0):

- *la matrice backward:*

$\underline{B}$	t_1	t_2	t_3
s_1	*1*		
s_2		*1*	
s_3			*1*

- *la matrice forward:*

$\underline{F}$	t_1	t_2	t_3
s_1	*1*		*1*
s_2			
s_3	*1*	*2*	

- *la marcatura iniziale:*

	M_0
s_1	
s_2	*1*
s_3	*2*

- $M_0[t_2 > M$*:*

	M
s_1	
s_2	
s_3	*4*

$=$

	M_0
s_1	
s_2	*1*
s_3	*2*

$-$

	B_{t_2}
s_1	
s_2	*1*
s_3	

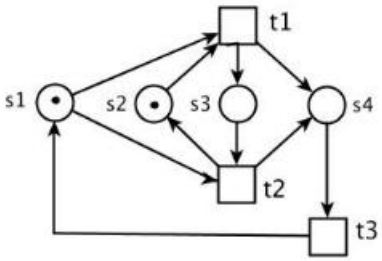
$+$

	F_{t_2}
s_1	
s_2	
s_3	*2*

Definizione 42. *Sia $\Sigma = (S, T, F, K, W; M_0)$ un sistema P/T tale che: $\forall s \in S$ $K(s) = \infty$ e $N = (S, T, F)$ sia **senza cappi**. Allora il sistema può essere rappresentato da un'unica matrice $\underline{N} : S \times T \to \mathbb{N}$ chiamata **matrice di incidenza**. Tale matrice ha $|S|$ righe e $|T|$ colonne e ha valore della posizione i, j è data dalla differenza tra la matrice forward e quella backward in tale posizione:*

$$\underline{N_{i,j}} = \underline{F_{i,j}} - \underline{B_{i,j}}$$

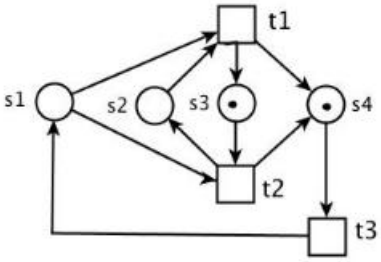
Esempio 16. ***Vedo quindi ogni transizione che effetto ha su un dato posto:***



$\underline{N} = \underline{F} - \underline{B}$

	t1	t2	t3
s1	-1	-1	1
s2	-1	1	
s3	1	-1	
s4	1	1	-1

dove, per esempio, la transizione t_1 toglie una marca a s_1 (-1) e aggiunge una marca in s_4 (1).
Vediamo anche il passaggio tra la marcatura iniziale M_0 (quella nellì'immagine sopra) e la marcatura M_1 (rappresentata nell'immagine seguente):



Aggiungiamo quindi le colonne delle marcature:

	t1	t2	t3	M0	M1
s1	-1	-1	1	1	
s2	-1	1		1	
s3	1	-1			1
s4	1	1	-1		1

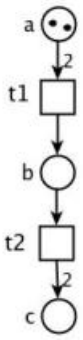
In M_0 è abilitata t_1 quindi:

$$M_0[t_1 > \Longleftrightarrow \underline{M_0} + \underline{t_1} \geq 0 \Longleftrightarrow \underline{M_0} + \underline{N_{t_1}} \geq 0$$

e si ottiene che:

$$M_0[t_1 > M_1 \Longleftrightarrow \underline{M_0} + \underline{t_1} = \underline{M_1}$$

ovvero il modo in cui calcolare M_1.
Vediamo un altro esempio:



$\underline{N} = \underline{F} - \underline{B}$

	t1	t2	M0	M1	M2
a	-2		2		
b	1	-1		1	
c		2			2

dove si vede, per esempio, che in M_0 la transizione t_2 non è abilitata, mentre, d'altro canto:

$$M_0[t_1 > M_1[t_2 > M_2 \Longleftrightarrow \underline{M_0} + \underline{t_1} + \underline{t_2} = \underline{M_2}$$

A partire dall'ultimo esempio si osserva che la colonna della transizione t_1 la posso ottenere dalla matrice moltiplicando tale matrice per il ***vettore caratteristico*** *di t_1, ovvero:*

$$\underline{t_1} = \underline{N_{t_1}} = \underline{N} \cdot \underline{c_{t_1}}$$

dove:

