

Modelli e sistemi concorrenti

A.A. 2023/2024

Indice

1. Introduzione	6
1.1. Modello sequenziale	6
1.2. Sistemi reattivi	6
1.3. Equivalence checking	8
1.4. Semantica a tracce	8
1.5. Non determinismo	9
1.6. Equivalenze diverse	9
2. Labeled Transition Systems (LTS)	11
2.1. Proprietà	13
2.2. Relazione di raggiungibilità	13
2.3. Classi di LTS	14
3. Equivalenze forti	15
3.1. Isomorfismo	15
3.2. Equivalenza a tracce	15
3.2.1. Trace preorder	16
3.2.2. Equivalenza a tracce completa	16
3.3. Simulazione	16
3.4. Bisimulazione	20
3.5. Bisimulazione ricorsiva (punto fisso)	22
3.6. Bisimulazione up to \sim	23
4. Equivalenze deboli	24
4.1. Tracce weak	24
4.2. Tracce weak complete	25
4.3. LTS τ -free	25
4.4. Simulazione weak	26
4.5. Simulazione strong	26
4.6. Simulazione weak completa	28
4.7. Bisimulazione weak	28
4.7.1. LTS τ free	30
4.8. LTS astratto per bisimulazione weak	31
4.9. Proprietà varie	32
4.10. Divergenza	32
4.11. Bisimulazione weak up to \approx	33
4.12. Rooted weak bisimilarity	34
5. Calculus of Communicating Systems (CCS)	35
5.1. Sintassi	35
5.2. Semantica	36
5.3. Flow graph	38
5.3.1. Operatore di linking / pipeline	39
5.4. Costanti	40
5.4.1. Costanti guardate	41
5.5. Processi	41
5.6. Semantica Operazionale Strutturata	41
5.7. Gerarchie	45
5.7.1. CCS Finite	46

5.7.2. CCS finite-state	48
5.7.3. CCS Regular	50
5.7.3.1. Buffer a n posizioni	54
5.7.4. BPP: Basic Parallel Processes	57
5.7.5. CCS Finite-net	59
5.7.6. CCS Finitary	60
5.7.6.1. Turing completezza	61
5.7.7. CCS Full	64
5.8. CCS Value Passing	65
5.9. CCS $^\oplus$	68
5.10. CCS $^{\text{hide}}$	68
5.11. CCS $^{\text{rel}}$	69
5.12. CSS Full relabeling	70
5.13. Composizione sequenziale	70
5.13.1. F-trace	70
5.13.2. F-bisimulation	71
5.13.3. Weak F-bisimulation	71
5.13.4. Rooted weak f-bisimulation	71
5.14. Finite BPA	71
5.15. BPA *	72
5.16. BPA	73
5.17. Process Algebra	74
5.18. PAER	74
5.19. CCS $^{\text{seq}}$	75
5.19.1. Da CCS $^{\text{seq}}$ a CCS up to \approx_f	76
5.20. CSP sync multi-part	76
5.21. Problema dei filosofi a cena	77
5.22. Multi-CCS	77
5.22.1. Interleaving	79
5.22.2. Semantica a step	80
6. Proprietà algebriche	82
6.1. Proprietà di \sim	82
6.1.1. +	82
6.1.2. 	84
6.1.3. (νa)	84
6.2. Legge di espansione / interleaving	85
6.3. Free names $fn(p)$	85
6.4. Bound names $bn(p)$	85
6.5. Sostituzione	86
6.6. Proprietà di \approx	88
6.6.1. Lemma di Hennessy	88
6.6.2. 	89
7. Congruenze comportamentali	90
7.1. Congruenza \Rightarrow Equivalence-checking compostionale	90
7.2. Congruenza \Rightarrow Minimizzazione compostionale	90
7.3. Alcune congruenze e non	90
7.3.1. Isomorfismo	90

7.3.2. Strong bisimilarity \sim	90
7.3.3. Trace equivalence $Tr(p)$	91
7.3.4. Completed trace/simulation equivalence	91
7.3.5. Weak bisimilarity \approx	91
7.3.6. Rooted weak bisimilarity \approx^c	91
7.4. $\approx^c \subseteq \approx$	92
8. Assiomatizzazioni	93
8.1. CCS finite “open”	93
8.2. Deduzione equazionale	93
8.3. Operatore $+$	94
8.4. Assiomatizzazioni (ground) sound e (ground) complete	94
8.5. Assiomatizzazione SB (non finita) di \sim per CCS finite	94
8.6. Assiomatizzazione per simulation/trace equivalence	96
8.7. Assiomatizzazione finita per trace equivalence	96
8.8. Assiomatizzazione WB per \approx^c	96
8.8.1. Saturation Lemma	96
8.9. Assiomatizzazione finita di \sim via operatori ausiliari di CCS finite	98
8.10. Assiomatizzazioni finite per rooted weak bisimilarity	98
9. Teoria dei punti fissi	99
9.1. Estremi	99
9.2. Reticolo	100
9.3. Monotonia	101
9.4. Potenza di funzione	101
9.5. Teorema di Knaster-Tarski	102
9.6. Calcolo dei punti fissi	103
9.7. Convergenza e divergenza	104
9.7.1. Sempre convergenza e sempre divergenza	105
9.8. Bisimulazione come punto fisso	106
9.8.1. Minimizzazione	107
10. Model checking	111
10.1. Logica Hennesy-Milner	111
10.2. Sintassi	111
10.3. Semantica	111
10.4. Negazione	113
10.5. Bisimilarità e soddisfacibilità	114
10.6. HML con ricorsione	115
10.6.1. Semantica per formule aperte	117
10.6.2. Semantica per formule aperte	118
10.6.3. Mutua ricorsione	120
10.6.3.1. Mutua ricorsione annidata	121
11. Mutua esclusione	122
11.1. Algoritmo Peterson	122
11.2. Algoritmo Hyman	123
11.3. Equivalenza	123
12. π calcolo	125
12.1. Bisimulazione weak barbed	126

13. Reti di petri	127
13.1. Reti di Petri P/T	127
13.2. Sistemi di reti P/T, marcatori e token game	128
13.3. Classi di reti P/T	130
13.4. Classi di sistemi di reti P/T	130
13.5. Sottorete dinamicamente raggiungibile	131
13.5.1. Sottorete dinamicamente ridotta	132
13.6. Sottorete staticamente raggiungibile	132
13.7. Proprietà decidibili	133
13.8. Isomorfismo	133
13.9. Interleaving marking graph	134
13.9.1. Equivalenza a tracce	134
13.10. Linguaggio rete Petri	135
13.11. Interleaving bisimilarità	135
13.12. Step Transition System	135
13.12.1. Fully concurrent STS	136
13.13. Reti di Petri non permissive NP/T	136
13.14. Turing completezza	137
13.15. Problema dell'ultimo uomo alzato	138

1. Introduzione

1.1. Modello sequenziale

Modello sequenziale (Pascal, ...) ha fine deterministico, dunque non si può fare qualcosa del tipo `while true do skip`. La semantica di un programma “classico” è una funzione parziale su `stores`. $\text{stores} \rightarrow \text{stores}$ che potremmo veder come $\llbracket _ \rrbracket : \text{Programmi} \rightarrow \text{Dominio di funzioni}$.

Composizionalità: per ogni operatore sintattico deve esistere un’operazione funzionale semantico.

Programmi equivalenti: stessa funzione calcolata. $Q_2 \simeq Q_2 \iff \llbracket Q_1 \rrbracket = \llbracket Q_2 \rrbracket$

Congruenza: equivalente preservata dagli operatori. $Q_1 \simeq Q_2 \iff$ per ogni operatore e per ogni programma P si ha che $Q_1; P \simeq Q_2; P$; $P \wedge P$; $Q_1 \simeq P$; Q_2 dove $_ ; _$ mette in sequenza due programmi.

