Domande VPC

Danilo Giannico

giugno 2022

Indice

1 Introduzione

1. Perché questo corso?

Solution:

- Fornire strumenti, teorici e pratici, necessari alla **verifica dei sistemi** (in particolare sistemi distribuiti)
- Obiettivo della verifica: **Affidabilità** dei sistemi
- Utilizzare linguaggi formali e strumenti software per la verifica di proprietà del sistema tramite verifica di proprietà del modello
- Perché? Sistemi malfunzionanti possono causare:
  - Perdita di dati
  - Perdita di soldi
  - Perdita di vite $->$ es. 2 incidenti Boeing 737 per MCAS: sistema software di controllo del volo che portava l'aereo in picchiata

2 Reti di Petri

1. Quali sono gli aspetti di un sistema distribuito che si modellano più facilmente con le Reti di Petri?

   Solution:

   - Per descrivere sistemi ad Eventi Discreti Dinamici (DEDS):
     - **Dynamic**: sistema descritto durante la sua evoluzione
     - **Event**: cosa causa un cambio di stato
     - **Discrete**: lo stato del sistema descritto da variabili discrete
   - Concorrenza (es. mutua esclusione)
   - Conflitto (preset transizioni hanno posti in comune)
   - Sincronizzazione (es. sistemi di produzione)

2. Mi parli delle reti di Petri

   Solution:

   - Inventate nel 1962 durante la tesi di dottorato del matematico Carl Adam Petri.
   - Formalismo per descrivere DEDS:
     - **Dynamic**: sistema descritto durante la sua evoluzione
     - **Event**: cosa causa un cambio di stato
     - **Discrete**: lo stato del sistema descritto da variabili discrete
   - Per specificare sistema ma anche le sue proprietà
   - Grafo bipartito, diretto, pesato, etichettato, marcato (token)
   - (tutto ciò che segue)

3. Valutare formalismo rispetto a questi attributi

Solution:

- *Formale*: sì
- *Intuitivo*: semplice da capire (visuale)
- *Succinto*: dipende dalla classe scelta e dal tipo di sistema
- *Efficace*: insieme di soluzioni molto ricco
- *Espressivo*: molto per la concorrenza, meno per es. tempo
- Può essere usato per generare *codice* iniziale: sì, reti WN

4. Perché linguaggio formale?

Solution: Unica interpretazione, non ambiguo

5. Definizione rete di Petri

Solution:

- Grafo bipartito, diretto, pesato, etichettato, marcato (token)
- Petri net + initial state = PN System
- Una rete di Petri N è una quadrupla (4-tuple):

$$N = (P, T, F, W)$$

  - P posti e T transizioni, finiti e non vuoti e $P \cap Q = \emptyset$
  - F flow relation: $F \subset P \times T \cup T \times P$
  - W weight function: $W : F \to \mathbb{N}^+$
- $G = (V, E)$ dove $V = P \cup T$ e $E = F$

6. Sintassi

Solution:

- PN in forma matriciale:

$$N = (P, T, Pre, Post)$$

  - **Pre**-function Pre: $P \times T \to \mathcal{N}$
    - Pre(p,t) = W(p,t) se $(p, t) \in F$
    - Pre(p,t) = 0 se $(p, t) \notin F$
      $-> Pre \in \mathcal{N}^{PxT}$
  - **Post**-function Post: $P \times T \to \mathcal{N}$
    - Post(p,t) = W(t,p) se $(t, p) \in F$
    - Post(p,t) = 0 se $(t, p) \notin F$
      $-> Post \in \mathcal{N}^{PxT}$

- Matrice di incidenza:

$$C = Post - Pre$$

- Posti: variabili di stato
- Transizioni: cambiamento di stato
- Marking: valutazione delle variabili di stato:
  - Funzione: $m : P \to \mathcal{N}$
  - Vettore nel numero dei posti (forma matriciale): $m \in \mathcal{N}^P$
- PN system:

$$S = (N, m0))$$

  dove
  - $N = (P, T, W, N)$ è una PN;
  - m0 è una marcatura (iniziale)
- Preset e postset:

$$\bullet t = Pre[-, t] = Pre[P, t]$$

  (vettore colonna della matrice Pre relativa a t)

$$t\bullet = Post[-, t] = Post[P, t]$$

  (vettore colonna della matrice Post relativa a t)

$$\bullet p = Pre[p, -] = Pre[p, T]$$

  (vettore riga della matrice Pre relativa a p)

$$p\bullet = Post[p, -] = Post[p, T]$$

  (vettore riga della matrice Post relativa a p)

7. Semantica:

Solution: Definita da come evolve lo stato (state equation) e da quando evolve lo stato (abilitazione)

- PN evolution: è dato dallo scatto (firing) delle transizioni. Una transizione può scattare se abilitata
- Abilitazione: $t \in T$ è abilitata in m se:

$$m \geq Pre[-, t]$$

$-$ > Disuguaglianza fra vettori, quindi elemento per elemento:

$$\forall p \in P : (p, t) \in F,\ m(p) \geq W(p, t)$$

$$\forall p \in \bullet t,\ m(p) \geq W(p, t)$$

$-$ > Marcatura non diventa mai negativa

- State equation: se una transizione è abilitata in m, può scattare e il suo scatto produce la marcatura m' tale che:

$$m' = m + C[-, t]$$

$$m' = m + Post[-, t] - Pre[-, t]$$

- Scatto: si scrive $m[t > m'$ oppure $m \xrightarrow{t} m'$. Ossia diciamo che m' è raggiungibile da m in un passo (attraverso t)
- Firing sequence: $\sigma = [t_1, .., t_k]$, con $t_i \in T$, è una firing sequence scattabile in m0, e scriviamo $m_0[\sigma > m_k$, iff $\exists$ un insieme di marcature $\{m_0, .., m_k\}$ tali per cui

$$\forall i \in [1..k],\ m_{i-1}[t_i > m_i$$

- Firing vector: $\overline{\sigma}$ è il vettore caratteristico (autovettore) della firing sequence σ

$$\overline{\sigma} : T \to \mathcal{N}$$

$$\overline{\sigma} \in \mathcal{N}^T$$

$-$ > codifica il numero di volte in cui le transizioni scattano in σ (è quindi un vettore finito nella dimensione di T) $-$ > si perde informazione

- State equation through firing sequence: Se σ + scattabile in m, allora il suo scatto produce una marcatura m' tale che:

$$m' = m + C \bullet \overline{\sigma}$$

- Raggiungibilità: Se esiste un firing vector $\overline{\sigma}$ tale che:

$$m' = m + C \bullet \overline{\sigma}$$

  Non implica che $\exists\ \sigma$ scattabile (m' potrebbe non essere una marcatura raggiungibile: soluzione spuria dell'equazione di stato) $->$ L'equazione di stato fornisce un insieme di equazioni lineari che caratterizzano un sovrainsieme dello spazio degli stati

- Negative reachability: attraverso equazione di stato possiamo dimostrare che uno stato non è raggiungibile, ossia quando l'equazione non ha soluzioni (implicazione solo in questo verso) $->$ per dimostrare correttezza es. mutua esclusione

8. Linguaggio di una PN

Solution: Insieme di tutte le firing sequence in m0:

$$\mathcal{L}(S) = \{\sigma(t1, .., tk) : \sigma\ is\ a\ firing\ sequence\ for\ S\ in\ m0\}$$

9. Semantica interleaving vs step

Solution:

Interleaving semantics:

- Il linguaggio delle firing sequence appena definito è detto linguaggio sotto la semantica interleaving
- Le transizioni scattano una alla volta

Step semantics:

- Le transizioni possono scattare in parallelo
- Enabling degree: Il grado di abilitazione di $t \in T$ nella marcatura m è:

$$e_t(m) = max\{k \in \mathcal{N}^+ :\ m > k * Pre[-, t]\}$$

  $->$ il numero di volte che una transizione può scattare in parallelo

- Step: **s** è un multinsieme (insieme in cui elementi possono apparire più volte) di transizioni ($s : T \to \mathcal{N}$)
- Def: uno step è abilitato in m se $m \geq Pre * \overline{s}$ (vettore caratteristico di s)
- Def: lo scatto di un step abilitato in m porta a:

$$m' = m + C * \overline{s}$$

10. RS, RG

Solution:

- **Reachability Set**: di un PN system $S = (N, m0)$, che indichiamo con $RS(S)$ o $RS(N, m0)$ è l'insieme di tutte le marcature raggiungibili da m0 attraverso una firing sequence di $\mathcal{L}(S)$:

$$RS(N, m0) = \{m : \exists \sigma \in \mathcal{L}(S) \mid m0[\sigma > m\}$$

- **Reachability Graph**: grafo diretto così definito:

$$RG_N(m0) = (V, E)$$

$$V = RS(N, m0)$$

$$(v1, v2) = (m1, m2) \in E \iff \exists t \in T : \ m1[t > m2$$

11. Proprietà di base

Solution:

- **Finito**: Un sistema è finito *iff* il RG è finito
- **Assenza di deadlock**: Non esiste uno stato raggiungibile in cui nessuna transizione è abilitata:

$$\nexists \ m \in RS(N, m0) : \forall t \in T \ m < Pre[-, t]$$

- **Vivo**: Per ogni transizione t, da ogni marcatura raggiungibile è possibile raggiungere una marcatura dove t è abilitata:

$$\forall t \in T, \forall m \in RS(N, m0), \ \exists m' \wedge \exists \sigma : m[\sigma > m' \wedge m' \geq Pre[-, t]$$

- **Reversibile**: Per ogni marcatura raggiungibile m esiste una firing sequence scattabile in m che porta alla marcatura iniziale

$$\forall m \in RS(N, m0)\ \exists \sigma : m[\sigma > m0$$

12. Classi di reti

Solution:

Sottoclassi (stesse regole di abilitazione e scatto ma vincoli strutturali: più semplici):

- Ordinarie: Tutti i pesi degli archi a 1

$$W : \ F \rightarrow \ \{1\}$$

- Macchina a stati: Ordinaria + transizioni 1 in 1 out + stato iniziale

$$|\bullet t| = |t \bullet| = 1$$

  $->$ non più sincronizzazione

- Marked graph: Ordinaria + posti 1 in 1 out

$$|\bullet p| = |p \bullet| = 1$$

  $->$ rappresenta molto bene sistemi di produzione (no scelta)

- Free choice: 2 transizioni o abilitate insieme o non abilitate

$$\bullet t \cap \bullet t' \neq \emptyset \Rightarrow \bullet t = \bullet t' \ (Pre[-, t] = Pre[-, t'])$$

