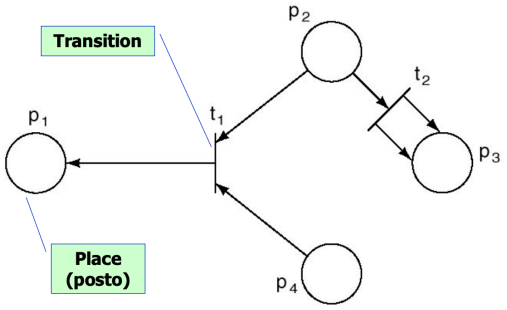
- ***Lezione 10 – (11/11/2024)***

Un esempio di linguaggio di modellazione formale, dotato di una sintassi visuale semplice e intuitiva, è la *rete di Petri (Petri Net).* Sebbene questo linguaggio non sia stato originariamente concepito per la specifica del software, si è rivelato estremamente efficace in tale ambito.

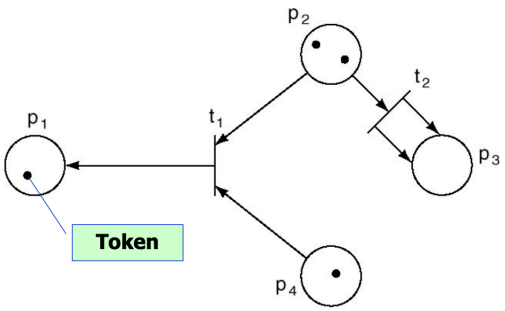


I costrutti essenziali delle reti di Petri sono:

1. *Posti (places):* rappresentano componenti o stati di un sistema.
2. *Transizioni (transitions):* modellano le azioni o esecuzioni che provocano un cambiamento di stato nel sistema.

Le reti di Petri vengono utilizzate per descrivere il *comportamento dinamico di un sistema*, mostrando come questo evolve partendo da uno stato iniziale, attraversando stati intermedi, fino a raggiungere uno stato finale. L'obiettivo è esplorare lo **spazio degli stati** del sistema, analizzandone i possibili comportamenti e valutando alcune proprietà fondamentali.

Dal punto di vista strutturale, i *posti* e le *transizioni* sono collegati tramite *archi orientati*. I posti possono contenere dei *token*, elementi che abilitano l'esecuzione delle transizioni. Questo meccanismo rende la rete di Petri un potente strumento per rappresentare il comportamento dinamico di un sistema.

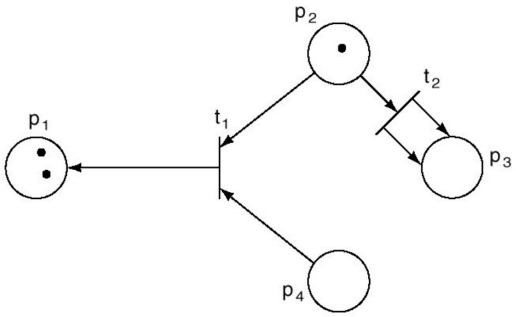


Affinché una transizione possa essere eseguita (o, in termini tecnici, *firing)*, deve essere abilitata. Una transizione è abilitata quando:

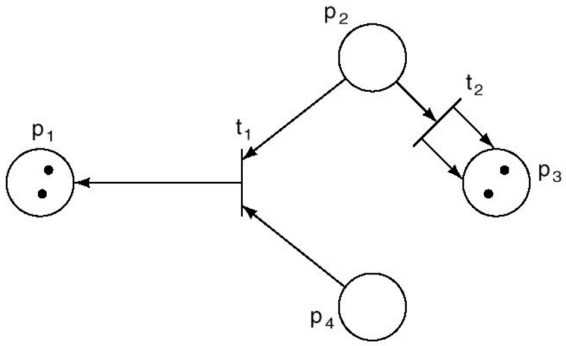
- Ogni posto collegato in ingresso alla transizione contiene almeno un *token.*

Quando questa condizione è soddisfatta, la transizione può "scattare", consumando i token dai posti in ingresso e producendo token nei posti in uscita.