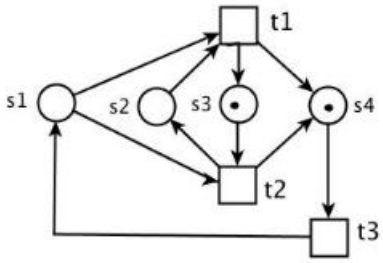
$$\underline{c_{t_1}} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

per cui si ha che:

$$\underline{M_0} + \underline{N} \cdot \underline{c_{t_1}} = \underline{M_1}$$

Definizione 43. *Sia $\sigma \in T^*$ una sequenza di transizioni.*
Il ***vettore di Parikh*** *(nome del ricercatore che lo ha introdotto) di σ è il vettore colonna di $|T|$ elementi, $\underline{c_\sigma}$, tale che $\underline{c_\sigma}(\underline{t_i})$ è il numero di occorrenze di t_i in σ.*

Esempio 17. *Dato:*



con:

	t1	t2	t3	M0	M1
s1	-1	-1	1	1	
s2	-1	1		1	
s3	1	-1			1
s4	1	1	-1		1

Si prenda per esempio $M_0[t_1t_3t_2t_3t_2 > M_1$.
Si ha che:

- $\sigma = t_1t_3t_2t_3t_2$
- $\underline{c_\sigma} = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix}$ *ovvero ho due transizioni* t_1*, una* t_2 *e due* t_3

Si ha quindi che:

$$M_0[\sigma > M_1 \Longrightarrow \underline{M_0} + \underline{N} \cdot \underline{c_\sigma} = \underline{M_1}$$

Quest'ultima equazione è detta ***equazione di stato*** *o* ***firing lemma*** *e descrive lo scatto di una sequenza, permettendo di simulare algebricamente il comportamento di una rete.*

La validità dell'equazione di stato è condizione necessaria, non sufficiente, affinché una sequenza di transizioni generi una marcatura in un sistema. *Infatti:*

- *se l'equazione non è soddisfatta, per un certo vettore di Parikh, allora non c'è una sequenza di transizioni, con quel vettore di Parikh, che faccia raggiungere quella marcatura*

- *se l'equazione è soddisfatta, con* $\underline{c_\sigma} \geq 0$*, non è detto che ci sia una sequenza* σ *di transizioni tale che* $M_0[\sigma > M_1$ *e in tal caso si dice che il vettore* $\underline{c_\sigma}$ ***non è realizzabile***

Esempio 18. *Vediamo un esempio:*

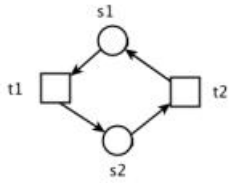


Figura 2.53: Esempio di una rete con la marcatura $\underline{M_0} = (0,0)^T$. Si ha che vale l'equazione $(0,0)^T + \underline{N} \cdot \underline{c_\sigma}$, con $\sigma = t_1 t_2$, ma tale sequenza σ non può scattare in M_0, quindi $\underline{c_\sigma}$ non è realizzabile

Capitolo 3

Logica PLTL

3.1 Ripasso Logica Proposizionale

Si ringrazia Marco Natali per questo ripasso
La logica è lo studio del ragionamento e dell'argomentazione e, in particolare, dei procedimenti inferenziali, rivolti a chiarire quali procedimenti di pensiero siano validi e quali no. Vi sono molteplici tipologie di logiche, come ad esempio la logica classica e le logiche costruttive, tutte accomunate di essere composte da 3 elementi:

- **Linguaggio**: insieme di simboli utilizzati nella Logica per definire le cose
- **Sintassi**: insieme di regole che determina quali elementi appartengono o meno al linguaggio
- **Semantica**: permette di dare un significato alle formule del linguaggio e determinare se rappresentano o meno la verità.

Noi ci occupiamo della logica Classica che si compone in *logica proposizionale* e *logica predicativa*.

La logica proposizionale è un tipo di logica classica che presenta come caratteristica quella di essere un linguaggio limitato in quanto si possono esprimere soltanto proposizioni senza avere la possibilità di estenderla a una classe di persone.