  $->$ non determinismo

- 1-safe: tutti i posti hanno bound a 1 (sottoclasse non topologica, dipende da marcatura)

Sovraclassi(regole di abilitazione e/o scatto modificate):

- PN con archi inibitori:
  - Quintupla: $N = (P, T, Pre, Post, Inh)$
  - $Inh : \ P \times T \rightarrow \mathcal{N}^+ \cup \infty$

- Transizione t abilitata se:

$$m \geq Pre[-,t] \wedge m < Inh[-,t]$$

- Espressività:
  - Stessa capacità computazionale di Macchine di Turing (classe di complessità P)
  - Posso fare semplicemente dei "test a 0", ossia una transizione può scattare solo se non ci sono token in un determinato posto (es. lettori-scrittori)
  - C'è un chiaro vantaggio dal punto di vista modellistico
  - In alcuni casi si riescono a costruire reti PT "simili" $->$ usando posto complementare, ma con limitazione (es. n° di permessi per lettori)

- PN con priorità:
  - Quintupla: $N = (P, T, Pre, Post, Pri)$
  - $Pri : T \to \mathcal{N}$
  - Transizione t ha concessione in m se $m \geq Pre[-,t]$
  - Transizione t abilitata se ha concessione e $\forall t'$ con concessione in m $Pri(t) \geq Pri(t')$
  - Espressività:
    - Stessa capacità computazionale di Macchine di Turing (classe di complessità P)
    - Utile quando ci sono situazioni locali che devono tenere conto di una situazione globale (es. priorità forchetta fra tutti i filosofi)
    - Però da utilizzare con cautela
    - Si può passare (costruzione) da reti con priorità a reti con archi inibitori e viceversa (ma non banale)

- Reti colorate

13. Esempi da commentare (sui lucidi)

3 Reti WN

1. Vantaggi

Solution:

- **Modelling convenience**:
  - Rappresentazione più compatta
  - Rappresentazione più naturale dei dati nel modello $->$ mantenere identità token
  - I modelli sono più facilmente parametrizzabili
  - È possibile ripiegare (fold) sottoreti simili in un'unica, usando i colori per identificare token appartenenti alle varie sottoreti
- **Solution convenience**:
  - Se colori e funzioni di colori sono definite appropriatamente, è possibile sfruttare automaticamente simmetrie:
  - Riduzione spazio degli stati attraverso RG simbolico (SRG) $->$ usando classi di equivalenza di colori

2. Definizioni

Solution:

- Le reti di Petri colorate appartengono alla sovraclasse delle High-level Petri Net, che include diversi formalismi:
  - Colored PN (CPN): Colori possono essere qualsiasi insieme numerabile e ogni funzione su questo insieme è permessa per definire abilitazione e scatto delle transizioni
  - Predicate-transition net (reti algebriche)
  - Well-formed nets (WN): CPN con restrizioni sintattiche $->$ vantaggio computazionale a livello di soluzione (a livello di modellazione può essere meno semplice)

Colored PN

- C: insieme finito di classi di colore

- Marking: funzione da P a $Bag(C) ->$ associa ad ogni posto P un multinsieme di colori (quando si tratta di identità, usiamo un insieme)
- Archi: hanno una funzione associata dai colori delle transizioni ai colori dei posti
- Def: Una CPN è una sestupla (six-tuple):

$$N = (P, T, Pre, Post, C, cd)$$

  - P e T sono insiemi finiti, disgiunti e non vuoti
  - C è un insieme finito di classi di colori (es. $C = \{phil, forks\}$; $phil = phil1, phil2$; $forks = fork1, fork2$
  - $cd: P \cup T \to C$ è una funzione che definisce il color domain di ogni posto e di ogni transizione $->$ può essere una classe o un prodotto cartesiano di classi
  - Pre[p,t], Post[p,t]: $cd(t) \to Bag(cd(p)) ->$ in Pre e Post ci sono le funzioni dai colori delle transizioni ai colori dei posti che devono essere valutate per l'abilitazione e lo scatto
- Esempi:
  - $Pre[filosofi, prendi] = Id$ (Id è una funzione)
  - $Pre[forchette, prendi] = Fl + Fr$ (dove fl e fr sono funzioni)
  - $Pre[forchette, prendi](phil1) = fk2 + fk3$
- Abilitazione: Una istanza di transizione t per un certo colore c, $\langle t, c\rangle$, è abilitata in una certa marcatura quando il multinsieme dei token colorati in ciascun posto di input della transizione t contiene almeno tanti token di ogni colore $c' \in cd(p)$ quanti ne sono indicati dalla molteplicità di c' in Pre[p,t](c):

$$\langle t, c\rangle \; is \; enabled \; at \; m \iff m \geq Pre[-, t](c)$$

- Scatto: Lo scatto di una istanza di transizione abilitata $\langle t, c\rangle$ è un'operazione atomica che rimuove/aggiunge ad ogni posto il multinsieme di token ottenuto applicando la funzione associata agli archi:

$$m \overset{\langle t,c\rangle}{\to} m' \iff m' = m + C[-, t](c)$$

Reti WN

- Estensione colorata delle reti con archi inibitori e priorità (stessa potenza computazionale)

- Dominio di colore solo associato ai Posti, non più alle transizioni
- Transizioni possono essere parametrizzate attraverso un insieme di variabili tipate:
  - Variabili associate agli archi $->$ impongono colore delle transizioni in cui arrivano
  - Possibili valori delle variabili sono gli stessi del color domain del posto da cui parte l'arco
  - Un assegnamento dei valori delle variabili di una data transizione t identifica una istanza di t
- Sintassi:
  - Posti contengono token di tipo uguale al color domain del posto
  - Color domain può essere una classe di colore o un prodotto cartesiano di classi
  - Archi etichettati da funzioni dal color domain dei posti in input a $\mathcal{N}$ (quanti token di un certo colore prelevare)
  - Funzione: è una combinazione lineare di:
    - funzione **identità**: $\langle var \rangle -> \langle x \rangle(c) = c$ per ogni colore c
    - funzione **intero insieme**: S_{col} dove col indica una classe. Applicata all'insieme valuta l'intero insieme (anche questa funzione costante)
  - Esempio: se P ha color domain $ProcxMem$ allora la funzione identità sull'arco (in o out) avrà la coppia di variabili $\langle xp, xm \rangle$
  - Esempio: $col = \{c1, c2, c3\}$; se sull'arco da p a t c'è la funzione S_{col} allora t potrà scattare solo se ci sono almeno 3 token dei 3 distinti colori in p.
  - Esempio: la funzione $\langle x, S_{col} \rangle$ restituisce il cartesiano di x con tutti gli assegnamenti possibili in col
  - Guardia associata all'arco: vincolo sullo scatto (es. $[x! = y]$)
  - Scope delle variabili limitato all'intorno della transizione
- Abilitazione: t è abilitata se ci sono "abbastanza" token nei posti in ingresso:

$$m(p, c) \geq Pre[p, t](c)$$

  dove $pre[p, t]$ è la funzione sull'arco da p a t
- Sottoclassi statiche: es. class Proc=pr{1..2} **is** Veloci + q5..6 **is** Lenti $->$ usate nella marcatura iniziale e nelle guardie
- Classi circolari: es class Philo = circular p{1..N}

3. CRG e RG unfolded

Solution:

- **Colored RG**:
  - Estensione intuitiva alle marcature colorate della definizione di RG
  - Insieme di tutti gli stati possibili
  - Stato: distribuzione token colorati nei posti
- **Unfolded**: Per ogni WN posso generare una rete PT equivalente $->$ RG isomorfi

4. SRG

Solution:

- Quando abbiamo modello con comportamento simmetrico
- Possiamo aggregare stati che hanno comportamento simmetrico
- Esempio:

$$L0(1), C(), L1(l1)$$
$$L0(1), C(), L1(l2)$$

  Posso aggregarli nello stato generico:

$$L0(1), c(), L1(x1)$$

- "$x1$" è colore aggregato/variabile
- **Marking equivalence**: marcature sono equivalenti se hanno la stessa evoluzione futura:
  - Abilitano lo stesso insieme di transizioni
  - Le marcature raggiunte dallo scatto di transizioni corrispondenti sono a loro volta marking equivalent
    $->$ **bisimulazione**
- Posso costruire RG sugli stati aggregati $->$ SRG
- Vantaggio: $|SRG| \leq |RG| ->$ "risparmio" dipende dal livello di simmetria del sistema

Costruzione SRG:

- Costruirlo direttamente senza fare prima RG e poi bisimulazione
- Stato equivalente ad un altro stato se differiscono solo per una permutazione di colori
- Usare **permutazioni** per trovare stati equivalenti $->$ mapping fra colori
- Su permutazioni possiamo costruire relazioni di equivalenza sulla quale partizionare il RS in **classi di equivalenza**
- Transizioni simili accorpate
- Colori rimpiazzati da variabili che rappresentano insiemi di colori $->$ **sottoclassi dinamiche**
- Partiziono classi di colore: $\{z_i\}$ (sottoclassi dinamiche) sono una partizione della classe di colore $->$ Ogni assegnamento dei colori che rispetta la partizione porta ad una marcatura equivalente
- Accanto a marcatura, riporto cardinalità sottoclasse: es. $|Z1| = 2 -> $ posso assegnare a $z1$ tutti i possibili sottoinsiemi di 2 elementi della classe di colore corrispondente

5. Aggregazione e bisimulazione

Solution:

- Simili come concetto $->$ Marcature sono equivalenti se hanno la stessa evoluzione futura
- Simulazione basata sulle azioni però non guarda lo stato: es. posti diversi / 2 token di colori diversi o dello stesso colore $->$ bisimulazione li aggrega se bisimili, SRG no
- SRG distingue di più

4 Algebra dei processi

1. Introduzione

Solution:

- **Algebra**: Sia I un insieme non vuoto. Diciamo struttura algebrica su I l'insieme:
$$A = \{I, \alpha_1, ..., \alpha_k\}$$
Dove $\alpha_1, ..., \alpha_k$ sono operazioni interne in **I**, rispettivamente a $n_1, ..., n_k$ argomenti
- **Processi**: Nell'algebra dei processi, **I** è l'insieme dei processi
- Cos'è (la scoteca è?):
  - Descrizione astratta di sistemi concorrenti e non deterministici
  - Focus su azioni osservate più che su stati raggiunti
  - Interazioni tra processi indipendenti come comunicazione (no variabili condivise primitive, ma si possono simulare)
  - Specificare sistemi
  - Definiscono leggi algebriche per manipolare processi
  - Ne esistono varie
- **CCS**:
  - Calculus of Communicating Systems
  - 1980
  - Robin Milner
- **CSP**:
  - Calculus of Sequential Processes
  - 1978
  - Tony Hoare
- Process algebra ingredients:
  - $Act = \{a, b, c, ..\}$: Insieme di azioni
  - $\forall a \in Act,\ \exists \overline{a} \in Act$: azioni e co-azioni $->$ azioni osservabili
  - Azioni silenti: $\tau \in Act -> $ azioni non osservabili