1.2. Sistemi reattivi

Un sistema reattivo prende stimoli come input e reagisce con altri stimoli verso l'esterno. Un esempio classico, semplice (e anche sequenziale) è la macchina del caffè. Un altro esempio un po' più complesso è il PC: hai un ritorno nello schermo in base agli stimoli delle operazioni fatte. Un altro esempio sono i sistemi di controllo delle centrali nucleari che fanno controlli ad intervalli.

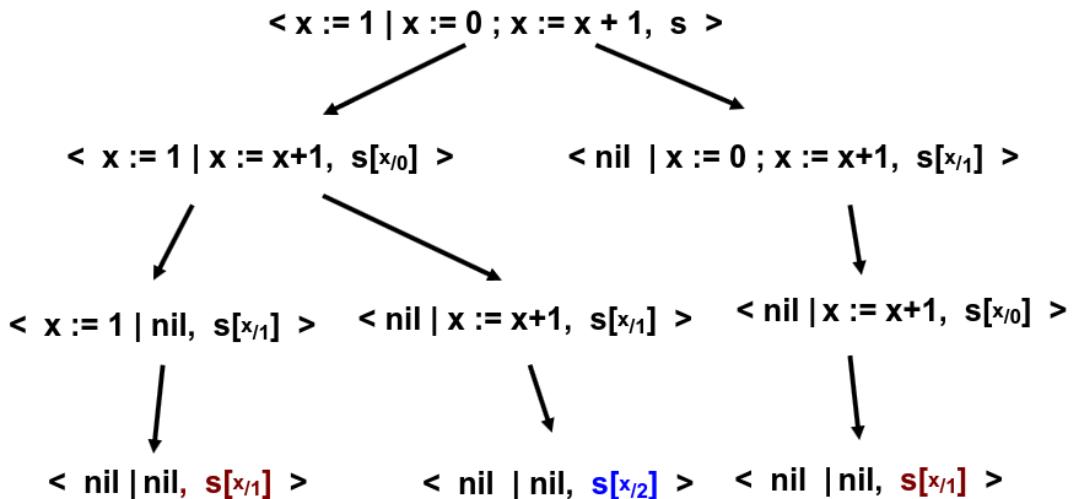
Si ha il concetto di non terminazione e non determinismo. In lato semantico si ha $\llbracket Q | P \rrbracket$ in cui Q parallelo P .

Presi

$$P = x := 0; x := x + 1$$

$$Q = x := 1$$

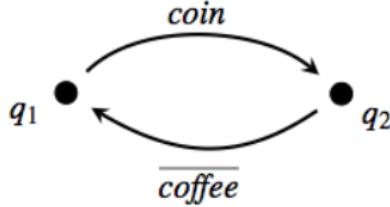
per rappresentare la semantica $\llbracket Q | P \rrbracket$ in cui, alla fine si ha $\llbracket P \rrbracket = \llbracket Q \rrbracket = \lambda s. s[x/1]$ si può costruire un tracciato operazionale come:



la semantica non è descritta mediante una funzione perché \simeq non è una congruenza. La nozione di equivalenza non è congruente rispetto all’operazione di parallelo. Il parallelo sintattico fra due programmi deve poter definire un parallelo semantico fra due funzioni. Nell’esempio si vede come non possano essere equiparati i programmi in modo parallelo perché il primo non modifica

la variabile in modo atomico. Il modello usato non è dunque una funzione ma descritto in modo simile agli automi. Si vede il cambiamento di stato in base ad una determinata azione.

Composizionalità: non c'è un modo per definire l'operazione semantica \parallel . Quindi la semantica non è compositiva.



Due atomi sono equivalenti quando riconoscono lo stesso linguaggio. L'equivalenza così descritta non va bene come semantica per un sistema reattivo. È riduttivo dire che le transizioni si riferiscono ad una singola azione, perché sarebbe una cosa più sequenziale. Si vede come il primo modello è parallelo mentre il secondo modello mette in sequenza l'esecuzione dei due programmi: quindi il parallelismo non è un concetto primitivo.

- **Automi (Sistemi di transizioni etichettate)** Una cosa sola può essere fatta ad ogni passo e lo stato è monolitico, ovvero non si vedono i componenti di un sistema distribuito. Parallelismo non primitivo.

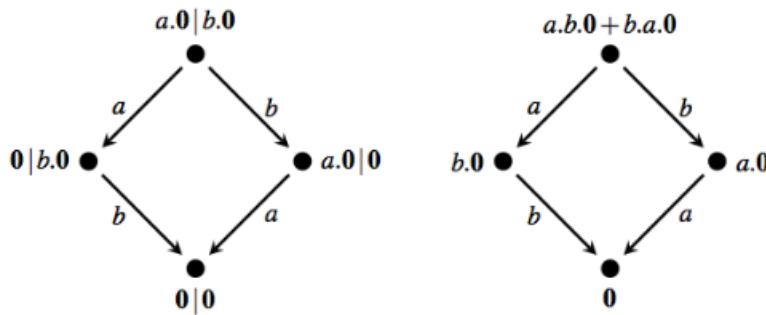


Figura 3: LTS semplice

- **Modello parallelo** Non vedo come il sistema è articolato nelle sue componenti parallele, ma il parallelismo è primitivo. È un automa con transizioni etichettate da multinsiemi di azioni. Non si vede la distribuzione del sistema.

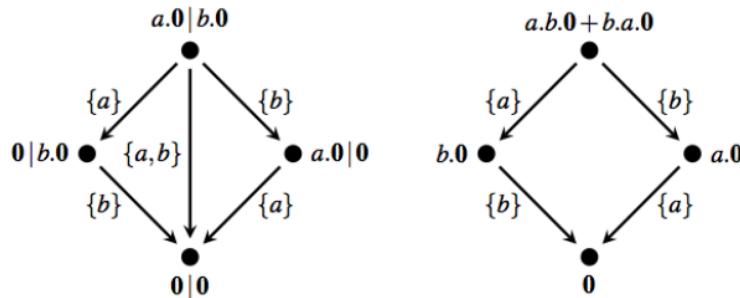


Figura 4: Modello parallelo

- **Modello distribuito** Si usano le reti di Petri. Lo stato del sistema è distribuito, che rappresenta il processo attivo nel momento: guardando quanti e quali sono i sistemi sequenziali

indipendenti eseguiti in modo parallelo. Lo stato globale del sistema è dato da quanti token (pallini) sono messi nei posti.

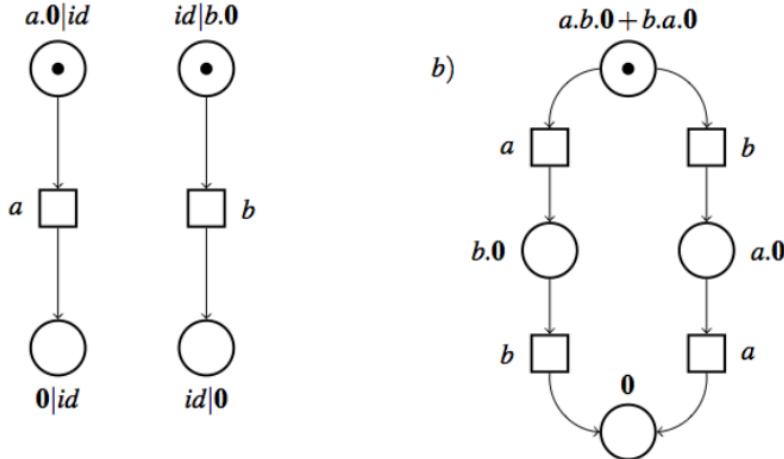


Figura 5: Modello distribuito

Numero di stati che possono essere infiniti. Nei LTS non ci sono stati iniziali e finali. Nei sistemi real time il tempo è continuo, a differenza degli stati negli LTS in cui invece è discreto. Nell'esempio del caffè, se volessimo modificarlo inserendo due coin lo stato q_2 non dovrebbe poter andare nello stato q_3 e restituire il caffè: al massimo chiede all'utente cosa vuole fare.