3.1.1 Sintassi

Il linguaggio di una logica proposizionale è composto dai seguenti elementi:

- Variabili Proposizionali: $P, Q, R \ldots$
- Connettivi Proposizionali: $\wedge, \vee, \neg, \rightarrow, \iff$
- Simboli Ausiliari: (,)
- Costanti: T, F

La sintassi di un linguaggio è composta da una serie di formule ben formate (FBF) definite induttivamente nel seguente modo:

1. Le costanti e le variabili proposizionali $\in FBF$.
2. Se A e $B \in FBF$ allora $(A \wedge B)$,$(A \vee B)$,$(\neg A)$,$(A \rightarrow B)$, $(A \iff B)$, TA e FA sono delle formule ben formate.
3. nient'altro è una formula

Esempio 19. *Vediamo degli esempi:*

- $(P \wedge Q) \in Fbf$ *è una formula ben formata*
- $(PQ \wedge R) \notin Fbf$ *in quanto non si rispetta la sintassi del linguaggio definita.*

Definizione 44. *Sia $A \in FBF$, l'insieme delle sottoformule di A è definito come segue:*

1. *Se A è una costante o variabile proposizionale allora A stessa è la sua sottoformula*
2. *Se A è una formula del tipo $(\neg A')$ allora le sottoformule di A sono A stessa e le sottoformule di A'; $\neg$ è detto connettivo principale e A' sottoformula immediata di A.*
3. *Se A è una formula del tipo $B \circ C$, allora le sottoformule di A sono A stessa e le sottoformule di B e C; $\circ$ è il connettivo principale e B e C sono le due sottoformule immediate di A.*

È possibile ridurre ed eliminare delle parentesi attraverso l'introduzione della precedenza tra gli operatori, che è definita come segue:
$\neg, \wedge, \vee, \rightarrow, \iff$.

In assenza di parentesi una formula va parentizzata privileggiando le sottoformule i cui connettivi principali hanno la precedenza più alta.
In caso di parità di precedenza vi è la convenzione di associare da destra a sinistra.

Esempio:
$\neg A \wedge (\neg B \rightarrow C) \vee D$ diventa $((\neg A) \wedge ((\neg B) \rightarrow C) \vee D)$.

Definizione 45. *Un albero sintattico T è un albero binario coi nodi etichettati da simboli di L, che rappresenta la scomposizione di una formula ben formata X definita come segue:*

1. Se X è una formula atomica,l'albero binario che la rappresenta è composto soltanto dal nodo etichettato con X
2. Se $X = A \circ B$, X è rappresentata da un albero binario che ha la radice etichettata con $\circ$, i cui figli sinistri e destri sono la rappresentazione di A e B
3. Se $X = \neg A$,X è rappresentato dall'albero binario con radice etichettata con $\neg$, il cui figlio è la rappresentazione di A

Poiché una formula è definita mediante un albero sintattico, le proprietà di una formula possono essere dimostrate mediante induzione strutturale sulla formula, ossia dimostrare che la proprietà di una formula soddisfi i seguenti 3 casi:

- è verificata la proprietà per tutte le formule atomo A
- supposta verifica la proprietà per A, si verifica che la proprietà è verificata per $\neg A$
- supposta la proprietà verificata per A_1 e A_2, si verifica che la proprietà è verifica per $A_1 \circ A_2$, per ogni connettivo $\circ$.

3.1.2 Semantica

La semantica di una logica consente di dare un significato e un interpretazione alle formule del Linguaggio.

Definizione 46. *Sia data una formula proposizionale P e sia $P_1, \ldots, P_n$, l'insieme degli atomi che compaiono nella formula A.Si definisce come* interpretazione *una funzione $v : \{P_1, \ldots, P_n\} \mapsto \{T, F\}$ che attribuisce un valore di verità a ciascun atomo della formula A.*

I connettivi della Logica Proposizionale hanno i seguenti valori di verità:

A	B	$A \wedge B$	$A \vee B$	$\neg A$	$A \Rightarrow B$	$A \iff B$
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

Essendo ogni formula A definita mediante un unico albero sintattico, l'interpretazione v è ben definito e ciò comporta che data una formula A e un interpretazione v, eseguita la definizione induttiva dei valori di verità, si ottiene un unico $v(A)$.

Definizione 47. *Una formula nella logica proposizionale può essere di diversi tipi:*

- ***valida o tautologica:*** *la formula è soddisfatta da qualsiasi valutazione della Formula*
- ***Soddisfacibile non Tautologica:*** *la formula è soddisfatta da qualche valutazione della formula ma non da tutte.*
- ***falsificabile:*** *la formula non è soddisfatta da qualche valutazione della formula.*
- ***Contraddizione:*** *la formula non viene mai soddisfatta*

Teorema 1. *Si ha che:*

- *A è una formula valida se e solo se $\neg A$ è insoddisfacibile.*
- *A è soddisfacibile se e solo se $\neg A$ è falsificabile*

Modelli e decidibilità

Si definisce *modello*, indicato con $M \models A$, tutte le valutazioni booleane che rendono vera la formula A. Si definisce *contromodello*, indicato con , tutte le valutazioni booleane che rendono falsa la formula A.