Esempio1: se facciamo scattare la transizione da t_1, facciamo scattare un token da p_2 e p_4 e poi ne generiamo uno solo (arco uscente uno solo), il quale viene depositato nel posto p_1

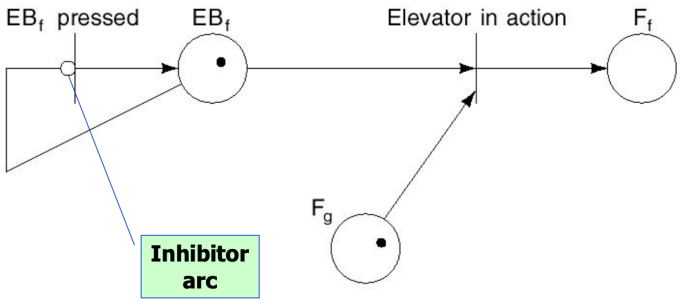


A questo punto scatta pure la transizione da t_2, e non essendoci almeno un token in ogni posto collegato in input, abbiamo uno stato finale.

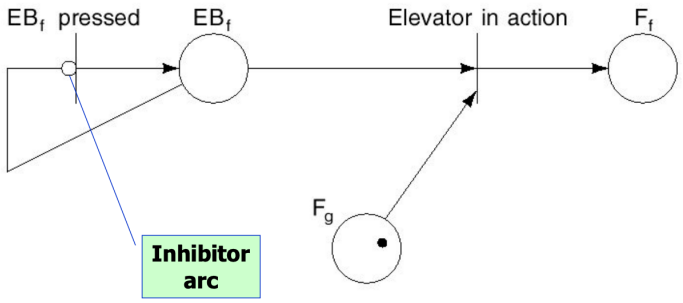


Questa rete di Petri che vediamo rappresentata in figura serve a specificare un requisito molto semplice, che ci dice come un ascensore deve comportarsi.

Il posto f_g rappresenta l'ascensore al piano terra (ground floor)
Il posto f_f rappresenta l'ascensore al primo piano (first floor)
Il posto chiamato EB_f rappresenta il pulsante dell'ascensore (elevator botton)



Dall'immagine possiamo notare che il token che sta all'intero di f_g ci dice che l'ascensore sta al piano terra. Mentre, il token che sta all'interno del posto EB_f ci dice che quel pulsante è stato premuto.
A questo punto voglio rappresentare il movimento dell'ascensore, questo lo faccio attraverso la transizione che si chiama "ascensore in movimento".
La transizione è abilitata quando esiste almeno un token che collega gli archi in ingresso. Quindi essendo abilitata può essere eseguita.



Possiamo notare che se nel posto chiamato EB_f, il quale rappresenta il pulsante dell'ascensore (elevator botton), non è presente il token. Ci serve un'altra transizione, che deve rappresentare la "pressione del pulsante", questa transizione è data da "EB_f pressed". Questa è una transizione con arco inibitore (con il pallino davanti), ovvero ha la funzione di inserire il token nel posto in input.

Nella rete di Petri esaminata abbiamo notato che ogni transizione rappresenta un'azione, l'esecuzione di queste azioni è istantanea, quindi non c'è un tempo di esecuzione della transizione. Per esempio, quando l'ascensore sta a piano terra l'ascensore si ritrova al primo piano, ovviamente questa cosa è assurda.
Questa limitazione è stata superata da dei "dialetti" delle Reti di Petri:

- introduciamo nome di *GSPN*, ovvero *Generalized Stochastic Petri Net*, ad ogni transizione viene associata un tempo con valor medio che ci permette di valutare e convalidare gli aspetti prestazionali del sistema.
- un ulteriore dialetto utilizzato è *CPN, Colored Petri Net*. Consiste in un insieme di colori che possono descrivere comportamenti diversi del sistema per diverse classi di utente.