2. Sintassi CCS

Solution:

- Processi che interagiscono fra di loro comunicando su azioni e co-azioni
- Agente/Processo = espressione che è sintatticamente riconoscibile come un termine dell'algebra
- Grammatica:

$$E ::= (E) \mid a.E \mid E + E \mid E||E \mid E\backslash L \mid E[f] \mid 0 \tag{1}$$

- a, b, c: Azioni
- $\overline{a}, \overline{b}, \overline{c}$: Co-azioni
- τ: Azione silente
- A, B, C Agenti
- (E) : Ordine di valutazione
- $a.E$: **Prefisso**. Eseguo a, poi mi comporto come E $->$ Operatore di sequenza
- $E + E$: **Scelta non deterministica**
- $E||E$: **Composizione parallela**
- $E\backslash L$: **Restrizione** su L $->$ tutte le azioni in L non eseguibili
- $E[f]$: **Rietichettatura**. E rietichettato attraverso la funzione di mapping f
- 0 : nil $->$ Termino
- Convenzioni:
  - "." ha priorità più alta di "+"
  - ".0" è omesso

3. Semantica CCS

Solution:

- Approccio semantico: Structural Operational Semantics (SOS) $->$ Strutturata in base agli operatori

- Assioma $->$ non ha precondizioni
- Insieme di proof rule
- Definiscono come evolvono gli agenti/termini

  - *a*.p –a→ p
    - (a.p evolves with a in p -- **axiom**)
  - p—*a*→p′ |-- p+q –*a*→ p′
    - (given that p evolves with a in p′, then p+q evolves with a in p′) – **proof rule**
  - q—*a*→q′ |-- p+q –*a*→ q′
  - p—*a*→p′ |-- p||q –*a*→ p′||q
  - q—*a*→q′ |-- p||q –*a*→ p||q′
  - p—*a*→p′, q—*a*→q′ |-- p||q –τ→ p′||q′
  - p—*a*→p′ , *a* ∉ R |-- p\R –*a*→ p′\R
  - p—a→p′ |-- p[m]—m(*a*)→p′[m]

- Composizione parallela:
  - Semantica **interleaving**
  - **Sincronizzazione**
- Relabeling: $m : Act \to Act$ $->$ funzione applicata alle azioni, sui termini solo indicata ([m])

4. Conseguenze della restrizione

Solution:

- Riduce le scelte possibili
- Se applicata a un parallelo, sarà possibile solo la sincronizzazione e non più l'interleaving

5. Derivation Graph

Solution:

- **Derivativi**: I derivativi $D(E)$ di un agente E sono gli agenti (quindi espressioni) derivati da E applicando le proof rule o gli assiomi
  $->$ Stati
- **Derivation Graph**: Grafo (N, A) dove $N = D(E)$ e $(n1, n2) \in A$ se $n2$ può essere derivato da $n1$ applicando una proof rule o l'assioma

6. Differenza buffer

Solution:

- Buffer a n posizioni:
  - Unica scelta di *put* o *get*
  - Ma riscrivere nuovi processi per aggiungere posizioni (stati intermedi)
- Composizione parallela di n buffer a 1 posizione:
  - Riutilizzo processo
  - Poter scegliere che posizione usare
  - Ma troppe scelte
- Composizione parallela con etichettatura:
  - Posizionamento degli elementi all'interno del buffer "automatico" attraverso azioni di sincronizzazione τ
  - L'unico che offrirà una *put* sarà il primo buffer libero di sinistra, mentre l'unico che offrirà una *get* sarà l'ultimo buffer pieno di destra
  - Vantaggi di entrambi
  - Stesso comportamento osservabile di buffer a n posizioni

7. Rapporto con Reti di Petri

Solution:

- **Reti di Petri**:
  - Fortemente basato su elementi grafici

  - Ricco insieme di tecniche risolutive
  - Stati e eventi hanno la stessa impostanza
- **Algebre dei processi**:
  - Fortemente basate su come l'osservatore "vede" gli eventi di un sistema
  - Eventi sono più rilevanti degli stati
  - Nozioni di equivalenza
- **Rete di Petri equivalente**:
  - Possiamo costruire rete di Petri equivalente
  - Ossia per cui il Derivation Graph e il Reachability Graph sono isomorfi

8. Sintassi CSP

Solution:

- No azioni e co-azioni
- No restrizione
- Insieme di sincronizzazione
- Grammatica:

$$\boxed{P ::= Nil \mid a.P \mid P + P \mid P||_S P \mid P/L \mid A} \tag{2}$$

- P/L : **Rietichettatura**
- S : **Insieme di sincronizzazione**

9. Semantica CSP

Solution:

- Stesso assioma CCS
- Stesse proof rule
- Tranne per Composizione parallela:

$$P ::= Nil \mid a.P \mid P + P \mid P \|_S P \mid P/L \mid A$$

Parallel Composition

$$\frac{P \xrightarrow{a} P'}{P \|_S Q \xrightarrow{a} P' \|_S Q} \; a \notin S \qquad \frac{Q \xrightarrow{a} Q'}{P \|_S Q \xrightarrow{a} P \|_S Q'} \; a \notin S$$

$$\frac{P \xrightarrow{a} P' \;\; Q \xrightarrow{a} Q'}{P \|_S Q \xrightarrow{a} P' \|_S Q'} \; a \in S$$

10. Differenza CCS-CSP

Solution:

CCS:

- Processi che interagiscono fra di loro comunicando su azioni e co-azioni
- Restrizione
- Sincronizzazione solo tra 2 processi alla volta (azione e co-azione)
- Sincronizzazione e interleaving insieme
- Sincronizzazione attraverso azione non visibile τ
- Parentesi non contano (sincronizzazione sempre fra azione e coazione)

CSP:

- No azioni e co-azioni
- No restrizione
- Insieme di sincronizzazione
- Sincronizzazione a 3 o più processi
- No sincronizzazione e interleaving insieme
- Sincronizzazione visibile dall'esterno (non con τ)
- Parentesi contano

5 Analisi

1. Elenco diverse tecniche di analisi

Solution: Tecniche:

- **Enumerative**:
  - Analisi dello spazio degli stati: derivation/reachability graph
  - Verifica di proprietà di base
- **Trasformazione**: analisi per riduzione
  - Insieme di regole di riduzione strutturale per PN
  - Leggi equazionali per PA (es. A+A=A)
- **Analisi strutturale sulla matrice di incidenza**: solo per PN $->$ P e T invarianti
- **Equivalenze**: definite per PA, estese alle PN
- **Enumerative: Model checking** sullo spazio degli stati per proprietà di logica temporale

2. Definizioni di proprietà di base

Solution:

- **Assenza di deadlock**: Non esiste uno stato raggiungibile in cui nessuna transizione è abilitata:

$$\nexists\ m \in RS(N, m0) : \forall t \in T\ m < Pre[-, t]$$

- **Boundedness**:
  - Bound di un posto p in S: massimo numero di token raggiungibili in quel sistema:

$$b(p) = max\{m(p) :\ m \in RS(N, m0)\}$$

  - Posto: Un posto è *bounded* se:

$$b(p) < \infty$$

- Sistema: Un sistema è *bounded/limitato* se tutti i posti sono bounded:

$$\forall p \in P,\ b(p) < \infty$$

- Proprietà: S è bounded *iff* il suo RS è finito
- Structurally bounded: N è strutturalmente limitata se $\forall\ finite\ m0$, $(N, m0)$ è bounded

- **Liveness**:
  - Transizione: Una transizione t è *viva* se può scattare "infinitamente spesso" $->$ per ogni marcatura raggiungibile m, esiste una firing sequence scattabile in m che porta in una marcatura m' in cui t è scattabile:

$$\forall m \in RS(N, m0),\ \exists m' \wedge \exists \sigma : m[\sigma > m' \wedge m' \geq Pre[-, t]$$

  - Sistema: Un sistema è *vivo* se tutte le transizioni sono vive $->$ per ogni transizione t, da ogni marcatura raggiungibile m è possibile raggiungere una marcatura m' dove t è abilitata:

$$\forall t \in T, \forall m \in RS(N, m0),\ \exists m' \wedge \exists \sigma : m[\sigma > m' \wedge m' \geq Pre[-, t]$$

  - Proprietà: Un sistema con un RG finito che ha un'unica componente fortemente connessa, in cui ogni transizione appare almeno una volta, è vivo
  - Structurally live: N è strutturalmente viva se $\exists\ finite\ m0$: $(N, m0)$ è viva

- **Reversability**:
  - **Home state**: m è un *home state* se per ogni marcatura raggiungibile m' esiste una firing sequence scattabile in m' che porta a m

$$\forall m' \in RS(N, m0)\ \exists \sigma : m'[\sigma > m$$

  - **Reversability**: Un sistema è *reversibile* se m0 è un home state $->$ per ogni marcatura raggiungibile m esiste una firing sequence scattabile in m che porta alla marcatura iniziale:

$$\forall m \in RS(N, m0)\ \exists \sigma : m[\sigma > m0$$

  $->$ Boundedness, Liveness e Reversability sono indipendenti

- **Marking mutual exclusion**: pi e pj sono in marking mutual exclusion se non sono mai marcati insieme:

$$\nexists m \in RS(N, m0) :\ m(pi) > 0\ \wedge\ m(pj) > 0$$

- **Firing mutual exclusion**: ti e tj sono in firing mutual exclusion se non possono mai scattare insieme:

$$\nexists m \in RS(N, m0) :\ m \geq Pre[-, ti] + Pre[-, tj]$$

3. Differenza marking / liveness invariance

Solution:

- **Marking invariance**: Una proprietà di una singola marcatura che deve essere verificata su tutte le marcature

  $\phi(m)$ è una marking invariant property se:

  $$\forall m \in RS(N, m0),\ \phi(m)\ is\ true$$

  Esempi:

  - K-boundedness di un posto p:

    $$\forall p \in RS(S),\ b(p) \leq k$$

  - Marking mutual exclusion tra p e p':

    $$\forall m \in RS(S),\ m(p) = 0 \vee m(p') = 0$$

  - Assenza di deadlock:

    $$\forall m \in RS(S),\ \exists t \in T :\ m \geq Pre[-, t]$$

- **Liveness invariance**: Per ogni marcatura raggiungibile esiste almeno una marcatura raggiungibile da essa che soddisfa la proprietà