La logica proposizionale è decidibile (posso sempre verificare il significato di una formula). Esiste infatti una procedura effettiva che stabilisce la validità o no di una formula, o se questa ad esempio è una tautologia. In particolare il verificare se una proposizione è tautologica o meno è l'operazione di decidibilità principale che si svolge nel calcolo proposizionale.

Definizione 48. *Se $M \models A$ per tutti gli M, allora A è una tautologia e si indica $\models A$*

Definizione 49. *Se $M \models A$ per qualche M, allora A è soddisfacibile*

Definizione 50. *Se $M \models A$ non è soddisfatta da nessun M, allora A è insoddisfacibile*

3.1.3 Equivalenze Logiche

Date due formule A e B, si dice che A è *logicamente equivalente* a B, indicato con $A \equiv B$, se e solo se per ogni interpretazione v risulta $v(A) = v(B)$.

Nella logica proposizionale sono definite le seguenti equivalenze logiche, indicate con $\equiv$,:

1. **Idempotenza:**

$$A \vee A \equiv A$$
$$A \wedge A \equiv A$$

2. **Associatività:**

$$A \vee (B \vee C) \equiv (A \vee B) \vee C$$
$$A \wedge (B \wedge C) \equiv (A \wedge B) \wedge C$$

3. **Commutatività:**

$$A \vee B \equiv B \vee A$$
$$A \wedge B \equiv B \wedge A$$

4. **Distributività:**

$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$$
$$A \wedge (B \vee C) \equiv (A \wedge B \vee (A \wedge C)$$

5. **Assorbimento:**

$$A \vee (A \wedge B) \equiv AA \wedge (A \vee B) \equiv A$$

6. **Doppia negazione:**

$$\neg\neg A \equiv A$$

7. **Leggi di De Morgan:**

$$\neg(A \vee B) \equiv \neg A \wedge \neg B$$
$$\neg(A \wedge B) \equiv \neg A \vee \neg B$$

8. **Terzo escluso:**

$$A \vee \neg A \equiv T$$

9. **Contrapposizione:**

$$A \rightarrow B \equiv \neg B \rightarrow \neg A$$

10. **Contraddizione**

$$A \wedge \neg A \equiv F$$

Completezza di insiemi di Connettivi

Un insieme di connettivi logici è completo se mediante i suoi connettivi si può esprimere un qualunque altro connettivo. Nella logica proposizionale valgono anche le seguenti equivalenze, utili per ridurre il linguaggio:

$$(A \rightarrow B) \equiv (\neg A \vee B)$$
$$(A \vee B) \equiv \neg(\neg A \wedge \neg B)$$
$$(A \wedge B) \equiv \neg(\neg A \vee \neg B)$$
$$(A \iff B) \equiv (A \rightarrow B) \wedge (B \rightarrow A)$$

L'insieme dei connettivi $\{\neg, \vee, \wedge\}$, $\{\neg, \wedge\}$ e $\{\neg, \vee\}$ sono completi e ciò è facilmente dimostrabile utilizzando le seguenti equivalenze logiche.

3.2 Logica PLTL

La semplice logica proposizionale ha però dei limiti, si introduce quindi la **logica PLTL (*Propositional Linear Time Logic*)** che introduce il concetto di **tempo**, in ottica **lineare**, nel processo logico.
La logica PLTL viene usata per **model checking** e lo scopo generale che ci si propone è quello di presentare un approccio formale alla progettazione e all'implementazione di sistemi basato su un linguaggio formale che permetta di specificarli e ragionare sulle loro proprietà. Si vogliono studiare sia sistemi grandi, come software o addirittura sistemi operativi, che piccoli, come per esempio una coppia di semafori. Tutti questi sistemi hanno però la medesima caratteristica, ovvero assumono in ogni **istante di tempo** un determinato **stato** definito dalle **variabili** del sistema stesso. Si hanno quindi nel sistema, in ogni istante di tempo:

- uno **stato**, univocamente definito dai valori delle variabili
- una **transizione** che segnala un passaggio da uno stato ad un altro
- una **computazione** che rappresenta una sequenza di stati di cui ogni coppia forma una transizione. Si introduce quindi il concetto **temporale**

Il **tempo** viene pensato come un oggetto discreto, cadenzato dalle transizioni, su cui può essere definita anche una relazione d'ordine (potrebbe servire che un processo termini prima, dopo o nello stesso tempo di un altro).