Quindi il concetto di stato nelle Reti di Petri, si vede in modo indiretto, in base a come sono distribuiti i token nei vari posti.

- ***Macchine a stati Finiti (FSM)***

L'obiettivo di una macchina a stati finiti è uguale a quello delle Reti di Petri, ovvero è quello di rappresentare l'evoluzione di un sistema attraverso un insieme di stati che il sistema attraversa a partire da uno stato iniziale fino ad uno stato finale.

La differenza, rispetto alla Rete di Petri, è che la primitiva di base è quella di rappresentare direttamente un possibile stato del sistema.

In questo contesto, possiamo utilizzare 2 o 3 primitive, dove la primitiva principale è il rettangolo, che rappresenta lo stato. Gli archi orientati indicano la transizione tra uno stato e il successivo.

Possiamo rappresentare gli stati iniziali e finali con due tonalità di grigi diversi (come in foto), e li possiamo riconoscere pure perché gli stati iniziali non avranno nessuno stato in ingresso e gli stati finali non hanno nessun arco in uscita.

Il seguente esempio rappresenta il comportamento di una cassaforte.

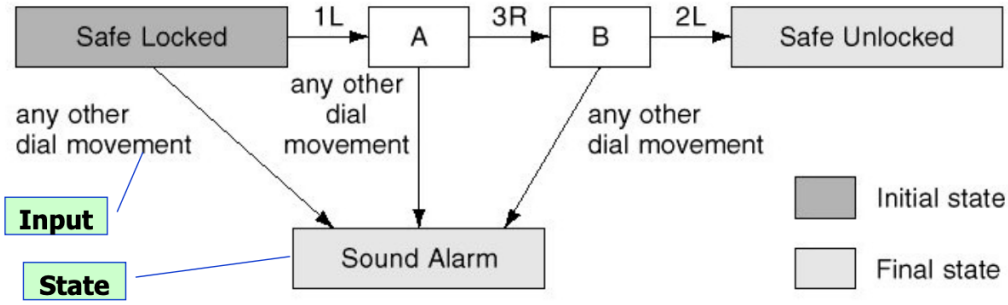


Table of Next States

Dial movement \ Current state	Safe locked	A	B
1L	A	Sound Alarm	Sound Alarm
1R	Sound Alarm	Sound Alarm	Sound Alarm
2L	Sound Alarm	Sound Alarm	Safe Unlocked
2R	Sound Alarm	Sound Alarm	Sound Alarm
3L	Sound Alarm	Sound Alarm	Sound Alarm
3R	Sound Alarm	B	Sound Alarm

N.B: Possiamo vedere nell'immagine la rappresentazione visuale, una tabella di stati successivi

Da questo esempio di macchina a stati finiti possiamo vedere:
Lo stato iniziale è Safe Locked (Stato Chiuso) e lo stato finale può essere Safe Unlocked (Stato Aperta) o Sound Alarm (Suono di Allarme)

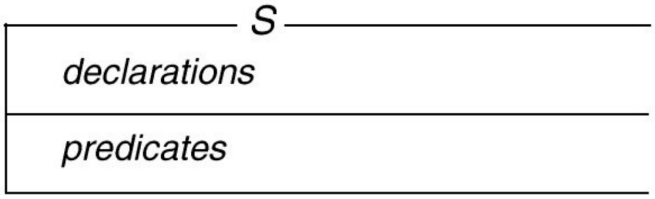
Tra lo Stato di Chiusura della cassaforte e lo Stato di Apertura della cassaforte ha una serie di stati intermedi che rappresentano le combinazioni. Ad esempio:

- lo stato 1L indica un giro verso sinistra
- lo stato 3R indica tre giri verso destra
- lo stato 2L indica un giro verso sinistra