  Π è una liveness invariant property se:

  $$\forall m \in RS(N, m0), \exists m' \in RS(n, m) :\ m'\ satisfies\ \Pi$$

  Esempi:

  - Liveness di t:

    $$\forall m \in RS(N, m0), \exists m' \in RS(n, m) :\ m' \geq Pre[-, t]$$

  - m_H è un home state:

    $$\forall m \in RS(N, m0), \exists m' \in RS(n, m) :\ m' = m_H$$

  - Reversability:

    $$\forall m \in RS(N, m0), \exists m' \in RS(n, m) :\ m' = m0$$

4. Tecniche enumerative:

Solution:

- **Problema**:
  - Esplosione spazio degli stati
  - Spazio degli stati infinito
  - Decidibilità: es. dati 2 sistemi decidere se i loro RG sono uguali
- **Coverability Graph**
  - Grafo finito per riconoscere che rete non è limitata
  - Quindi boundedness è decidibile per reti PT
  - Copertura fra marcature: m' copre m ($m \leq m'$) se:

$$\forall p \in P,\ m'(p) \geq m(p)$$

  - Se trovo un cammino che da m mi porta ad una sua copertura m' in cui almeno un posto è strettamente maggiore, allora la rete è unbounded $->$ perché in quella marcatura sarà nuovamente scattabile la stessa firing sequence quindi il cammino è ripetibile all'infinito
- **Algoritmo calcolo RG**:
  - Input: $(N, m0)$
  - Output: RG
    1. Inizialmente in RG solo la marcatura iniziale
    2. Per ogni m non taggato (taggo i nodi già visitati):
    3. Per ogni transizione abilitata in m:
       (a) Calcolo m'
       (b) Se esiste un m'' (già trovato) che viene strettamente ricoperto: ossia mi porta in m' e m' copre m'' (e per almeno un posto è maggiore stretto), allora l'algoritmo fallisce e termina $->$ Sistema unbounded
       (c) Se non esiste e m' non è stato già trovato, lo aggiungo ai nodi e aggiungo come arco fra m e m' la transizione t
  - Complessità: esponenziale in P,T e $m0$ $->$ spesso il controllo sull'unbounded non viene fatto
  - Calcolo Coverability tree: stesso algoritmo ma quando trovo post unbounded lo etichetto

- **Algoritmo per Marking invariance**:
  - Input: $RS(N, m0)$ e proprietà Π
  - Output: TRUE o FALSE
    1. Per ogni m non taggato (taggo i nodi già visitati):
    2. Se m non soddisfa Π, return FALSE
    3. Se finisco i nodi, return TRUE
  - **Complessità**: lineare in $|RS|$, quindi esponenziale in P, per il costo di verifica di Π
- **Algoritmo per Liveness invariance**:
  - Ridurre il problema alle componenti fortemente connesse terminali (**BSCC**)
  - Reversability $->$ Verificare che sistema abbia un'unica SCC
  - Input: $RG(N, m0)$ e proprietà Π
  - Output: TRUE o FALSE
    1. Decompongo RG in componenti fortemente connesse
    2. Shrinking $->$ Componenti fortemente connesse diventano vertici del grafo ridotto
    3. Calcolo l'insieme delle BSCC
    4. Per ogni componente fortemente connessa terminale, se non contiene un m che soddisfa Π, return FALSE
    5. Ritorna TRUE
  - **Complessità**: richiede costruzione di SCC $->$ $O(|V| + |E|)$ più il costo della verifica di Π in ogni BSCC

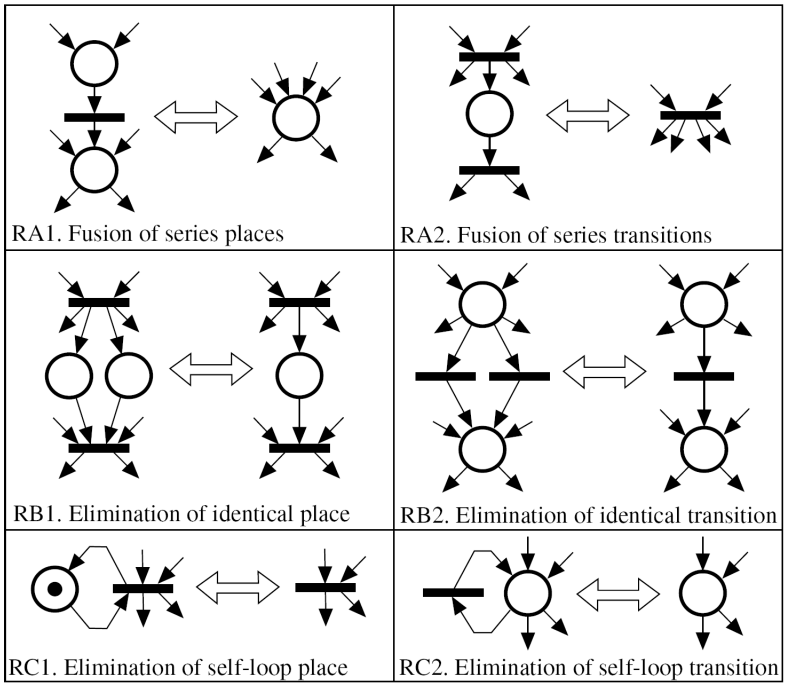
5. Trasformazioni:

Solution:

- Lista di regole con precondizioni strutturali o comportamentali (dipende da marcatura)
- Riduzione è property preserving:
  - Liveness: sì
  - Boundedness: sì ma non k-boundedness
  - Esistenza di home states ma non gli stessi (possono essere rimossi/uniti)

- Reversibilità: no (perché elimino/unisco posti)
  $->$ Però perdiamo informazioni

- La rete ottenuta è "più facile" da analizzare $->$ es. RG più piccolo, sottoclasse
- Problemi riscrittura sistemi:
  - Completezza: riesco a generare tutte le reti ridotte esistenti?
  - Confluenza: se applico le regole in ordine diverso, ottengo la stessa rete?
- **Regole**:



6. Analisi strutturale su matrice di incidenza:

Solution:

- **State equation**:
  - L'idea dietro questo tipo di analisi è quella di usare la State Equation per verificare delle proprietà
  - Esempio: marking mutual exclusion fra p e p' può essere ridotto all'**assenza** di soluzioni per:

$$\{m = m0 + C * \sigma \wedge m[p] > 0 \wedge m[p'] > 0\}$$

- Se invece trovo soluzione, non vuol dire che m appartenga a RS
- Procedura semi-decidibile perché

$$RS(S) \subset LRS^{SE}(S)$$

(Linearized RS in accordo con State Equation)

- **P e T semiflussi**:
  - Def: Un **p-flow** è un vettore nella dimensione dei posti y:

$$P \to \mathcal{Q} : \ y \cdot C = 0$$

$-> $ è un annullatore sinistro della matrice di incidenza
  - Def: Un **t-flow** è un vettore nella dimensione delle transizioni x:

$$T \to \mathcal{Q} : \ C \cdot x = 0$$

$-> $ è un annullatore destro della matrice di incidenza
  - Def: un p-flow non-negativo è un **p-semiflow**
  - Def un t-flow non negativo è un **t-semiflow**
- **Definizioni**:
  - Def: Il **supporto** $||y||$ di un p-semiflusso è:

$$||y|| = \{p \in P : \ y[p] \neq 0\}$$

  - Def: Una rete è **conservativa** se esiste almeno un p-semiflusso y in cui $||y|| = P -> $ tutti i posti compaiono in almeno un p-semiflusso
  - Def: Una rete è **consistente** se esiste almeno un t-semiflusso x in cui $||x|| = T -> $ tutte le transizioni compaiono in almeno un t-semiflusso
  - Def: Un p-semiflusso è **canonico** se il massimo comun divisore dei suoi elementi non nulli è 1
  - Def: Un insieme **generatore** di p-semiflussi $\Psi = \{y_1, .., y_n\}$ è l'insieme del numero minimo di p-semiflussi (p-semiflussi **minimali**) tali per cui riesco a generare tutti gli altri per combinazione lineare
  - Proprietà: Un semiflusso è minimale *iff* è canonico e il suo supporto non contiene strettamente il supporto di ogni altro p-semiflusso. Inoltre l'insieme di semiflussi minimali di una rete è sempre finito e unico, ma il loro calcolo può essere esponenziale nella dimensione di C
- **Token conservation law**:

- P-invariante: da p-semiflussi

$$y \in \mathcal{N}^P, y \cdot C = 0 \Rightarrow \forall m0, \forall m \in RS(N, m0)\ y \cdot m = y \cdot m0$$

  - La somma pesata dei token nei posti coperti è costante per ogni marcatura raggiungibile $->$ **token count**
  - **Boundedness**: m è finito $->$ tutti i posti coperti da almeno un p-semiflusso sono limitati $->$ Se la rete è conservativa allora è bounded
  - **Negative reachability**: se m non rispetta l'equazione, allora non è raggiungibile

- **Cyclic behaviour law**:
  - T-invariante: da t-semiflusso

$$x \in \mathcal{N}^T, C \cdot x \Rightarrow \exists m0, \exists \sigma \in \mathcal{L}(N, m0) :\ m0 \xrightarrow{\sigma} m0\ and\ \overline{\sigma} = x$$

  - Esiste una firing sequence che ripristina la marcatura iniziale
  - Il firing vector di questa sequenza è il t-semiflusso x

- **Applicazioni**: ricavare proprietà della rete sfruttando i p/t invarianti
  - Bound dei posti $->$ p-semiflussi
  - Marking mutual exclusion $->$ p-semiflussi
  - Assenza di deadlock $->$ per assurdo usando p-semiflussi
  - Liveness

- **Sovraclassi**: Sia N una rete con archi inibitori/priorità e N' la rete corrispondente PT senza archi inibitori/priorità, allora

$$RG(N, m) \subseteq RG(N', m)$$

  - Safety properties: se vere su N' vere anche su N
  - Liveness properties: se vere su N' potrebbero non esserlo su N

- **Sottoclassi**: Tecniche di analisi più potenti
  - State machine: singolo p-semiflusso di peso 1
  - Marked graph: tutti i circuiti elementari della rete sono p-semiflussi