Definizione 51. *Si definisce **sistema reattivo** una componente:*

- *non terminante e interattivo*
- *che può leggere il proprio input non solo all'inizio della computazione e che può produrre output non solo alla fine della sua computazione (si possono quindi avere multipli input e multipli output)*
- *che interagisce con altri componenti distribuiti o concorrenti*

Si hanno però sistemi non terminanti che usano un numero **finito** di variabili e di conseguenza si ha un numero di stati, in cui transita il sistema, **finito**. Si hanno infatti sistemi di transizione finiti che *"comprimono"* sistemi non terminanti, con computazioni di lunghezza infinita, in una rappresentazione finita.

Si ricorda che si possono inoltre categorizzare le proprietà che vogliamo specificare come:

- **proprietà di safety**
- **proprietà di liveness**
- **proprietà di fairness**

Si può quindi già intuire che:

le logiche temporali sono frammenti della logica del primo ordine

Infatti con le logiche temporali, come la *logica PLTL*, si supera il limite di rappresentazione temporale della logica proposizionale, permettendo, per esempio, di rappresentare proposizioni del tipo *"prima o poi", "accade sempre che" etc...*.
Si ha un **tempo lineare e discreto**, che si può pensare di rappresentare nella logica classica proposizionale (per esempio come indice) ma questo comporta il doversi dotare di infinite variabili proposizionali (corrispondenti ad ogni *"step"* temporale) e comporta l'ottenimento di una formula proposizionale di lunghezza infinita. Si può pensare di passare allo studio di questi casi con la *logica predicativa* ma sarebbe ben più del necessario, in quanto le procedure sarebbero di complessità molto elevata, a causa della grande espressività della logica predicativa. Inoltre la logica dei predicati è indecidibile.
Si cerca quindi la via di mezzo cercando logiche specializzate, meno espressive della logica predicativa ma decidibili rispetto ai problemi presi in considerazione. Si cerca una logica con procedure efficienti rispetto all'ampiezza della descrizione del problema in analisi (ovvero tipicamente l'ampiezza del sistema di transizioni), che è direttamente proporzionale al numero di stati del sistema (lineare rispetto al numero degli stati), e rispetto alla lunghezza della formula che esprime la proprietà da testare.

3.2.1 Sintassi

Vediamo innanzitutto la **sintassi** della logica PLTL.
Si ha a che fare con un *vocabolario* più ampio rispetto a quelli della logica proposizionale, vengono infatti aggiunti dei **connettivi** utili alla rappresentazione temporale richiesta.

Definizione 52. *Nella logica PLTL, oltre ai connettivi logici classici della logica proporzionale, si definiscono i seguenti connettivi:*

- ***X**, detto anche **next** o **tomorrow**. È un connettivo unario e viene rappresentato con* $\circ$
- ***F**, detto anche **sometime** o **future**. È un connettivo unario e viene rappresentato con* $\diamond$
- ***G**, detto anche **globally** o **always**. È un connettivo unario e viene rappresentato con* $\square$
- ***U**, detto anche **until**. È un connettivo binario*

*Viene inoltre indicato con V l'**insieme delle variabili proposizionali**, variabili che hanno lo stesso significato della logica proposizionale, quindi ogni variabile corrisponde ad una singola proposizione del linguaggio naturale.*

Definizione 53. *Diamo ora una definizione, formale, procedendo per induzione di **formula** del linguaggio PLTL (definendo così l'aspetto sintattico della logica PLTL):*

- $\forall p \in V$ *vale che p è una formula del linguaggio PLTL, ovvero le variabili proposizionali della logica classica sono formule del linguaggio PLTL. Questa è una* formula atomica
- *i simboli* $\top$ *(detto anche* true *che semanticamente semplifica una tautologia) e* $\bot$ *(detto anche false o* bottom *che semanticamente specifica una formula sempre falsa, ovvero una contraddizione) sono formule del linguaggio PLTL. Anche questa è una* formula atomica
- *se A è una formula del linguaggio PLTL allora lo sono anche:*
  - $\neg A$
  - $\mathbf{X}A$
  - $\mathbf{F}A$
  - $\mathbf{G}A$