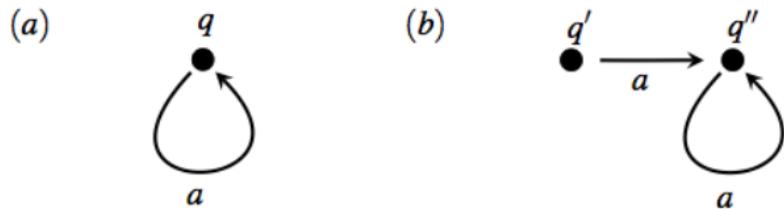
Dire che due sistemi sono equivalenti se fanno le stesse computazioni non va bene per i sistemi reattivi.

In base a quali esperimenti l'osservatore fa sulle black-box, escono delle equivalenze diverse. Tra due modelli di sistemi possiamo capire:

- Intercambiabilità: hanno stesso comportamento (non si parla di equivalenze a tracce di esecuzione).
- Equivalence-checking: confronto tra modello astratto (specifico) e modello concreto (implementazione).

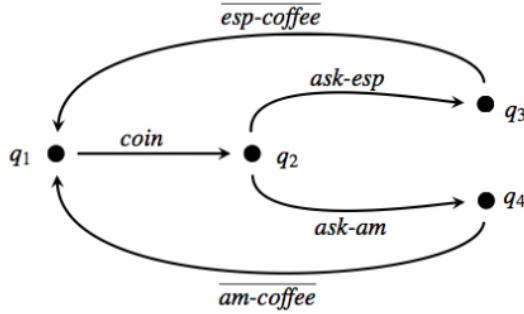
1.3. Equivalence checking

Possiamo considerare (b) una corretta implementazione della specifica (a).



1.4. Semantica a tracce

Dato l'esempio della macchinetta che sceglie la tipologia del caffè:



si vede come le tracce siano

$$\text{Tracce}(q_1) = \{\epsilon, \text{coin}, \text{coin ask-esp}, \text{coin ask-am}, \dots\}$$

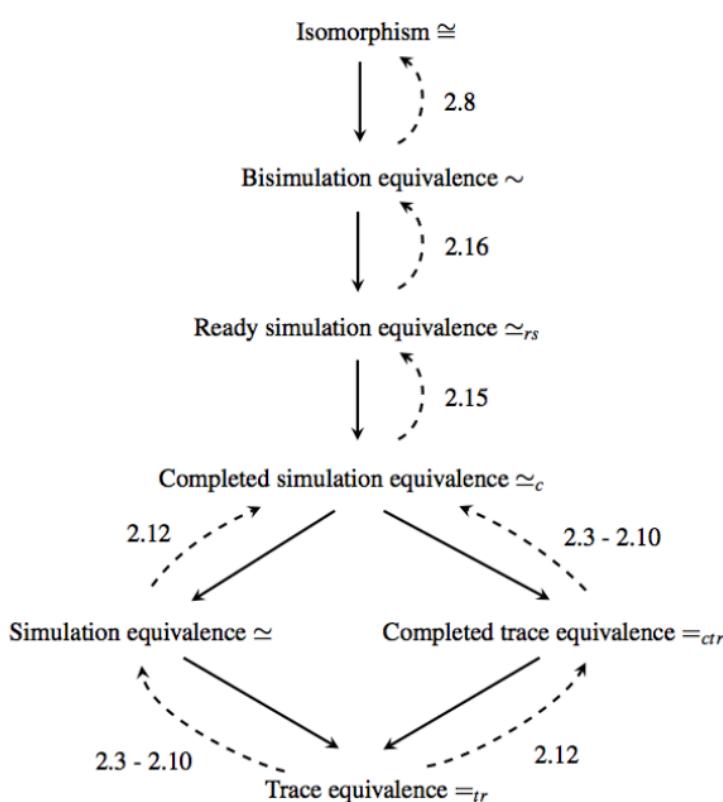
Due stati q_1 e q_2 sono equivalenti a tracce se $\text{Tracce}(q_1) = \text{Tracce}(q_2)$. Questo è troppo astratto.

1.5. Non determinismo

Il caso in cui è la macchina a scegliere la bevanda. Dunque, una volta inserita la moneta, è la macchina a scegliere quale dei due caffè erogare.

1.6. Equivalenze diverse

Possiamo usare l'isomorfismo negli LTS (ma è troppo concreta) oppure la bisimulazione.



lezione 1

Un'azione non osservabile è segnata da τ .

Linguaggi regolari non vanno bene nei sistemi reattivi: ad esempio le equivalenze a tracce danno spesso risposte sbagliate nel contesto; l'equivalenze a tracce sono PSPACE completi. La

bisimulazione per l'equivalenza, invece, ha buone proprietà ed è complessivamente molto veloce $O(m \log n)$.

\bar{a} è complementare ad a . Ricordando che il primo è output e il secondo input.

Non c'è l'equivalente dei criteri espressivi che si hanno nei linguaggi sequenziali (che si basano sulla Turing completezza).

2. Labeled Transition Systems (LTS)

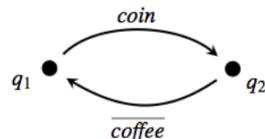
Essi non sono formalmente dei grafi. Un grafo con archi orientati è più concreto. Normalmente le transizioni vengono rappresentate come terne: se due archi hanno stesso stato di arrivo, fine ed etichetta, allora non va bene. Gli LTS possono avere stati infiniti contabili. Non sono nemmeno automi: gli automi sono a stati finiti, partizionano gli stati in finali/non finali e hanno stati iniziali. Negli LTS sono tutti stati finali.

Le etichette (o azioni) sono attività basiche usate per etichettare le transizioni. Preso un insieme di input di azioni contabile \mathcal{L} come a, b, \dots segniamo le co-azioni (output) come $\overline{\mathcal{L}}$ come $\overline{a}, \overline{b}, \dots$. L'insieme $\mathcal{L} \cup \overline{\mathcal{L}}$ è fatto dalle azioni invisibili α, β, \dots . Definiamo, dunque, $\text{Act} = \mathcal{L} \cup \overline{\mathcal{L}} \cup \{\tau\}$ ($\tau \notin \mathcal{L} \cup \overline{\mathcal{L}}$ e denota un'azione invisibile) come l'insieme delle azioni su un range di μ .

Un LTS è $TS = (Q, A, \rightarrow)$ con $Q \neq \emptyset$ insieme di stati contabili. $A \subseteq \text{Act}$ è insieme contabile delle azioni. μ è una qualche azione di A . Non si usa Act perché si possono scrivere solo le azioni minime usate poi realmente. $\rightarrow \subseteq Q \times A \times Q$ è la relazione di transizione, anch'essa contabile.

La transizione $(q, \mu, q') \in \rightarrow$ si ha q sorgente, q' target e μ etichetta di transizione. Parliamo di rooted LTS (TS, q_0) dove q_0 è lo stato iniziale.