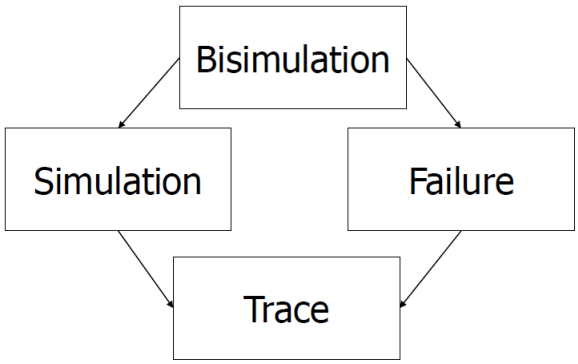
- **Free choice - Teoremi**:
  - **Commoner's Theorem**: Un sistema free choice è vivo *iff* tutti i *sifoni* contengono *trappole* marcate nella marcatura iniziale:

    - Sifoni: $P' \subseteq P : \bullet P \subseteq P\bullet -> $ tendono a svuotarsi
    - Trap: $P' \subseteq P : \ P\bullet \subseteq \bullet P -> $ tendono a riempirsi (intrappolano token)
  - **Rank Theorem**: Se N è una rete free choice, (N,m0) è <u>vivo</u> e <u>bounded</u> *iff*:
    - N è fortemente connessa and
    - N è coperta da macchine a stato (da p-semiflussi)
    - $Rank(C) = |\Phi| - 1$
    - Tutti i sifoni di N sono inizialmente marcati

7. Equivalenze:

Solution:

- **Gerarchia**:



  Dove un arco da $\approx_1$ a $\approx_2$ ($\approx_1$ più raffinato di $\approx_2$) significa:

$$P \approx_1 Q \Rightarrow P \approx_2 Q$$

- **Tracce**:
  - <u>Def</u>: l'insieme T(E) delle tracce dell'agente E è l'insieme di tutte le sequenze finite che possono essere prodotte dall'evoluzione di E
  - <u>Def</u>: se E e F sono agenti, diciamo che E è equivalente a F rispetto alle tracce:

$$E \approx_{tr} F \iff T(E) = T(F)$$

  - Per agenti con un numero finito di stati, l'equivalenza su tracce finite implica l'equivalenza su quelle infinite

- **Fallimento**: Un fallimento di un agente E è una coppia (σ, X):
  - $\sigma \in T(E)$
  - $X \in Act$
  - $\exists F : E \xrightarrow{\sigma} F$ e nessuna delle azioni in X è possibile in F

$$E \approx_{fl} F \iff Fail(E) = Fail(F)$$
$$E \approx_{fl} F \Rightarrow E \approx_{tr} F$$

- **Simulazione**: dati S derivativi di E e F, $R \subseteq S \times S$ è una relazione di simulazione se:
  - $E\ R\ F$
  - Se $E'\ R\ F'$ e $E' \xrightarrow{a} E'$, allora esiste un $F'' : F' \xrightarrow{a} F''$
  - Diciamo che **F simula E**

$$E \approx_{sim} F \iff E\ R\ F \wedge F\ Q\ E$$
$$E \approx_{sim} F \Rightarrow E \approx_{tr} F$$

- **Bisimulazione**:
  - Come simulazione ma con $R = Q$
  - **Algoritmo**: Partizione iniziale in cui ci sono tutti e poi divido(split) in base ad azioni che possono fare e se con quell'azione arrivano nella stessa partizione
  - **State based bisimulation**: Quando le azioni non sono distinguibili, come prima ma split in base alle proprietà degli stati
  - **Weak bisimulation**: Quando non considero le azioni non visibili τ intermedie:

$$m \overset{a}{\Rightarrow} m' \ when\ m \xrightarrow{a} m_{int} \xrightarrow{\tau} ... \xrightarrow{\tau} m'$$

6 LTL

1. Introduzione

Solution:

Definizioni

- **Esecuzione**: Una sequenza finita o infinita di stati
- **Condizione iniziale**: Lo stato iniziale soddisfa la condizione iniziale: $I(s0)$
- **Transizione**: Ci si muove da uno stato s_i a uno stato s_{i+1} eseguendo una transizione $e \rightarrow t$:
  - $e(s_i)$ ossia s_i soddisfa e (Condizione di abilitazione)
  - s_{i+1} è ottenuto applicando t a s_i
- Assumiamo che tutte le sequenze siano **infinite** estendendo quelle finite attraverso lo **stuttering** dell'ultimo stato (il suo successore è sé stesso)
- **Transition system**:
  - Un insieme finito di variabili **V**
  - Un insieme di stati Σ
  - Un insieme finito di transizioni **T**
  - Una condizione iniziale I
  - Denotiamo con $R(s, s')$ il fatto che s' sia un successore di s
- **LTL**:
  - **Linear Temporal Logic**
  - Pnueli, **1977**
  - Logica per descrivere sistemi in termini di esecuzioni lineari
  - Interpretazione sulla singola esecuzione, e poi su tutte (prese singolarmente)

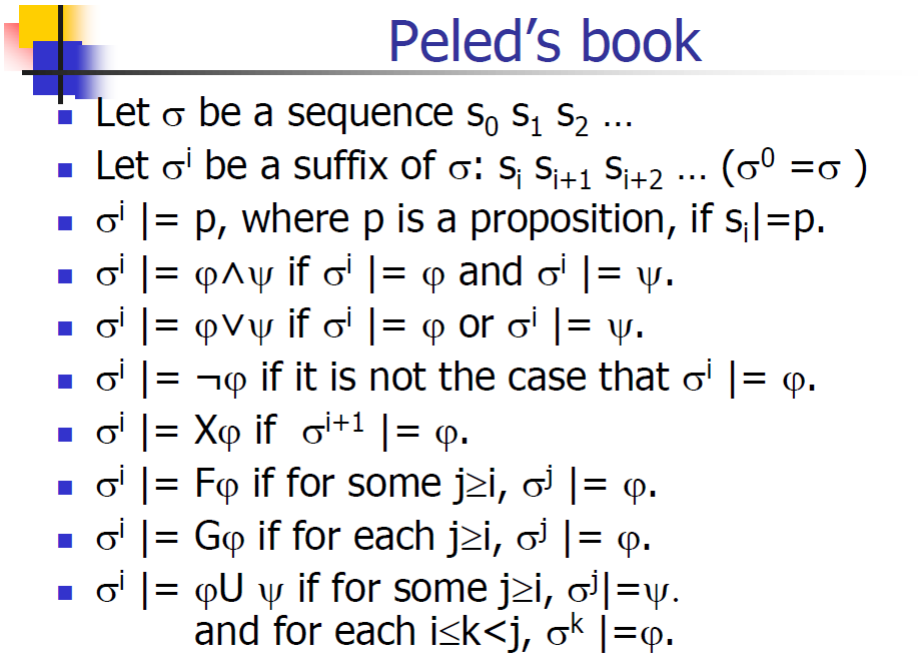
2. Sintassi

Solution:

$$\phi = \ p \mid (\phi) \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid G\phi \mid F\phi \mid X\phi \mid \phi U \phi \tag{3}$$

Solution:

- **Modello lineare**: Le formule LTL sono interpretate su un modello lineare: sequenze infinite su un insieme di stati S
- **Peled's book**:
  - Semantica definita su **sequenze** σ utilizzando **suffissi**
  - Definiamo **relazione di soddisfazione** $\models$ come: $(\sigma, \phi) \in \models$, e scriviamo $\sigma \models \phi$



- **Katoen's book**:
  - Semantica definita su **stati**
  - Formule interpretate su modello lineare usando il **modello lineare di Kripke**:
    $$M(S, R, L)$$
  - **S** è l'insieme degli **stati**
  - **R** è la funzione **successore** $R : S \to S$ che assegna ad ogni stato il suo unico successore $R(S)$

- **L** è la funzione di **etichettatura** $L : S \to 2^{AP}$ dove AP è l'insieme delle **proposizioni atomiche**
- Dato un modello M e una formula ϕ, scriveremo $(M, s) \models \phi - >$ Dato che ogni stato ha un solo successore, la coppia identifica univocamente una sequenza

Katoen's book

Let $R^0(s) = s$ and $R^{n+1}(s) = R(R^n(s))$, for any $n > 0$

- s |= p, where p a proposition, if p ∈ L(s).
- s |= φ∧ψ if s |= φ and s |= ψ.
- s |= φ∨ψ if s |= φ or s |= ψ.
- s |= ¬φ if ¬(s |= φ).
- s |= Fφ if ∃ j≥0: R^j(s) |= φ.
- s |= Xφ if R(s) |= φ.
- s |= Gφ if for each j≥0, R^j(s) |= φ.
- s |= φU ψ if for some j≥0, R^j(s)|=ψ.
  and for each 0≤k<j, R^k(s) |=φ.

4. Model checking:

Solution:

- **Model checking problem**: Dato un modello finito M, uno stato s e una proprietà ϕ, abbiamo che $s \models \phi$?
- **Satisfiability**: Data una formula ϕ, esiste un modello M e uno stato s tali che $(M, s) \models \phi$?
- **Validity**: Data una formula ϕ, abbiamo per tutti i modelli M e per tutti gli stati che $(M, s) \models \phi$?
- **Automa di Buchi**:
  - Il linguaggio delle formule LTL può essere rappresentato usando un automa di Buchi

- Dato $F = \{F_i\}$ un insieme di insiemi accettanti
- $F_i = \{f_1, f_2, .., f_n\}$, dove f_i sono stati accettanti
- **Condizione di accettazione**: esecuzione deve passare **infinitamente spesso** da almeno uno stato accettante per ogni insieme f_i

- **Approccio**:
  - Verificare che
    $$L(Model) \subseteq L(spec)$$
    (linguaggio del modello sottoinsieme linguaggio specifica) $->$ Ossia che tutte le esecuzioni del modello siano un sottoinsieme di tutte le sequenze che soddisfano la formula
  - Equivalente a verificare:
    $$L(Model) \cap \overline{L(spec)} = \emptyset$$
  - Specifichiamo il modello come **automa di Buchi** (tutti gli stati sono accettanti)
  - Specifichiamo il **negato** della formula come **automa di Buchi**
  - Ne facciamo l'**intersezione**
  - Se intersezione **vuota**, allora formula vera, altrimenti abbiamo un controesempio

- **Complessità**:
  $$O(|Sys| * 2^{|\phi|})$$
  dove Sys è la dimensione del sistema e $2^{|\phi|}$ è la dimensione dell'automa della formula

5. Fairness:

Solution:

- **2** soluzioni per considerare solo esecuzioni fair:
  - Modificare **algoritmo** MC per considerare solo esecuzioni fair
  - Forzare la fairness nella **formula**: invece di verificare ϕ verifico
    $$fair - constraint \Rightarrow \phi$$