- *se A e B sono formule del linguaggio PLTL allora lo sono anche:*
  - $A \rightarrow B$
  - $A \wedge B$
  - $A \vee B$
  - $A\mathbf{U}B$
- *nient'altro appartiene all'insieme delle formule del linguaggio PLTL*

Quindi si nota come ***l'insieme delle formule PLTL contiene quello delle formule classiche della logica proporzionale****. Si ha quindi che l'insieme delle formule PLTL non è altro che un'estensione di quello delle formule classiche della logica proposizionale.*

Definizione 54. *La sintassi delle formule del linguaggio PLTL possono essere definite usando anche la notazione* ***BNF (Backus-Naur Form o Backus Normal Form)****, ovvero, $\forall p \in V$:*

$$A ::= p \mid \top \mid \bot \mid (\neg A) \mid (A \wedge A) \mid (A \vee A) \mid (A \rightarrow A) \mid (\mathbf{X}A) \mid \mathbf{F}A \mid \mathbf{G}A \mid (A\mathbf{U}A)$$

Non si è usata quindi una definizione ricorsiva ma viene invece usata una ***grammatica***

La BNF (Backus-Naur Form o Backus Normal Form) è una metasintassi, ovvero un formalismo attraverso cui è possibile descrivere la sintassi di linguaggi formali (il prefisso meta ha proprio a che vedere con la natura circolare di questa definizione). Si tratta di uno strumento molto usato per descrivere in modo preciso e non ambiguo la sintassi dei linguaggi di programmazione, dei protocolli di rete e così via, benché non manchino in letteratura esempi di sue applicazioni a contesti anche non informatici e addirittura non tecnologici. La BNF viene usata nella maggior parte dei testi sulla teoria dei linguaggi di programmazione e in molti testi introduttivi su specifici linguaggi.
In termini formali, la BNF può essere vista come un formalismo per descrivere grammatiche libere dal contesto.
Una specifica BNF è un insieme di regole di derivazione ciascuna espressa nella forma:

*<simbolo> ::= _espressione_*

3.2.2 Semantica

Definizione 55. *La **semantica**, ovvero il significato dei connettivi della logica PLTL, è data usando i cosiddetti **modelli lineari** (si ricorda l'uso di un tempo lineare).*
Si consideri una struttura algebrica di questo tipo:

$$M = \langle S, \rho, \rightarrow, \Vdash \rangle$$

dove:

- S *è un **insieme infinito di stati**, detti anche **mondi***
- $\rho \in S$ *è uno stato del modello detto anche **root** o **radice**. In tale stato viene codificato il **tempo zero***
- $\rightarrow$ *è una relazione binaria su S detta **relazione di transizione** la quale introduce un **ordinamento lineare** sugli elementi di S. Si ha quindi che:*

  $$\rightarrow \subseteq S \times S$$

  *e, $\forall \alpha \in S$, **esiste ed è unico** $\beta \in S$ tale che vale:*

  $$\alpha \rightarrow \beta$$

  Preso quindi uno stato qualsiasi ho un solo modo per passare ad un altro stato (esiste quindi un "prima" e un "dopo").
  Inoltre vale che, $\forall \alpha \in S$, $\alpha \not\rightarrow \rho$, dove $\rho \in S$ è l'unico elemento di S a godere di questa proprietà (in quanto ρ codifica il tempo zero).
- $\Vdash$ *è una relazione binaria, inclusa in $S \times V$, detta **relazione di soddisfacibilità**. Solitamente con $\alpha \Vdash$ si indica che $(\alpha,) \in \Vdash$ è valido, ovvero che alpha **soddisfa** p, con $\alpha \in S$ e $p \in V$*

*La struttura M viene detta **modello per PLTL**.*
Si nota come questi modelli lineari seguano un ordinamento simile a quello che si ha tra i numeri in $\mathbb{N}$.
Si nota come la semantica della logica PLTL sia drasticamente più complessa di quella della logica classica proposizionale

Bisogna dare ora significato ai connettivi PLTL. Per i connettivi della logica proposizionale si usano tavole di verità e induzione ma per la logica PLTL le cose sono un po' diverse.
Per dare un significato alle formule del linguaggio PLTL bisogna basarsi sui modelli per PLTL