Esempio

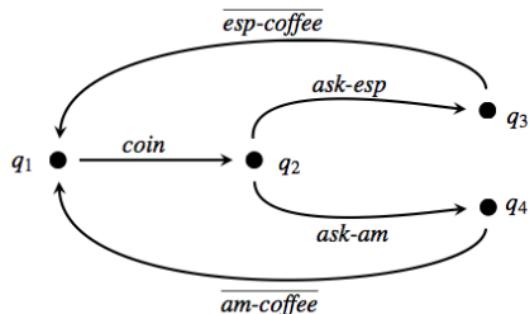


si può rappresentare mediante un

$$TS = (\{q_1, q_2\}, \{\text{coin}, \overline{\text{coffee}}\}, \{(q_1, \text{coin}, q_2), (q_2, \overline{\text{coffee}}, q_1)\}).$$

Esempio

Il minimo LTS che rappresenta



$$\begin{aligned}
TS = & (\{q_1, q_2, q_3, q_4\}, \\
& \{\text{coin, ask-esp, ask-am, } \overline{\text{am-coffee}}, \overline{\text{esp-coffee}}\}, \\
& \{(q_1, \text{coin}, q_2), (q_2, \text{ask-esp}, q_3), (q_2, \text{ask-am}, q_4), (q_3, \overline{\text{esp-coffee}}, q_1), (q_4, \overline{\text{am-coffee}}, q_1)\} \\
&)
\end{aligned}$$

Nell'insieme A posso inserire azioni mai usata nelle transizioni: questo mi permette di poter definire graficamente più rappresentazioni. La buona norma è quindi di usare la sua versione minima. Aggiungere stati cambia la rappresentazione: si inserisce un nodo isolato.

Un esempio è dato da $TS = (\{q\}, \{a\}, \{(q, a, q)\})$



ma chiaro definiamo la rappresentazione grafica come non unica, anche se si va a cercare una versione minima.

La notazione $q \xrightarrow{a} q'$ denota $(q, a, q') \in \rightarrow$.

$$\begin{aligned}
q \xrightarrow{a} & \iff \exists q'. q \xrightarrow{a} q' \\
q \not\xrightarrow{a} & \iff \nexists q'. q \xrightarrow{a} q' \\
q \rightarrow & \iff \exists a \in A. \exists q'. q \xrightarrow{a} q' \\
q \not\rightarrow & \iff \nexists a \in A. \nexists q'. q \xrightarrow{a} q'
\end{aligned}$$

equivalente a dire

$$\begin{aligned}
q \rightarrow & \iff \exists \mu \in A. q \xrightarrow{\mu} \\
q \not\rightarrow & \iff \forall \mu \in A. q \not\xrightarrow{\mu}
\end{aligned}$$

Una computazione (o percorso) in un LTS di lunghezza n da uno stato q ad uno q' è la sequenza di transizioni

$$q_1 \xrightarrow{\mu_1} q'_1 q_2 \xrightarrow{\mu_2} q'_2 q_3 \cdots q'_{n-1} q_n \xrightarrow{\mu_n} q'_n$$

tale che $q = q_1, q' = q'_n$ e $\forall i \in [1, n]. q'_i = q_{i+1}$. Dunque

$$q_1 \xrightarrow{\mu_1} q_2 \xrightarrow{\mu_2} \cdots q_n \xrightarrow{\mu_n} q_{n+1}$$

Con $n = 0$ si ha un percorso vuoto e $q = q' = q_1$. Con $\forall i \neq j$ con $j \in [1, n+1]. q_i \neq q_j$ si ha che il percorso è aciclico.

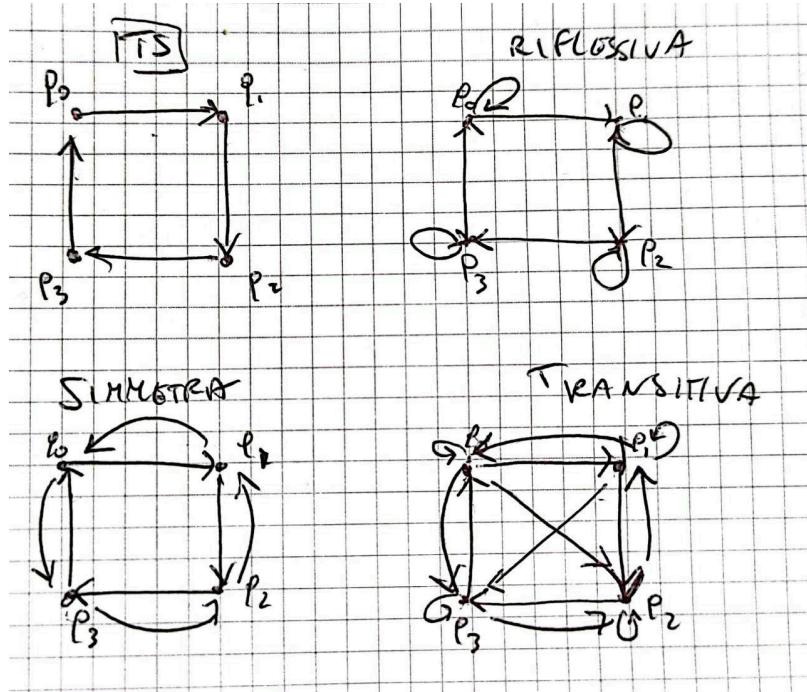
Se $q \rightarrow q'$ diciamo che esiste un percorso fra i due e dunque q' è raggiungibile da q . Tutti gli stati raggiungibili da Q sono presenti in Q_q . $Q_q = \{q' \in Q \mid q \xrightarrow{*} q'\}$. Una computazione può essere infinita.

$q \rightarrow$ non è in stato di deadlock. $q \not\rightarrow$ sì. Un rooted-LTS è aciclico se non vi sono cicli partendo dallo stato q_0 .

2.1. Proprietà

La chiusura riflessiva (e transitiva) di \rightarrow è \rightarrow' t.c. $\rightarrow \subseteq \rightarrow'$, ovvero la più piccola relazione riflessiva (e transitiva). La proprietà riflessiva definisce, dato uno stato q , la relazione $q \rightarrow q$. La proprietà simmetrica definisce, dati gli stati q e q' , la relazione $q \rightarrow q'$ e $q' \rightarrow q$. La proprietà transitiva definisce, dati gli stati q, q', q'' t.c. $q \rightarrow q'$ e $q' \rightarrow q''$ la relazione $q \rightarrow q''$.

Esempio dell'LTS TS .



\rightarrow^* è una transizione di sequenza.

2.2. Relazione di raggiungibilità

L'insieme di tutte le stringhe di A è denotato da A^* sul range di σ (include anche ϵ). Con delle proprietà:

- σ_1 concatenato a σ_2 è $\sigma_1\sigma_2$
- $\epsilon\sigma = \sigma\epsilon = \sigma$

La raggiungibilità è la relazione $\rightarrow^* \subseteq Q \times A^* \times Q$ definita come chiusura riflessiva e transitiva di \rightarrow .

$$\overline{q \xrightarrow{\epsilon} q}$$

$$\frac{q_1 \xrightarrow{\mu} q_2}{q_1 \xrightarrow{\mu} q_2}$$

$$\frac{q_1 \xrightarrow{\sigma_1} q_2 \quad q_1 \xrightarrow{\sigma_2} q_2}{q_1 \xrightarrow{\sigma_1\sigma_2} q_2}$$

Quindi $q_1 \xrightarrow{\sigma} q_2$ significa che lo stato q_2 è raggiungibile da q_1 se esiste una stringa σ t.c. $q_1 \xrightarrow{\sigma} q_2$.

Si vede come la definizione dello stato raggiungibile dal cammino è equivalente alla relazione di raggiungibilità \rightarrow^* .

$sort(q)$ è l'insieme delle azioni etichettate che vengono eseguite a partire da q .

$$sort(q) = \left\{ \mu \in A \mid \exists q'. q \xrightarrow{q} q' \xrightarrow{\mu} \right\}$$

Da questa definizione possiamo definire il rooted LTS $TS_q = (Q_q, sort(q), \xrightarrow{q}, q)$ detto anche “LTS raggiungibile da q ”. \xrightarrow{q} è la restrizione di \rightarrow in $Q_q \times sort(q) \times Q_q$. Quando tutti gli stati in Q sono raggiungibili da q_0 allora diciamo che il rooted LTS è ridotto. $TS = TS_{q_0}$.