- **Fairness constraint**:

– **Unconditional fairness**:

$$GF\phi$$

– **Weak fairness**:

$$FG\phi \Rightarrow GF\psi$$

– **Strong fairness**:

$$GF\phi \Rightarrow GF\psi$$

7 CTL

1. Introduzione

Solution:

- **Computational Tree Logic**
- Clarke & Emerson, **1980**
- La nozione lineare di tempo viene sostituita da una nozione **branching** $- >$ Ogni stato ha **più successori**: ogni stato ha **più futuri possibili**
- In CTL posso esprimere **possibilità**

2. Sintassi

Solution:

$$\begin{array}{l} State\ formula: \ \phi = \ p \mid (\phi) \mid \neg\phi \mid \phi \vee \phi \mid E\phi \mid A\phi \\ Path\ formula: \ \psi = \ X\phi \mid \phi U \phi \end{array} \tag{4}$$

3. Semantica

Solution:

- Formule interpretate su **modello di Kripke**:

$$M(S, R, L)$$

- **S** è l'insieme degli **stati**
- **R** è la funzione **successore** $R : S \to 2^S$ che assegna ad ogni stato l'insieme dei suoi successori $R(S)$
- **L** è la funzione di **etichettatura** $L : S \to 2^{AP}$ dove AP è l'insieme delle **proposizioni atomiche**
- Dato un modello M e una formula ϕ, scriveremo $(M, s) \models \phi$
- Def: Un **cammino** è un'infinita sequenza di stati tali che $(s^i, s^{i+1} \in R$

- Def: $\mathcal{P}_M(s)$ è l'insieme di tutti i cammini che originano in s:

$$\mathcal{P}_M(s) = \{\sigma \in S^\omega : \sigma[0] = s\}$$

- s è un **p-state** se $p \in L(s)$
- σ è un **p-path** se consiste solamente in p-states
- Semantica:

Given a Kripke structure M , $p \in AP$

- s |= p iff p ∈ L(s). ~ s è un p-state
- s |= ¬φ iff ¬(s |= φ).
- s |= φ∨ψ iff s |= φ ∨ s |= ψ.
- s |= EXφ iff ∃σ∈$\mathcal{P}_M$(s): σ[1] |=φ.
- s |= E[φ U ψ] iff ∃σ∈$\mathcal{P}_M$(s): ∃j≥0, σ[j] |=ψ
  ∧ for each 0≤k<j, σ[k] |=φ.
- s |= A[φ U ψ] iff ∀σ∈$\mathcal{P}_M$(s): ∃j≥0, σ[j] |=ψ
  ∧ for each 0≤k<j, σ[k] |=φ.

bla

4. Contronto LTL e CTL:

Solution:

- **LTL**:
  - Nozione **lineare** di tempo
  - $R : S \to S$: Ogni stato ha un solo **successore**
  - Non può esprimere **possibilità**
  - Proprietà definite sul **singolo cammino**, quindi (quando verifico il modello) su **tutti** i cammini che originano nello stato (tracce)
  - Esprimere **fairness** nella formula
- **CTL**:
  - Nozione **branching** di tempo
  - $R : S \to 2^S$: Ogni stato ha più **successori**

- Esprime **possibilità**
- Proprietà definite su **ogni possibile cammino** che origina nello stato
- Weak e Strong **fairness** non esprimibili nella formula

- **Teorema sintattico**:
  - Sia ϕ una formula CTL e ψ una formula LTL ottenuta eliminando da ϕ tutti i quantificatori di cammino, allora:
  - $\phi \equiv \psi$ oppure
  - Una formula LTL equivalente non esiste

- **Formule LTL non esprimibili in CTL**:
  - FGp
  - $F(p \wedge Xp)$
  - $GFp \Rightarrow Fq$

- **Formule CTL non esprimibili in CTL**:
  - $AF\ AGp$
  - $AF(p \wedge AXp)$
  - $AG\ EFp$

5. CTL*

Solution:

$$\begin{array}{l} State\ formula: \ \phi = \ p \mid (\phi) \mid \neg\phi \mid \phi \vee \phi \mid E\phi \mid A\phi \\ Path\ formula: \ \psi = \ \phi \mid \neg\psi \mid \psi \vee \psi \mid X\phi \mid \phi U \phi \end{array} \tag{5}$$

6. Model checking:

Solution:

- L'algoritmo risolve il problema: dato un modello M e una formula CTL ϕ, quali sono tutti gli stati s per cui $(M, s) \models \phi$?

- **Sotto-formule**:

  Definition of sub-formulae. Let p in AP, φ and ψ be CTL formulae, then the set of sub-formulae is defined as:

  $$\begin{aligned}
  &Sub(p) && = \{p\}\\
  &Sub(\neg\varphi) && = Sub(\varphi) \cup \{\neg\varphi\}\\
  &Sub(\varphi \vee \psi) && = Sub(\varphi) \cup Sub(\psi) \cup \{\varphi \vee \psi\}\\
  &Sub(EX\varphi) && = Sub(\varphi) \cup \{EX\ \varphi\}\\
  &Sub(E[\varphi\, U\, \psi]) && = Sub(\varphi) \cup Sub(\psi) \cup \{E[\varphi\, U\, \psi]\}\\
  &Sub(A[\varphi\, U\, \psi]) && = Sub(\varphi) \cup Sub(\psi) \cup \{A[\varphi\, U\, \psi]\}
  \end{aligned}$$

- **Sat-set**: Insieme di stati che soddisfano una formula
- **Algoritmo**:
  - Comincia con le sottoformule di lunghezza 1 e procede per induzione fin quando la formula di lunghezza $|\phi|$ è calcolata
  - *function* Sat(ϕ: formula CTL, S: insieme di stati): insieme di stati
  - If $\phi = true$ return S
  - If $\phi = false$ return $\emptyset$
  - If $\phi \in AP$ return $s : \phi \in L(s)$ (gli stati etichettati con ϕ)
  - :

    [] $\varphi = \neg\varphi_1$ --> return S - Sat(φ_1)

    [] $\varphi = \varphi_1 \vee \varphi_2$ --> return Sat(φ_1) $\cup$ Sat(φ_2)

    [] $\varphi = EX\varphi_1$ --> return $\{s \in S \mid \exists\ (s,s') \in R \wedge s' \in Sat(\varphi_1)\}$

    [] $\varphi = E[\varphi_1 U \varphi_2]$ --> return $Sat_{EU}(\varphi_1, \varphi_2)$

    [] $\varphi = A[\varphi_1 U \varphi_2]$ --> return $Sat_{AU}(\varphi_1, \varphi_2)$

    (* postcondition: $Sat(\varphi) = \{s \in S \mid (M,s) \models \varphi\}$

    end

- **Sat_{EU}**:
  - Inizialmente metto in Q tutti gli stati che soddisfano ψ ($Sat(\psi)$)

- Per ogni stato che è predecessore di uno degli stati in Q (uso RG), controllo se soddisfa ϕ (oppure calcolo intero insieme predecessori e poi intersezione con $Sat(\phi)$) e nel caso lo aggiungo in Q (oppure unione con Q)
- Ricomincio
- Termino quando non trovo altri stati (a punto fisso)

- **Complessità**:

$$O(|\phi| * |Sys|^3)$$

Alla terza per Sat_{AU}. Altrimenti $|Sys|^2$ in algoritmi più efficienti in cui si trasforma tutto in forma esistenziale.

In LTL: $O(|Sys| * 2^{|\phi|})$ ma comunque migliore perché formule di solito corte (n° di operatori), mentre $|Sys| = |V| + |E|$

7. Fairness

Solution:

- **Weak** e **Strong** fairness constraints non possono essere espressi in CTL (FG)
- Bisogna **modificare l'algoritmo** per considerare solo esecuzioni fair
- **Fair CTL**:
  - Un fair CTL-model è una quadrupla: $M = (S, R, L, F)$ dove $F \subseteq 2^S$ è un insieme di **vincoli di fairness**

$$F = \{F_1, F_2, ..\}$$

  - $F_i = \{f_1, f_2, .., f_n\}$, dove f_i sono **stati**
  - Un cammino σ è **F-Fair** se passa infinitamente spesso per almeno uno stato per ogni insieme f_i (come per Automa di Buchi)
  - ${\mathcal{P}^f}_M(s)$: insieme di tutti i cammini F-fair che originano in s

8 Timed Automata

1. Introduzione

Solution:

- **Correttezza** può dipendere dal tempo (es. pipeline dati)
- **Utilità** può dipender dal tempo (es. metro)
- Controllare **proprietà temporali** $->$ tempo deve essere rappresentato nel modello
- Alcuni formalismi temporizzati:
  - Timed automata
  - Timed Petri nets
  - Timed process algebra

  $->$ Semantica data in termini di **Timed Transition System**, un sistema di transizioni temporizzate

2. Come specificare tempo

Solution:

- Un valore $->$ ritardo fissato per un'attività
- Un intervallo (min-max) $->$ valore non deterministico dentro un intervallo
- Una distribuzione stocastica $->$ la durata di un'attività è un valore estratto da una distribuzione
- Definendo dei clock come variabili che incrementano costantemente

3. Differenza discreto-continuo

Solution:

Discreto:

- Tempo come entità discreta

- Tempo scorre in tick regolari
- Eventi possono accadere solo al tick
- Usato per rappresentare sistemi sincroni (sistemi con un clock globale discreto)
- Può rappresentare un'astrazione di un sistema continuo
- Stato del sistema: valore delle variabili/marking + valore dei clock

Continuo:

- Tempo come entità continua
- Tempo scorre continuamente
- Eventi possono accadere a ogni istante
- Usato per rappresentare sistemi asincroni
- Stato del sistema: valore delle variabili/marking + valore dei clock

4. Definizioni: timed automata

Solution:

- **Timed automata**: Grafo a stati finiti con:
  - **Locazioni** e archi
  - **Variabili di clock**
  - **Vincoli di clock**:
    - **Invariante di locazione**: nelle locazioni, non può essere violato
    - **Guardia su arco**: nessun obbligo di prendere arco
  - **Reset di clock**
  - Il tempo scorre nelle locazioni e non negli archi
- **Clock**: Un clock è una variabile su $\mathcal{R}^+$
- **Vincoli di clock**: Sia C un insieme di clock, con $c \in C$ e $c \in \mathcal{N}$, allora:
  1. $x < c$ e $x \leq c$ sono vincoli di clock
  2. Se α è un vincolo di clock, allora $\neg\alpha$

3. Se α e β sono vincoli di clock, allora $\alpha \wedge \beta$ è un vincolo di clock
4. Nient'altro è un vincolo di clock

