Operatori per il Parallelismo in CSP

Composizione Parallela di Processi Sequenziali (1)

- Le notazioni introdotte permettono di descrivere agevolmente tutti i processi che si presentano solo sotto forma sequenziale
- Componendo in parallelo 2 o più processi sequenziali la complessità e l'interesse aumentano notevolmente

Composizione Parallela di Processi Sequenziali (2)

- Con la composizione parallela lo stato dell'intero sistema dipende dallo stato dei processi componenti
  - Il numero di stati aumenta esponenzialmente con l'aumentare dei processi
  - La difficoltà nel gestire tali sistemi è elevatissima

Composizione Parallela di Processi Sequenziali (3)

- CSP considera la combinazione parallela come un altro processo a cui applicare gli operatori già definiti
- Nell'esecuzione parallela i processi si influenzano l'un l'altro attraverso le rispettive comunicazioni

Obiettivo

- Evitare
  - Deadlock
  - Livelock
    - sequenza infinita di comunicazioni interne tra componenti, con un'apparenza esterna analoga al deadlock
  - Comportamenti non deterministici

Sincronizzazione di azioni visibili (1)

- L'operatore || è quello più semplice e permette di sincronizzare le azioni visibili svolte dai due processi operandi
- P || Q esegue l'azione a $\in \Sigma$ sse sia P che Q possono eseguire a
- Quindi:

$$(?x: A \text{ -> } P(x)) \mid\mid (?x: B \text{ -> } Q(x)) =$$

$$?x: A \cap B \text{ -> } (P(x) \mid\mid Q(x))$$

Sincronizzazione di azioni visibili (2)

- La legge precedente riflette l'idea che ogni processo parallelo è equivalente ad uno sequenziale:
  - I due prefissi posti in parallelo vengono trasformati in un singolo processo esterno al parallelismo
- Di conseguenza, CSP permette di governare agevolmente la complessità del sistema risultante

Esempio (1)

- Consideriamo un processo che comunica il simbolo **a**,
  - sempre tranne quando il numero di **a** comunicate è divisibile per N,
  - in tal caso comunica il simbolo **b**
- Tale processo può essere espresso come

$$\text{Mult}(N, m) = \begin{cases} a \text{ -> Mult}(N, N) \,\square\, b \text{ -> Mult}(N, m) & \text{se } m=0 \\ a \text{ -> Mult}(N, m-1) & \text{altrimenti} \end{cases}$$

Esempio (2)

- Combinando in parallelo Mult (N, 0) e Mult (M, 0) si ottiene

Mult(N, 0) || Mult(M, 0) = Mult(mcm(M, N), 0)

dove mcm(M, N) è il minimo comune multiplo tra M e N

- E' un handshacking: b si verifica solo se entrambi i comunicanti sono in accordo
  - non c'è direzione nella comunicazione

Esempio (3)

- Lo stesso meccanismo può avere l'effetto di un output di un processo verso un altro

(c!x -> P) || (c?y -> Q(y)) = c!x -> (P||Q(x))

c è input: Q evolve in funzione di c

c è output: P evolve indipendentemente da c

Interfaccia

- Per specificare che una coppia di processi deve sincronizzarsi rispetto a (deve effettuare un handshaking su) un determinato evento è definito l'operatore di parallelismo attraverso l'interfaccia X
- $P \,||_X\, Q$
  - forza i processi P e Q a sincronizzarsi rispetto a tutti gli eventi in X
  - ma permette comunque di effettuare eventi esterni a X

Modello delle Tracce

- È un modello per la descrizione del comportamento di un processo
  - ogni processo è rappresentato dal suo insieme di tracce
- Ad es.
  - traces(Stop) = {<>}, con <> sequenza vuota
  - traces(μP.a -> P □ b-> Skip) = $\{\langle a\rangle^n, \langle a\rangle^n \wedge \langle b\rangle, \langle a\rangle^n \wedge \langle b, \surd\rangle \mid n \in \mathbb{N}\}$, dove
    - s^t è la concatenazione di t a s
    - s^n è la concatenazione di s con sé stessa n volte
    - √ è il simbolo di terminazione

Proprietà Modello Tracce (1)

- Ogni processo è rappresentato dal suo insieme di tracce
- Per il set delle tracce valgono sempre le seguenti proprietà
  - P1: È non vuoto
    - include almeno la empty trace
  - P2: È chiuso rispetto all'operazione di prefix
- Le tracce assumono valori in un insieme di sequenza finite di simboli di Σ, con eventualmente un √ finale

Proprietà Modello Tracce(2)

- Sia X un insieme di simboli e X* l'insieme di sequenza finite di membri di X (inclusa la sequenza vuota), allora

  traces (P) ⊆ Σ* ∪ {s ^ <√> | s ∈ Σ*}
- Il modello delle tracce T è l'insieme di tutti i sottoinsiemi di Σ* ∪ {s ^ <√> | s ∈ Σ*} che soddisfano P1 e P2

Modello Tracce per CSP (1)

- traces (stop) = {<>}
- traces (a -> P) = {<>} ∪ {<a>^s | s ∈ traces (P)}
  - Processo vuoto oppure processo in cui l'evento iniziale a è seguito da una trace di P

Modello Tracce per CSP (2)

- traces (?x: A -> P(x)) = {<>} ∪ {<a>^s | a ∈ A ∧ s ∈ traces (P[a/x])}
  - analogo al precedente, con la differenza che l'evento iniziale è scelto nell'iniseme A e da questo dipende l'evoluzione successiva;
  - P[a/x] indica che a sostituisce le occorrenze di x

Modello Tracce per CSP (3)

- traces (c?x: A -> P(x)) = {<>} ∪ {<c.a>^s | a ∈ A ∧ s ∈ traces (P[a/x])}
  - analogo al precedente
- traces (P □ Q) = traces (P) ∪ traces (Q)
- traces (P ⊓ Q) = traces (P) ∪ traces (Q)
- traces (P||Q) = traces (P) ∩ traces (Q)

Modello Tracce per la ricorsione (1)

- P = F(P) significa che P ha lo stesso comportamento (e quindi la stessa trace) del processo F(P)
- In termini di tracce, la F() rappresenta sempre un mapping monotono daT a se stessa,
  - cioè, se traces $(Q_1) \subseteq$ traces (Q_2) allora traces $(F(Q_1)) \subseteq$ traces $(F(Q_2))$

Modello Tracce per la ricorsione (2)

- Quindi traces (F(Stop)) ⊆ traces (P)
- Applicando F() e la monotonia si ottiene traces $(F^n(Stop)) \subseteq$ traces (P) per ogni n, dove $F^n(Stop)$ è ciò che si ottiene applicando n volte a Stop
- Quindi

traces (P) = ∪ {traces$(F^n(Stop))$ | n ∈ $\mathbb{N}$}