2.3. Classi di LTS

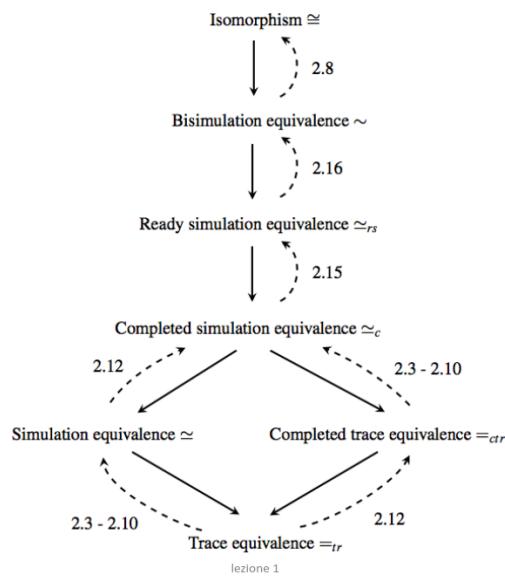
- **Finite** Se aciclico e Q e A sono insiemi finiti.
- **Finite state** Se Q e A sono insiemi finiti.
- **Boundedly branching** Se $\exists k \in \mathbb{N}$ t.c. $\forall q \in Q. T_q = \left\{ (q, \mu, q') \mid \exists \mu \in A. \exists q' \in Q. q \xrightarrow{\mu} q' \right\}$ ha cardinalità $\leq k$. k è detto grado di branching. Si guardano gli stati in uscita dello stato q .
- **Finitely branching** Se $T_q = \left\{ (q, \mu, q') \mid \exists \mu \in A. \exists q' \in Q. q \xrightarrow{\mu} q' \right\}$ è finito $\forall q \in Q$.
- **Image finite** Se $T_{q,\mu} = \left\{ (q, \mu, q') \mid \exists q' \in Q. q \xrightarrow{\mu} q' \right\}$ è finito $\forall q \in Q \wedge \forall \mu \in A$.
- **Deterministico** Se $q \xrightarrow{\mu} q'$ e $q \xrightarrow{\mu} q'' \Rightarrow q' = q'' \forall q \in Q \wedge \forall \mu \in A$

Finite state \Rightarrow Finite? No, perché il primo può avere cicli.

Boundedly branching \Rightarrow Finite-state? No, perché può avere un k grado ma infiniti stati.

3. Equivalenze forti

L'isomorfismo è considerato l'equivalenza più concreta. L'equivalenze a tracce è quella più astratta. Le prime tre di questa gerarchia sono computazionalmente poco pratiche, perché esponenziali.



3.1. Isomorfismo

Ha senso usarlo visto che gli LTS possono essere visti come grafi. Gli insieme degli stati è equipotente. L'isomorfismo preserva anche le etichette. Ogni isomorfismo è isomorfo con se stesso (riflessivo). È simmetrico perché lo si vede per un grafo con $n - 1$ nodi. È anche transitivo. Dati $TS_1 = (Q_1, A_1, \rightarrow_1)$ e $TS_2 = (Q_2, A_2, \rightarrow_2)$ un isomorfismo è una biezione $f : Q_1 \rightarrow Q_2$ in modo che $\forall q, q' \in Q_1 \wedge \mu \in A_1 \cup A_2$

$$q \xrightarrow{\mu} q' \iff f(q) \xrightarrow{\mu} f(q')$$

L'isomorfismo tra i due è segnato come $TS_1 \cong TS_2$. Se visto per un rooted LTS, va preservato anche lo stato iniziale $f(q_1) = q_2 \iff q_1$ è stato iniziale di TS_1 e q_2 è stato iniziale di TS_2 .

Troppo concreto come equivalenza perché non guarda come equivalenza il comportamento della relazione.

3.2. Equivalenza a tracce

Due automi sono equivalenti se riconosco le stesse stringhe: cammino da stato iniziale a uno di quelli finali che legge una sequenza di simboli. Non è utile per i sistemi reattivi, dato che, presa una sequenza σ si verifica se un LTS è capace di eseguire la sequenza interattivamente, più che se arriva alla fine. Se eseguito σ , riesce a fare anche tutti i suoi prefissi. La trasformazione che prende un NFA e lo trasforma in un equivalente DFA, è un'equivalenza a tracce.

Preso un LTS $TS = (Q, A, \rightarrow)$ e $q \in Q$, una traccia di q è una sequenza di azioni $\mu_1 \mu_2 \dots \mu_n$ t.c. esiste un percorso

$$q \xrightarrow{\mu_1} q_2 \xrightarrow{\mu_2} \dots \xrightarrow{\mu_n} q_n$$

$$Tr(q) = \{ \sigma \in A^* \mid \exists q' \in Q. q \xrightarrow{\sigma} q' \}$$

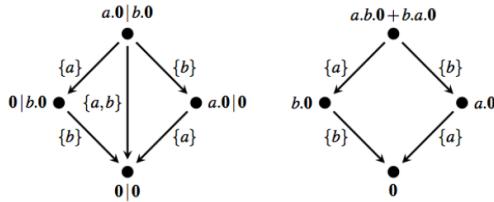
La traccia-equivalenza è segnata per due stati come $q_1 =_{tr} q_2$. Per il fatto che una traccia parte da uno stato, per un rooted LTS, mi basta far considerare $Tr(q_0)$ con q_0 stato iniziale.

3.2.1. Trace preorder

$$q_1 \leq_{tr} q_2 \iff Tr(q_1) \subseteq Tr(q_2)$$

come, ad esempio, $a.\mathbf{0} \leq_{tr} a.b.\mathbf{0}$ e $a.\mathbf{0} \leq_{tr} a.\mathbf{0} + b.\mathbf{0}$

Nell'esempio qui sotto si vede come i grafi che escono siano isomorfi. Se sono isomorfi, sono anche equivalenti per tracce.



Da un punto di vista espressivo per questo tipo di equivalenza, il non determinismo è irrilevante perché è sempre possibile creare una versione deterministica.

3.2.2. Equivalenza a tracce completa

Uno stato q è in deadlock se da q non vi è nessuna transizione, come nel caso di q_3, q_4, q_7 .



$Tr(a) = Tr(b) = \{\epsilon, a, ab\}$ però a va in deadlock in q_3 . Per vedere ciò bisogna osservare le tracce massimali (o complete).

Dato un $TS = (Q, A, \rightarrow)$ si definisce traccia completa per uno stato $q \in Q$ una sequenza di azioni $\mu_1 \dots \mu_n$ t.c. esiste un percorso

$$q \xrightarrow{\mu_1} q_2 \xrightarrow{\mu_2} \dots \xrightarrow{\mu_n} q_{n+1}$$

$$CTr(q) = \left\{ \sigma \in (A \cup \tau)^* \mid \exists q' \in Q. q \xrightarrow{\sigma} q' \wedge q' \not\rightarrow \right\}$$

L'equivalenza è segnata come $q_1 =_{ctr} q_2$.

Possono avere due comportamenti deadlock diversi. L'equivalenza a tracce completa è la semantica più astratta che osserva il deadlock.

Ahimè, ci sono casi in cui le tracce e tracce complete sono equivalenti, anche se non lo vorremo. La complessità è PSPACE completa. L'equivalenza e preorder a tracce sono utili per la verifica di proprietà di sicurezza, una proprietà che afferma che “qualcosa di brutto non accadrà mai”. Una proprietà di liveness afferma che “qualcosa di buono potrà forse accadere”.

3.3. Simulazione

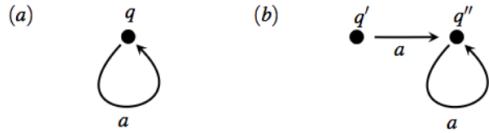
Relativamente più semplice alla costruzione delle tracce. La relazione di identità è una simulazione. È direzionata, il primo elemento nella coppia “comanda” cosa fare al secondo.