Definizione 56. *Siano:*

- $M = \langle S, \rho, \rightarrow, \Vdash \rangle$ *un modello per PLTL*
- $\alpha \in S$ *uno stato*
- A *una formula del linguaggio PLTL*

si ha che la ***soddisfacibilità*** *di A nello stato α di M viene indicata con:*

$$(M, \alpha) \Vdash A$$

e tale soddisfacibilità (o verità) della formula A viene definita per induzione sulla struttura di A nel seguente modo:

1. *se A è una variabile proposizionale allora vale $(M, \alpha) \Vdash A$ sse $\alpha \Vdash A$ per la relazione $\Vdash$ definita da M. Questo è un* ***caso base***
2. *se A è $\top$ allora vale $(M, \alpha) \Vdash \top$, ovvero il true è* ***soddisfatto*** *o* ***forzato*** *ovunque. Il simbolo* true *è vero in ogni stato e in ogni modello (siamo nel concetto di tautologia). Questo è un* ***caso base***
3. *se A è $\bot$ allora vale $(M, \alpha) \nVdash \bot$, ovvero il false non è mai soddisfatto qualsiasi sia lo stato. Questo è un* ***caso base***
4. *se A è $B \wedge C$ (con B e C formule PLTL) allora vale $(M, \alpha) \Vdash B \wedge C$ sse $(M, \alpha) \Vdash B$ e $(M, \alpha) \Vdash C$*
5. *se A è $B \vee C$ (con B e C formule PLTL) allora vale $(M, \alpha) \Vdash B \vee C$ sse $(M, \alpha) \Vdash B$ oppure $(M, \alpha) \Vdash C$*
6. *se A è $B \rightarrow C$ (con B e C formule PLTL) allora vale $(M, \alpha) \Vdash B \rightarrow C$ sse $(M, \alpha) \nVdash B$ oppure $(M, \alpha) \Vdash C$*
7. *se A è $\neg B$ (con B formula PLTL) allora vale $(M, \alpha) \Vdash \neg B$ sse vale $(M, \alpha) \nVdash B$*

8. *se A è del tipo $\mathbf{X}\,B$ allora vale $(M, \alpha) \Vdash \mathbf{X}\,B$ sse vale $(M, \beta) \Vdash B$ (ovvero quando B è soddisfatto nello stato β di M), dove $\beta \in S$ è lo stato successivo (nonché unico) ad α, ovvero si ha $\alpha \to \beta$*

9. *se A è del tipo $\mathbf{F}\,B$ allora vale $(M, \alpha) \Vdash \mathbf{F}\,B$ sse esiste una sequenza finita $\alpha_0 \to \alpha_1 \to \ldots \to \alpha_n$, con $\alpha = \alpha_0$ e $n \geq 0$, tale che $(M, \alpha_n) \Vdash B$ (ovvero sse esiste un punto "nel futuro", ovvero $alpha_n$, che però potrebbe anche essere α_0, dove la formula B è soddisfatta)*

10. *se A è del tipo $B\,\mathbf{U}\,B$ allora vale $(M, \alpha) \Vdash B\,\mathbf{U}\,C$ sse esiste una sequenza finita $\alpha_0 \to \alpha_1 \to \ldots \to \alpha_n$, con $\alpha = \alpha_0$ e $n \geq 0$, tale che $(M, \alpha_n) \Vdash C$ e tale che $(M, \alpha_i) \Vdash B,\ i = 0, 1, \ldots, n-1$ (ovvero deve esistere uno stato "nel futuro" di α che soddisfa C e uno stato intermedio a quello che soddisfa C che soddisfi B)*

11. *se A è del tipo $\mathbf{G}\,B$ allora vale $(M, \alpha) \Vdash \mathbf{G}\,B$ sse $(M, \alpha) \Vdash B$ e $(M, \beta) \Vdash \mathbf{G}\,B$ dove $\beta \in S$ è lo stato successivo (nonché unico) ad α, ovvero si ha $\alpha \to \beta$ (ovvero sse nello stato α è soddisfatto B e nello stato β è soddisfatto $\mathbf{G}\,B$). Questa è una definizione ricorsiva (anche se uso $\mathbf{G}\,B$ su uno stato successivo ad α)*

I primi 3 punti sono i casi base, quelli successivi sono i passi induttivi.
I primi 7 punti sono simili a quelli della logica proposizionale, se ci limitiamo ad un solo mondo