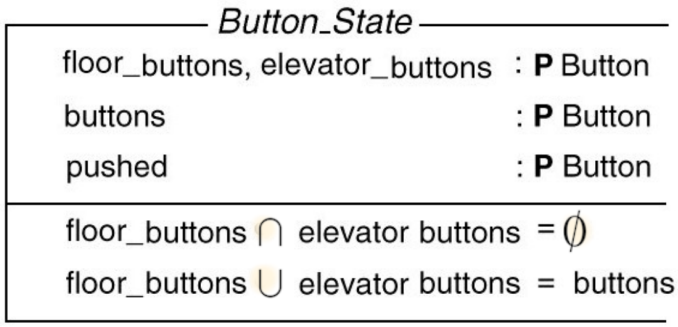
Quando il movimento è spagliato arriveremo allo stato finale Sound Alarm (Suono di Allarme).

Un altro esempio di notazione formale proposta appositamente per la specifica di prodotti software. Questa si chiama linguaggio Z.
Questo linguaggio mette a disposizione un'unica primitiva, è il concetto di schema.
Ogni schema ha:

- un nome
- un primo compartimento che contiene dichiarazioni di variabili
- un secondo dipartimento che contiene l'insieme di predicati che agiscono su quelle variabili



Un *esempio* di specifica di stato è il seguente:



Abstract Initial State

```
Button_init := [Button_State' | pushed' = ∅]
```

Questo è un esempio sempre relativo all'ascensore, nello specifico al pulsante dell'ascensore.

Qui vediamo che questo schema è formato:

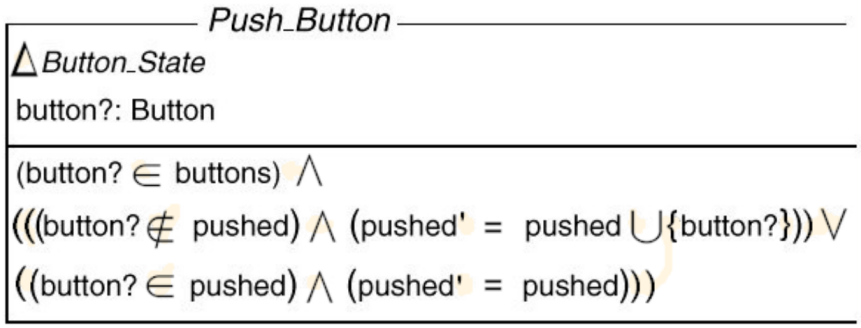
- dal nome → Button_State
- dal primo dipartimento di dichiarazioni di variabili → floor_buttons, elevator_buttons e così via. Per definire queste variabili le definiamo nel seguente modo:

  *nome variabile : **P** Button*

  dove *Button* è l'insieme di tutti i pulsanti e *P* indica l'insieme potenza

- un secondo dipartimento di predicati che agiscono su quelle variabili → vado a vedere come si comportano questi predicati definiti precedentemente. Faccio ciò usando i predicati e attraverso simboli di insiemistica indico le varie "azioni"

Un *esempio* di specifica di un operazione è il seguente:



Lo schema è formato nello stesso "formato" di uno schema per la specifica di uno stato. Possiamo notare:

- il nome che è dato da un'azione
- il Δ che per indicare su quali stati agisce e il *punto interrogativo ?:* specifica un parametro di input

Oltre alle specifiche formali, esistono delle *specifiche semi-formali*.

Il termine "modello del sistema" si riferisce a una rappresentazione formale o astratta del sistema, che aiuta a comprendere le sue proprietà e il funzionamento prima che venga effettivamente realizzato.

L'uso dei modelli nei sistemi software è integrato nei metodi di analisi dei requisiti, ovvero nella fase di specifica del software. Questi metodi fanno spesso uso di tecniche semi-formali per descrivere il sistema.

I metodi di analisi dei requisiti software si dividono principalmente in due categorie:

1. metodi di analisi strutturata, o procedurale;
2. *metodi di analisi orientata agli oggetti.*

Per avere una descrizione completa del sistema, è necessario creare diversi modelli che lo rappresentino da differenti prospettive. In particolare, si devono considerare le informazioni gestite dal sistema, le sue funzionalità e il suo comportamento dinamico.