Dato un $TS = (Q, A, \rightarrow)$ diciamo che una simulazione è una relazione $R \subseteq Q \times Q$ t.c. se $(q_1, q_2) \in R \Rightarrow \forall \mu \in A :$

$$\forall q'_1 \cdot q_1 \xrightarrow{\mu} q'_1, \exists q'_2 \cdot q_2 \xrightarrow{\mu} q'_2 \wedge (q'_1, q'_2) \in R$$

Diciamo che q è simulato da q' e lo segniamo come $q \lesssim q'$ se esiste una simulazione R t.c. $(q, q') \in R$. Se i due stati sono equivalenti per simulazione se $q \lesssim q' \wedge q' \lesssim q$ allora $q \simeq q'$. Se lo sono allora esistono due simulazioni R_1 e R_2 t.c. $(q, q') \in R_1$ e $(q', q) \in R_2$. L'equivalenza per simulazione \Rightarrow l'equivalenza a tracce.

Presi



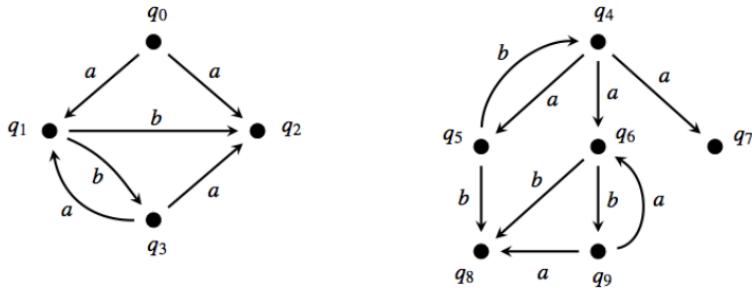
si vede come

$S_0 = \{(q, q)\}$ è una simulazione perché un cappio lo è sempre.

$S_1 = \{(q, q'), (q, q''), (q'', q)\}$ sì.

$S_2 = \{(q, q'), (q, q''), (q'', q')\}$ no, perché manca un nodo. Andrebbe bene solo se ci fosse anche (q'', q'') .

$S_3 = \emptyset$ sì, l'insieme vuoto è una simulazione.



la simulazione per la coppia (q_0, q_4) è $\{(q_0, q_4), (q_1, q_5), (q_2, q_7), (q_2, q_8), (q_3, q_4)\}$.

Proposizione Preso un $TS = (Q, A, \rightarrow)$. Per ogni $q, q' \in Q$, se $q \lesssim q'$ allora $q \leq_{tr} q'$. Dunque $Tr(q) \subseteq Tr(q')$.

Dimostrazione $q \lesssim q' \Rightarrow \exists R. R \subseteq Q \times Q : (q, q') \in R$. Si prova per induzione sul numero delle azioni σ .

- Caso $\sigma = \epsilon$

$q \xrightarrow{\epsilon}^* q$ e $q' \xrightarrow{\epsilon}^* q'$ e questo è un cappio, dunque $(q, q') \in R$.

- Caso $|\sigma| = n + 1$

Dunque esiste uno stato \bar{q} in cui una traccia σ' e un'azione μ t.c. $q \xrightarrow{\sigma'}^* \bar{q} \xrightarrow{\mu} q_1$ con $\sigma = \sigma' \mu$. Alla fine $\exists \bar{q}' : q \xrightarrow{\sigma}^* \bar{q}'$ con $(\bar{q}, \bar{q}') \in R$.

Visto che R è una simulazione, allora $\exists q'_1 : \bar{q} \xrightarrow{\mu} q'_1$ fa match con $\bar{q}' \xrightarrow{\sigma}^* q'_1$ con $(q_1, q'_1) \in R$. Mediamente cosa $\exists q'_1 : \bar{q} \xrightarrow{\sigma}^* q'_1$ che fa match con $\bar{q}' \xrightarrow{\sigma}^* q'_1$. Si vede che per $(q_1, q'_1) \in R$ si ha che $q \xrightarrow{\sigma}^* q_1$ fa match con $q' \xrightarrow{\sigma}^* q'_1$. ■

Corollario $q \simeq q' \Rightarrow q =_{tr} q'$

Non vale il viceversa perché, banalmente, il trace preorder non implica la simulation preorder (vedesi l'esempio della macchinetta del caffè che non deterministicamente porta a due stati diversi una volta inserita la moneta).

Data la natura di $q \lesssim q'$, ovvero che vale quando esiste una simulazione contenente la coppia (q, q') , allora possiamo dire che

$$\lesssim \bigcup \{R \subseteq Q \times Q \mid R \text{ è una simulazione}\}$$

$\lesssim \subseteq Q \times Q$ è la **più grande relazione di simulazione**.

Proposizioni

1. $\mathcal{J} = \{(q, q) \mid q \in Q\}$ è una simulazione.
2. Prese le simulazioni R_1 e R_2 , la composizione di esse $R_1 \circ R_2 = \{(q, q'') \mid \exists q'. (q, q') \in R_1 \wedge (q', q'') \in R_2\}$ è anch'essa una simulazione.
3. Anche $\bigcup_{i \in I} R_i$ è una simulazione.

Dimostrazione

1. $\forall q. q \xrightarrow{\mu} q'$ si ha che $q \xrightarrow{\mu} q'$ e quindi $(q', q') \in \mathcal{J}$.
2. Preso $(q, q'') \in R_1 \circ R_2 \Rightarrow \exists q' : (q, q') \in R_1 \wedge (q', q'') \in R_2$. Si ha $(q, q') \in R_1, q \xrightarrow{\mu} q_1 \Rightarrow \exists q_2 : q' \xrightarrow{\mu} q_2 \wedge (q_1, q_2) \in R_1$. Ma $(q', q'') \in R_2 \Rightarrow \exists q_3 : q'' \xrightarrow{\mu} q_3 \wedge (q_2, q_3) \in R_2$.

Dunque, per ogni coppia $(q, q'') \in R_1 \circ R_2 : q \xrightarrow{\mu} q_1 \Rightarrow \exists q_3. q'' \xrightarrow{\mu} q_3 \wedge (q_1, q_3) \in R_1 \circ R_2$.

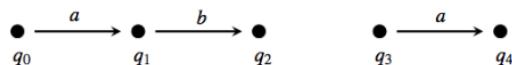
3. $(q, q') \in \bigcup_{i \in I} R_i \Rightarrow \exists j \in I. (q, q') \in R_j$. $q \xrightarrow{\mu} q_1 \Rightarrow \exists q_2. q' \xrightarrow{\mu} q_2 \wedge (q_1, q_2) \in R_j$. Dunque, $(q_1, q_2) \in \bigcup_{i \in I} R_i$ e $R_j \subseteq \bigcup_{i \in I} R_i$. Allora, $\bigcup_{i \in I} R_i$ è anch'essa una simulazione. \square

Preso una simulazione $S \subseteq Q \times Q, \forall (q_1, q_2) \in S. q_2 \not\rightarrow (\text{deadlock}) \Rightarrow q_1 \not\rightarrow$. Dunque, se alla simulazione S si aggiunge la coppia $(q, q'). q \not\rightarrow \Rightarrow S \cup (q, q')$ è comunque una simulazione.

$q \simeq q'$ è una relazione d'equivalenza perché è simmetrica (se $q \lesssim q'$ e $q' \lesssim q$ allora $q \simeq q'$).

La simulation preorder è la più grande simulazione che può essere definita su TS perché presa dalle unioni di tutte le simulazioni.

Esempio



Si vede come q_3 è simulato da q_0 . $\{(q_3, q_0), (q_4, q_1)\}$

Ma non vale il viceversa. $\{(q_0, q_3), (q_1, ?)\}$

Esempio

$a.\mathbf{0}$ non è simulato da $a.\mathbf{0} + b.\mathbf{0}$.

Esempio

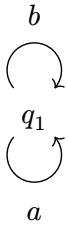


La simulazioni che vanno bene sono

$$\{(q_1, q_4), (q_2, q_5), (q_3, q_6)\} \text{ e } \{(q_4, q_1), (q_5, q_2), (q_6, q_3), (q_7, q_3)\}$$

ma quest'ultima non vale per la questione dei deadlock.