$->$ L'insieme dei vincoli di clock su C è indicato con $Cstr(C)$

- **Definizione timed automata**: Un timed automata A è una tupla

$$(L, l_0, E, Label, C, clocks, guard, inv)$$

  - clocks: $E \to 2^C$: una funzione che assegna a ogni arco $e \in E$ un insieme di clock $clocks(e)$ da resettare
  - guards: $E \to Cstr(C)$: una funzione che assegna ad ogni arco $e \in E$ un vincolo di clock ($guard(e)$) (guardia sull'arco)
  - inv: $E \to Cstr(C)$: una funzione che assegna ad ogni locazione $l \in L$ un vincolo di clock ($inv(e)$) (invariante di locazione)

5. Semantica: TTS

Solution:

- **Valutazione dei clock**: v è una funzione $v : C \to \mathcal{R}^+$, che assegna a ogni clock $x \in C$ il suo valore corrente $v(x)$ $->$ Denotiamo con $V(C)$ l'insieme di tutte le valutazioni dei clock su C
- **Stato**: Uno stato di un timed automata A è la coppia

$$(l, v)$$

  dove l è una locazione di A e v una valutazione sui clocks in C di A

- **Timed Transition System (TTS)**: sottostante ad un timed automata A, è definito come

$$M(A) = (S, s_0, \to)$$

  - $S = \{(l, v) \in L \times V(C) : v \models inv(l)\}$: ossia tutti i possibili stati
  - $s_0 = (l_0, v_0)$ dove $v_0(x) = 0 \forall x \in C$: tutti i clock a zero nello stato iniziale
  - **Relazione di transizione** è definito dalle regole:
    1. $(l, v) \xrightarrow{*} (l', reset\ clocks(e)\ in\ v)$ (transizione discreta) se tutte le seguenti condizioni sono soddisfatte:
       (a) $e = (l, l') \in E$: ci dev'essere un arco fra le due locazioni
       (b) $v \models guard(e)$: la guarda sull'arco è rispettata

(c) $(reset\ clocks(e)\ in\ v) \models inv(l')$: i clock dopo il reset, dovuto alla transizione, rispettano l'invariante di locazione della locazione in cui si arriva

2. $(l, v) \xrightarrow{d} (l, v + d)$ (let time elapses), per reali positivi d, se:

$$\forall d' \leq v + d \models inv(l)$$

- **Cammino temporizzato**: Il tempo trascorso su un cammino è definito come:

Definition 44. (Elapsed time on a path)
For path $\sigma = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots$ and natural i, the time elapsed from s_0 to s_i, denoted $\Delta(\sigma, i)$, is defined by:

$$\begin{aligned} \Delta(\sigma, 0) &= 0 \\ \Delta(\sigma, i{+}1) &= \Delta(\sigma, i) + \begin{cases} 0 & \text{if } a_i = * \\ a_i & \text{if } a_i \in \mathbb{R}^+. \end{cases} \end{aligned}$$

- **Cammino tempo-divergente**: Se

$$\lim_{i \to \infty} \Delta(\sigma, i) = \infty$$

$->$ Un timed automata è chiamato **non-Zeno** se da ogni stato ne può partire uno

6. TCTL

Solution:

Sintassi

- $$\boxed{\phi = \ p \mid \alpha \mid (\phi) \mid \neg\phi \mid \phi \vee \phi \mid z\ in\ \phi \mid E[\phi U \psi] \mid A[\phi U \psi]} \quad (6)$$
- D è l'insieme dei clock di formula
- $\alpha \in Cstr(C \cup D)$: vincolo sui clock dell'automa o sui clock di formula
- $z \in D$
- **z in** ϕ è detto **freeze identifier** e significa: "z in ϕ" è valido nello stato s se ϕ è soddisfatto in s, stato in cui il clock z parte da 0

Semantica

Def.: A RT-trajectory σ is an infinite sequence of states $s_i = (l_i, v_i)$ and delays δ_i:

$$\sigma = s_0 \xrightarrow{\delta_0} s_1 \xrightarrow{\delta_1} s_2 \xrightarrow{\delta_2} s_3 \xrightarrow{\delta_3} \ldots\ldots\ldots\ldots$$

Def.: A position in σ is the pair (i,δ): $i \in \mathcal{N}$ *and* $\delta \leq \delta_i$

Def.: location in the position (i,δ) is $loc(i,\delta) = l_i$

Def.: valuation in the position (i,δ) is $val(i,\delta) = v_i + \delta$

Def.: state in position (i,δ) is

$$\sigma(i,\delta) = (\, loc(i,\delta),\, val(i,\delta)\,)$$

Def: Semantics of TCTL. Let $p \in AP$, $z \in D$, $w \in V(D)$, $s \in S$ (States of the TTS), $\alpha \in Cstr(C \cup D)$, $s=(l,v)$, $v \in V(C)$, the set of TCTL formulae is given by:

$$\begin{array}{ll}
s, w \models p & \text{iff } p \in Label(s) \\
s, w \models \alpha & \text{iff } v \cup w \models \alpha \\
s, w \models \neg \phi & \text{iff } \neg (s, w \models \phi) \\
s, w \models \phi \vee \psi & \text{iff } (s, w \models \phi) \vee (s, w \models \psi) \\
s, w \models z \text{ in } \phi & \text{iff } s, \text{reset } z \text{ in } w \models \phi
\end{array}$$

.....cont.: let $p \in AP$, $z \in D$, $w \in V(D)$, $s \in S$, $\alpha \in Cstr(C \cup D)$, $P^{\infty}_{\mathcal{M}}(s)$ the RT-trajectories starting in s,

$$\begin{array}{ll}
s, w \models \mathbf{E}[\phi \,\mathbf{U}\, \psi] & \text{iff } \exists\, \sigma \in P^{\infty}_{\mathcal{M}}(s).\, \exists\, (i,d) \in Pos(\sigma). \\
& \quad (\sigma(i,d), w + \Delta(\sigma,i)) \models \psi \;\wedge \\
& \quad (\forall\, (j,d') \ll (i,d).\, \sigma(j,d'), w + \Delta(\sigma,j)) \models \phi \vee \psi)) \\
s, w \models \mathbf{A}[\phi \,\mathbf{U}\, \psi] & \text{iff } \forall\, \sigma \in P^{\infty}_{\mathcal{M}}(s).\, \exists\, (i,d) \in Pos(\sigma). \\
& \quad ((\sigma(i,d), w + \Delta(\sigma,i)) \models \psi \;\wedge \\
& \quad (\forall\, (j,d') \ll (i,d).\, (\sigma(j,d'), w + \Delta(\sigma,j)) \models \phi \vee \psi))
\end{array}$$

7. Clock equivalence

Solution:

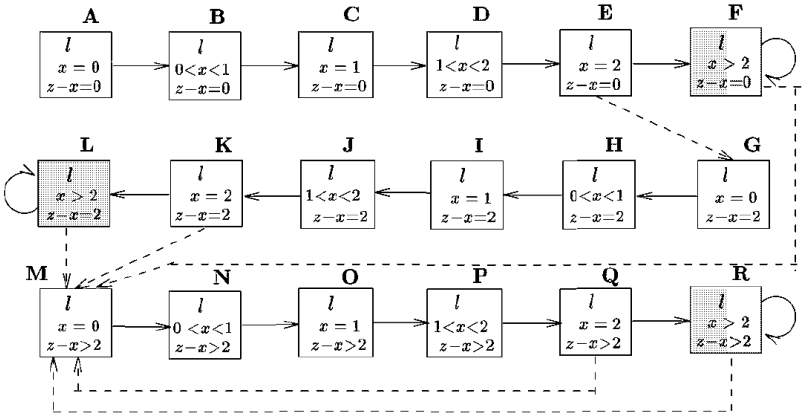
- Per la verifica è necessario costruire un **TTS finito** ed equivalente
- Trovare un numero finito di **classi di equivalenze** sulle valutazioni dei clock

Definition 49. (**Clock equivalence**)
Let $\mathcal{A}$ be a timed automaton with set of clocks C and $v, v' \in V(C)$. Then $v \approx v'$ if and only if

1. $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ or $v(x) > c_x$ and $v'(x) > c_x$, for all $x \in C$, and
2. $frac(v(x)) \leqslant frac(v(y))$ iff $frac(v'(x)) \leqslant frac(v'(y))$ for all $x, y \in C$ with $v(x) \leqslant c_x$ and $v(y) \leqslant c_y$, and
3. $frac(v(x)) = 0$ iff $frac(v'(x)) = 0$ for all $x \in C$ with $v(x) \leqslant c_x$.

1. Parte intera uguale o entrambi maggiori della massima costante presente
2. Per la parte frazionaria devono preservare l'ordine
3. La parte frazionaria o per entrambi 0 o per entrambi diversa da 0 (perché verifichiamo che un clock sia esattamente uguale ad un numero naturale)

- **Region automata**:
  - Una regione è una **coppia** $(l, [v])$ dove l è una locazione e $[v]$ è una classe di equivalenza sulle valutazioni dei clock
  - 2 tipi di transizione: *let time elapses* o *take a transition*



9 Decision Diagram

1. Saper leggere DD
2. Definizioni

Solution:

- **Decision Diagram:**
  - Grafo diretto acicliclo
  - Per codificare e memorizzare insiemi
  - Insiemi di tuple o insiemi per rappresentare funzioni
  - Una tupla è un assegnamento di valori su un insieme variabili (es. marcatura: numero di token per ogni posto)
  - In un BDD gli assegnamenti possibili sono 0 o 1
  - Ogni livello rappresenta una variabile
  - Ogni nodo rappresenta un possibile assegnamento di valori alla variabile associata a quel livello
  - Nodi terminali: 0 e 1
  - Una tupla è codificata da un cammino dalla radice a un nodo terminale
- **Canonico**:
  - Ordinamento associato alle variabili
  - Nodi organizzati in N+1 livelli
  - I nodi non hanno duplicati (no isomorfismo)
  - Livello 0 contiene solo 2 nodi terminali (0,1)
  - Un nodo non terminale ha solo due archi uscenti, uno per 0 e uno per 1
- **Quasi reduced**: non contiene nodi duplicati (condizione necessaria per essere canonico)
- **Fully reduced**: // e non contiene nodi ridondanti $->$ sia con 0 che con 1 vanno nello stesso nodo
- **Full storage**: codifico tutti i cammini
- **Sparse storage**: codifico solo i cammini che vanno in 1