Per descrivere la specifica *semi-formale* di un sistema software si usano 3 tipi di modelli:

- modello dei dati → rappresenta gli aspetti statici e strutturali relativi ai dati (data requirements)
    - ERD (*not UML*)

- Classdiagram (*UML*)

- modello comportamentale → rappresenta gli aspetti funzionali del sistema (functional requirements)

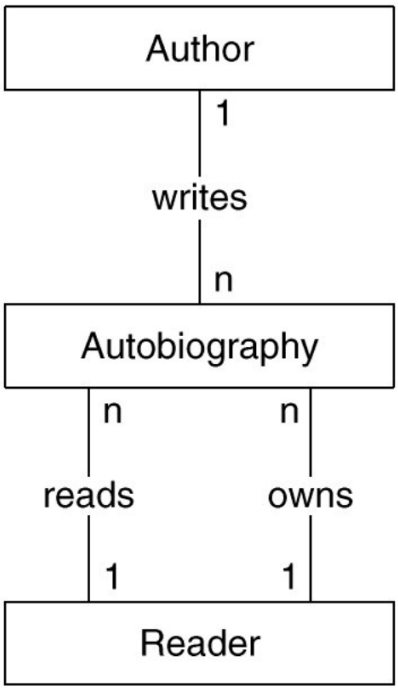
  - data flow diagram (*not UML*)
  - use case diagram (*UML*)
  - activity diagram (*UML*)
  - interaction diagram (*UML*)

- modello dinamico → rappresenta gli aspetti di "controllo“ e di come le funzioni del modello comportamentale modificano i dati introdotti nel modello dei dati

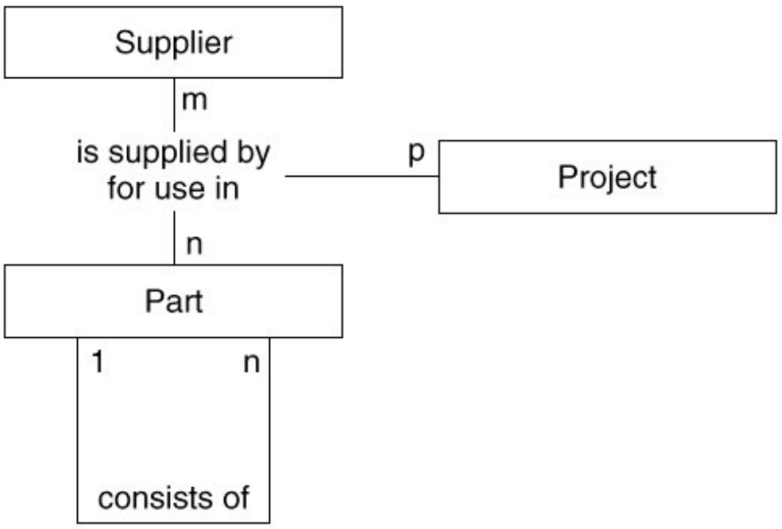
  - state diagram (*UML*)

Per quanto riguarda il modello ERD, ovvero Diagramma Entità Relazioni. Si usa per realizzare lo schema relazionale/logico di un database.

Nella figura affianco possiamo vedere un Diagramma Entità Relazioni, 1:N (uno a molti). Nel seguente esempio vediamo 3 informazioni che il sistema deve gestire che sono Autore, Autobiografia e Lettore. Le relazioni stabiliscono il numero di possibili istanze di un’entità che sono messe in corrispondenza con un’istanza dell’altra entità.



Nella figura affianco possiamo vedere un Diagramma Entità Relazioni, N:N (molti a molti).



Per quanto riguarda il modello DFS, ovvero Data Flow Diagram, serve a costruire modelli comportamentali. I costrutti di cui disponiamo in questo modello sono i seguenti quattro:

Segue un esempio di Software che elabora ordini. Possiamo notare un primo raffinamento (figura a destra) e un secondo raffinamento (figura in basso).