Un LTS con un solo stato che può essere simulato da qualsiasi altro LTS è un deadlock (minimale nel preordine). Un massimo nel simulation preorder è del tipo $q_1 \gtrsim q$ con q_1 def. come



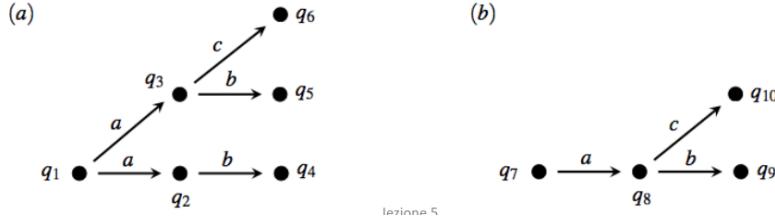
A livello di complessità il simulation preorder è incluso nel trace preorder. La simulazione fra due processi è efficiente ma non è sensibile ai deadlock. Vale che per ogni (q, q') in cui q' è un deadlock, allora anche q è un deadlock. Una **simulazione completa** è una relazione di simulazione $R \subseteq Q \times Q$ t.c. $\forall (q_1, q_2) \in R. q_1 \Rightarrow \Rightarrow q_2 \Rightarrow \Rightarrow$. In questo caso si dice che lo stato q è completamente simulato da q' e si denota come $q \lesssim_c q'$ se esiste una completa simulazione R t.c. $(q, q') \in R$. La completed simulation equivalence è $q \simeq_c q'$ e si ha se $q \lesssim_c q'$ e $q' \lesssim_c q$.

Secondo questo si vede come i due LTS sotto non vanno bene



dato che si hanno simulazioni $R_1 = \{(q_5, q_1), (q_6, q_2), (q_7, q_4)\}$ e $R_2 = \{(q_1, q_5), (q_2, q_6), (q_3, q_6), (q_4, q_7)\}$ dunque anche se sono simulation equivalence si vede come non dovrebbero esserlo causa il deadlock. Nel caso di completed simulation equivalence si vede come non passi il test per R_2 , dato che la coppia (q_3, q_6) non soddisfa il requisito aggiuntivo in cui q_3 è deadlock, ma q_6 no. R_1 è anche completed simulation equivalent.

La completed simulation equivalence non rispetta il “timing of choices”.



questi due sopra sono completed simulation equivalent, ma (q_2, q_8) non dovrebbero esserlo.

Quindi è troppo astratta.

3.4. Bisimulazione

Il primo (e il secondo) deve essere simulato nel secondo (e il primo) della coppia.

Dato un $TS = (Q, A, \rightarrow)$ diciamo che una bisimulazione è una relazione $R \subseteq Q \times Q$ e R^{-1} sono entrambe relazioni di simulazione. R è una relazione di bisimulazione se $(q_1, q_2) \in R \Rightarrow \forall \mu \in A :$

$$\forall q'_1. q_1 \xrightarrow{\mu} q'_1, \exists q'_2. q_2 \xrightarrow{\mu} q'_2 \wedge (q'_1, q'_2) \in R$$

$$\forall q'_2. q_2 \xrightarrow{\mu} q'_2, \exists q'_1. q_1 \xrightarrow{\mu} q'_1 \wedge (q'_1, q'_2) \in R$$

Se vi è una bisimulazione tra q e q' lo si denota come $q \sim q'$.

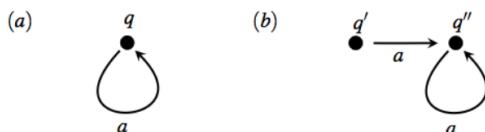
$R = \{(q, q') \mid (q, q') \in R\}$ è bisimulazione $\Leftrightarrow R^{-1} = \{(q', q) \mid (q, q') \in R\}$ è simulazione.

Se q e q' sono simulation equivalent $\Rightarrow \exists R_1, R_2$ simulazioni t.c. $(q, q') \in R_1 \wedge (q', q) \in R_2$ ma non è detto che $R_2 \equiv R_1^{-1}$.

$$q \sim q' \Rightarrow q \simeq_{rs} q' \Rightarrow q \simeq_c q' \Rightarrow q \simeq q'$$

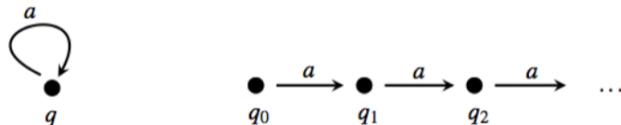
R è simmetrica $\Leftrightarrow R^{-1} = R$ e dunque questo $\Rightarrow R$ è bisimulazione.

- *Esempio*



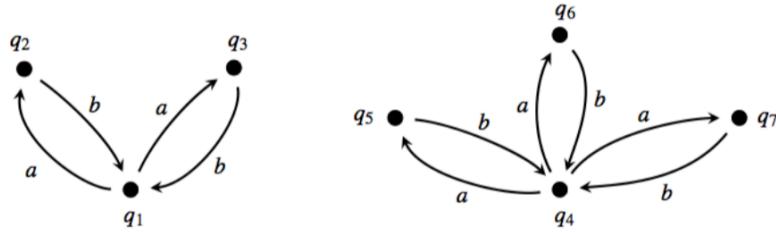
le bisimulazioni valide sono $R = \{(q, q'), (q, q'')\}, R_1 = \{(q, q'')\}, R_2 = \emptyset$. Le ultime due sono poco utili perché manca la coppia iniziale (q, q') .

- *Esempio*

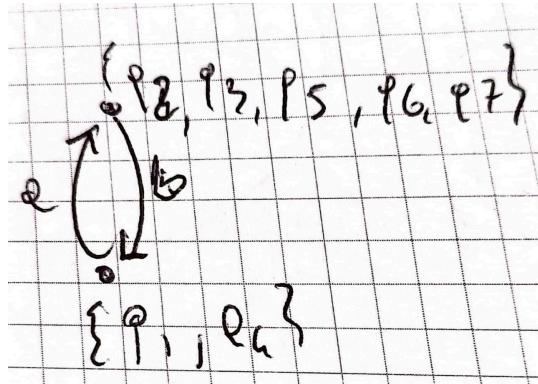


$Q \times Q$ è una bisimulazione. $R = \{(q, q_i) \mid i \in N\}$ è bisimulazione.

- *Esempio*



Una bisimulazione è $R = \{(q_1, q_4), (q_2, q_5), (q_2, q_6), (q_2, q_7), (q_3, q_5), (q_3, q_6), (q_3, q_7)\}$ ma si può minimizzare a

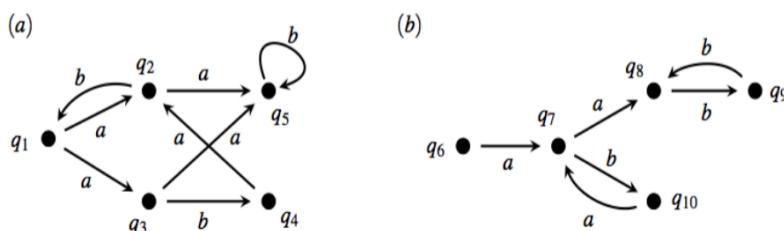


- *Esempio*



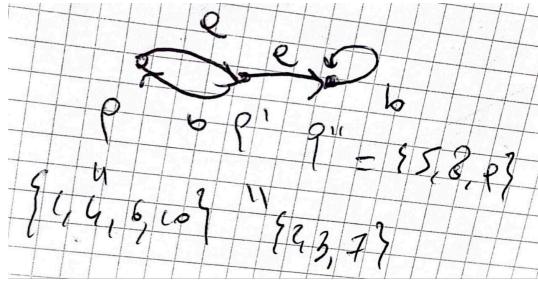
Non esiste una bisimulazione tale che contenta la coppia (q_1, q_7) perché (q_2, q_8) non è una coppia di bisimulazione.