3. Operazioni principali

Solution: Operazioni si fanno al livello di nodo (non a livello di insieme delle tuple codificate) $->$ Costo proporzionale al numero di nodi del grafo $->$ molto efficiente
Operazioni base di tipo insiemistico:

- Unione $+$
- Intersezione $*$
- Controllo di equivalenza $==$: facile se canonici (perché sono univoci per codificare lo stesso insieme) $->$ devono essere identici

Operazioni specifiche:

- Valutare se un elemento appartiene all'insieme $\mathcal{EV}$: se esiste cammino che porta nel terminale 1
- Operazione di post immagine/probulletto relazionale $\bullet$: permette di codificare funzioni
- Enumerare i cammini del BDD: visita di un grafo

4. Commentare algoritmi

Solution:

- Algoritmi ricorsivi sui nodi
- **Unique Table**: tabella hash che memorizza nodi e assicura di non avere nodi duplicati:
  - **UniqueTable.Insert**(level, 0–value, 1–value) $->$ new DD node
- **Operation Cache**:
  - Tabella hash
  - Memorizza risultati operazioni intermedie già fatte (esistono perché nodi sono in comune): permette di non fare discese ricorsive ridondanti
  - Per raggiungere complessità polinomiale
  - **OperationCache.Insert**(operation, 1st–operand, 2nd–operand, result)
  - **OperationCache.Search**(operation, 1st–operand, 2nd–operand, result)

Unione:

- Operazioni sui nodi: unione di BDD si riduce a unione di nodi
- *UnionBDD(BDD p, BDD q)*:
  - Cerco in Operation Cache se unione già fatta, altrimenti la calcolo
  - Caso base: **OR** dei nodi terminali
  - (p.lvl == q.lvl): inserisco nuovo nodo in tabella (stesso livello, unione valori uscenti da 0, unione valori uscenti da 1)
  - altri due casi: vuol dire che c'è un nodo ridondante che non abbiamo codificato $->$ stessa destinazione (nodo più basso) per 0 e per 1
  - Inserisco risultato in Operation Cache

Intersezione: uguale a Unione ma con **AND**

5. DD per RS vs DD per next-state function

Solution:

- Codifichiamo funzioni sotto una forma di tipo insiemistico
- Se abbiamo L variabili, usiamo 2*L livelli: predecessori e successori (prima e dopo applicazione funzione)
- Codificano sempre tuple, cambia l'interpretazione che ne diamo.
- Dato un DD che rappresenta un insieme di tuple e un DD di tipo relazionale che rappresenta una funzione, possiamo usare l'operazione di post immagine per ottenere i valori successori dei valori delle tuple $->$ se per quel nodo successore esiste un cammino che lo attraversa, che ha, come valori dei predecessori, i valori codificati nel DD originale (valori assegnabili per quelle variabili).
- Possiamo così codificare transizioni (*next-state function*) come BDD relazionali:
  - Livello 0 P0: marcatura P0 prima della transizione
  - Livello 1 P0': marcatura P0 dopo scatto transizione
  - Livello 2 P1: ecc.

6. Algoritmo RS

Solution:

- Possiamo codificare transizioni (*next-state function*) come BDD relazionali:
  - Livello 0 P0: marcatura P1 prima della transizione
  - Livello 1 P0': marcatura P1 dopo scatto transizione
  - Livello 2 P1: ecc.
- Unione + post immagine $->$ esplorazione spazio degli stati rete di Petri:
- Prendiamo tutte le transizioni e ne facciamo l'unione $->$ Next-state function NFS (BDD relazionale/di trasformazione) $\mathcal{N}$
- *GenerateRS-BFS*(BDD $\mathcal{X}$, BDD $\mathcal{N}$):
  1. Partiamo da marcatura iniziale: $-> \mathcal{X} = \mathcal{X}^{init}$
  2. Facciamo poi l'operazione di post immagine fra marcatura iniziale e NFS $->$ marcature a distanza uno (risultato dell'applicazione delle transizioni alla marcatura iniziale)
  3. $\mathcal{X}$ = Unione fra marcatura iniziale e risultato post immagine (marcature a distanza uno)
     $-> O = Union(O, RelationalProduct(X, N))$
  4. Si ripete (successivamente applico post immagine a insiemi di marcature sempre più grandi fin quando arrivo a punto fisso)

7. Efficienza

Solution:

- *Generazione esplicita RS*:
  - **Strutture dati esplicite**: ogni stato richiede una locazione diversa in memoria $-> O(|X_{rch}|)$ *memory*
  - **Algoritmo esplicito**: stati manipolati uno a uno $-> O(|X_{rch}|)$ *time*

  $->$ Richiesta di memoria cresce linearmente quando vengono trovati nuovi stati
- *Generazione simbolica RS*:
  - **Strutture dati simboliche**: ogni locazione di memoria potrebbe memorizzare informazione riguardo più stati (dipende dal numero di nodi che

sono necessari per codificare quegli stati) $->$ $O(|X_{rch}|)$ *memory* solo nel caso peggiore

- **Algoritmo simbolico**: stati manipolati un insieme ala volta $->$ $O(|X_{rch}|)$ *time* solo nel caso peggiore

$->$ Richiesta di memoria cresce e si riduce quando vengono trovati nuovi stati (numero di nodi non proporzionale al numero di stati codificati, ma dipendono da ordine variabili e da località/indipendenza). Picchi di solito verso metà del processo, non alla fine.

- Molto efficiente in ambito di processi concorrenti, esempio filosofi: maggior parte delle sotto-marcature uguali (stati interni, località delle transizioni, indipendenza) $->$ sharing; sotto-marcature diverse riguardano interazioni

8. Multi-way DD

Solution:

- Generalizzazione del BDD agli interi
- Ciascuna variabile ha un proprio dominio, es. : $S4 = \{0, 1, 2, 3\}, S3 = \{0, 1, 2\}$
- Definizioni uguali a quelle per BDD
- Unione: uguale ma con ciclo per fare unione fra tutti i possibili valori che le variabili p e q possono assumere
- Algoritmo RS: uguale

9. Model checking CTL con MDD

Solution: Da EX, EU e EG possiamo derivare l'intero set di operatori di CTL, quindi sono sufficienti solo i seguenti algoritmi:

- **Algoritmo EX**:
  - $ComputeEX(Sp, \mathcal{N}^{-1})$
  - Sp è l'insieme degli stati che soddisfano p
  - $\mathcal{N}^{-1}(s_i)$ è l'insieme dei predecessori di s_i
  - Esplicito: L'algoritmo quindi per ogni stato $s \in Sp$ aggiunge $\mathcal{N}^{-1}(s)$

  - Restituisce l'insieme degli stati che soddisfano $EX\ p$
  - Simbolico: Per ottenere $\mathcal{N}^{-1}(s_i)$, per ogni MDD che rappresenta una transizione, ne facciamo l'inverso. Facciamo poi l'unione di tutte ottenendo la next-state function inversa (*previous-state function*)
  - Con gli MDD quindi, possiamo ottenere in un passo l'insieme di soddisfacibilità applicando la post immagine all'insieme Sp (quindi non per ogni stato, ma direttamente sull'insieme), codificato anch'esso in un MDD, con la previous-state function:

$$S_\phi = RelationalProduct(Sp, \mathcal{N}^{-1})$$

- **Algoritmo EU**:
  - $ComputeEu(Sp, Sq)$
  - Inizialmente $S_\phi = Sq$
  - Faccio un ciclo di applicazioni della funzione inversa a S_ϕ, ottenendo ad ogni passo l'insieme dei predecessori
  - Esplicito: Per ogni stato predecessore degli stati aggiunti finora, verifichiamo se soddisfano p: se soddisfano, li inseriamo
  - Simbolico: Per ogni insieme degli stati predecessori, ne facciamo l'intersezione con Sp e poi uniamo il risultato con S_{phi}:

$$Prev = RelationalProduct(S_{phi}, \mathcal{N}^{-1});$$

$$Prev = Intersection(Prev, Sp);$$

$$S_\phi = Union(S_\phi, Prev);$$

- **Algoritmo EG**:
- $ComputeEG(Sp)$:
- Esplicito: Usa SCC in cui è soddisfatto p ($S_\phi = SCC(Sp)$), e poi cerca all'indietro stati che finiscono nella SCC
- Simbolico: Inizialmente $S_\phi = Sp$
- Calcolo ciclicamente l'intersezione fra S_ϕ e $EX(S_\phi, \mathcal{N}^{-1})$:

$$Prev = RelationalProduct(S_{phi}, \mathcal{N}^{-1});$$

$$S_\phi = Intersection(S_{phi}, Prev);$$

10. Model checking LTL con MDD

Solution:

- Prendiamo automa di buchi del sistema e l'automa di buchi del negato della formula e ne facciamo l'automa prodotto (intersezione).
- Se l'intersezione è vuota, la formula è verificata, altrimenti abbiamo un contro-esempio
- $SatELTL(A_{sys}, A_{\neg\phi})$:
- Prendo automa di buchi del negato della formula
- Prendo RS e NSF della rete di Petri come MDD
- $TS = A_{sys} \otimes A_{\neg\phi}$
- Calcolo RS_0^{TS}: MDD di RS in cui codifico anche locazione automa di Buchi (ogni stato in una singola locazione)
- Calcolo NFS^{TS}: MDD di NFS in cui codifico anche transizioni automa di Buchi
- Produco RS^{TS}: RS incrociato facendo post immagine fra RS_0^{TS} e NFS^{TS}
- Calcolo FS^{TS}: uso $E_{fair}G(true)$ per calcolare i fair states usando le locazioni accettanti come fair sets (quali sono gli stati che passano infinitamente spesso da locazioni accettanti) $->$ LTL è implementato attraverso CTL fair, perché CTL fair contiene la logica di accettazione dell'automa di Buchi
- Ritorno intersezione RS e FS^{TS}
- Se vuota, formula di partenza soddisfatta

11. Model checking CTL* con MDD

Solution:

- $SatCTL*$
- Riscrive la formula in formula che non contiene quantificazioni, che sia quindi nel linguaggio LTL
- A questa applico $SatELTL$

- Per riscrivere, mantengo tutto ciò che c'è in LTL. Se incontro quantificatore, valuto ricorsivamente la formula con $SatCTL*$. Ottengo un sat-set di stati che soddisfano quella formula e lo sostituisco con una nuova a_p (atomic proposition), restituendola.

12. $ComputeDeadlock(RS, \mathcal{N}^{-1})$

Solution:

- Trovo stati che non sono predecessori di nessuno:

$$Prev = RelationalProduct(RS, \mathcal{N}^{-1});$$

$$S_\phi = Difference(RS, Prev);$$