- *Esempio*



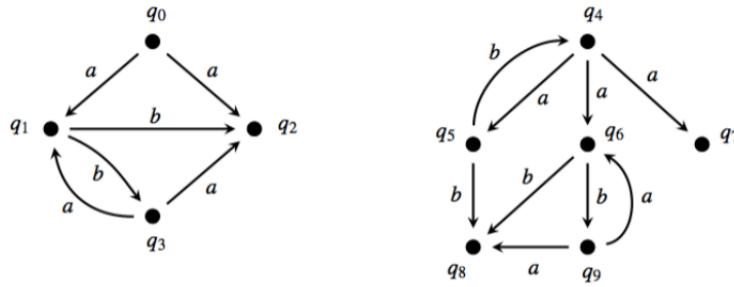
$$R = \{(q_1, q_6), (q_2, q_7), (q_3, q_7), (q_5, q_8), (q_1, q_{10}), (q_4, q_{10}), (q_5, q_9)\}$$

può essere minimizzato a



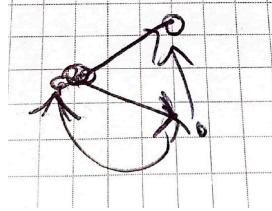
e vedere che vi è una bisimulazione che contiene la coppia (q_1, q_6) .

- *Esempio*



una bisimulazione possibile $R = \{(q_0, q_4), (q_1, q_5), (q_1, q_6), (q_2, q_7), (q_2, q_8), (q_3, q_9), (q_3, q_4)\}$

ma può essere minimizzato a



Proposizioni

1. $\mathcal{I} = \{(q, q) \mid q \in Q\}$ è una bisimulazione.
2. Dalla bisimulazione R , la relazione inversa $R^{-1} = \{(q', q) \mid (q, q') \in R\}$ è una bisimulazione.
3. Prese le bisimulazioni R_1 e R_2 , la composizione di esse $R_1 \circ R_2 = \{(q, q'') \mid \exists q'.(q, q') \in R_1 \wedge (q', q'') \in R_2\}$ è anch'essa una bisimulazione.
4. Anche $\bigcup_{i \in I} R_i$ è una bisimulazione.

$$\sim = \bigcup \{R \subseteq Q \times Q \mid R \text{ è una bisimulazione}\}$$

\sim è una relazione d'equivalenza perché:

1. È riflessiva: l'identità è inclusa in \sim .
2. È simmetrica: $q \sim q' \Rightarrow q' \sim q$ se $(q, q') \in R$ ma anche $(q', q) \in R^{-1}$ ma $R \subseteq \sim$, $q' \sim q$ per proposizione 3.
3. È transitiva: se $q \sim q'$ e $q' \sim q'' \Rightarrow q \sim q''$, ma presa per la similitudine $\exists R_1. \exists R_2. (q, q') \in R_1 \wedge (q', q'') \in R_2 \Rightarrow (q, q'') \in R_1 \circ R_2 \subseteq \sim$.

\sim è la più grande bisimulazione da proposizione 4.

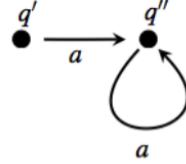
3.5. Bisimulazione ricorsiva (punto fisso)

Si definisce $\sim' \subseteq Q \times Q$ con $q_1 \sim' q_2 \iff \forall a \in A$

1. $\forall q'_1.q_1 \xrightarrow{a} q'_1, \exists q'_2.q_2 \xrightarrow{a} q'_2 \wedge q'_1 \sim' q'_2$
2. $\forall q'_2.q_2 \xrightarrow{a} q'_1, \exists q'_1.q_1 \xrightarrow{a} q'_1 \wedge q'_1 \sim' q'_2$

\sim è la più grande soluzione di questa definizione ricorsiva.

Presa



- $R = \{(q'', q'')\}$ è bisimulazione, ma non bisimulazione ricorsiva.
- $R_1 = \{(q', q'), (q'', q'')\}$ è bisimulazione e bisimulazione ricorsiva.
- $R_2 = \emptyset$ è bisimulazione e bisimulazione ricorsiva.
- La più grande bisimulazione e soluzione ricorsiva più grande è $S = \{q', q''\} \times \{q', q''\}$.

3.6. Bisimulazione up to \sim

Si definisce $R \subseteq Q \times Q$ con \sim se $(q_1, q_2) \in R \Rightarrow \forall a \in A$

1. $\forall q'_1.q_1 \xrightarrow{a} q'_1, \exists q'_2.q_2 \xrightarrow{a} q'_2 \wedge q'_1 \sim R \sim q'_2$
2. $\forall q'_2.q_2 \xrightarrow{a} q'_1, \exists q'_1.q_1 \xrightarrow{a} q'_1 \wedge q'_1 \sim R \sim q'_2$

Teorema Se R è bisimulazione up to \sim , allora $R \subseteq \sim$. Questo è vero anche se R non è una simulazione, le coppie messe in R sono bisimili.

Dimostrazione Se R è bisimulazione up to \sim , allora $\sim R \sim$ è bisimulazione. Per definizione di \sim , vale che $\sim R \sim \subseteq \sim$. Dato che $\text{Id} \subseteq \sim \Rightarrow R = \text{Id} R \text{ Id} \subseteq \sim R \sim$. \square

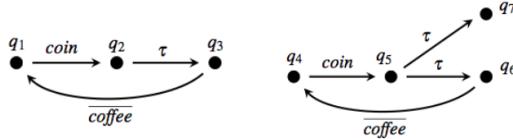
Presi n stati e m transizioni, la complessità della bisimulation equivalence è $O(m \log n)$. Inoltre è anche decidibile su stati infiniti.

4. Equivalenze deboli

Non sempre l'azione non osservabile τ va bene, dato che potrebbe essere un'azione osservabile che cambia stato.

“freccia doppia” \xrightarrow{a} è chiusura transitiva e riflessiva di \rightarrow ; lo fa anche a livello di computazione. Quindi $q \xrightarrow{a} q'$ vuol dire che potrebbe aver fatto delle azioni non osservabili prima e/o dopo. $q \xrightarrow{\varepsilon} q'$ è una computazione eventualmente vuota.

Le azioni non osservabili sono rappresentate con τ e può essere una rappresentare di un sync tra processi. Quando si trova in un punto di scelta (come un deadlock) diviene osservabile.



Definizione Per un LTS $TS = (Q, A \cup \{\tau\}, \rightarrow)$ dove $\tau \notin A$ la relazione $\Rightarrow \subseteq Q \times A^* \times Q$ è una chiusura debole riflessiva e transitiva di \rightarrow .

$$\frac{q_1 \xrightarrow{\alpha} q_2}{q_1 \xrightarrow{\alpha} q_2}$$

$$\frac{q_1 \xrightarrow{\tau} q_2}{q_1 \xrightarrow{\varepsilon} q_2}$$

$$\frac{}{q_1 \xrightarrow{\varepsilon} q_2}$$

$$\frac{q_1 \xrightarrow{\sigma_1} q_2 \quad q_2 \xrightarrow{\sigma_2} q_3}{q_1 \xrightarrow{\sigma_1 \sigma_2} q_3}$$

Quindi un percorso del tipo $q_1 \xrightarrow{\tau} q_2 \xrightarrow{\tau} \dots q_n \xrightarrow{\tau} q_{n+1}$ con $n \geq 0$ si può rappresentare come $q_1 \xrightarrow{\varepsilon} q_{n+1}$.

Quindi un percorso del tipo $q \xrightarrow{\alpha} q'$ va bene solo se $\exists q_1, q_2$ t.c. $q \xrightarrow{\varepsilon} q_1 \xrightarrow{\alpha} q_2 \xrightarrow{\varepsilon} q'$.

Quindi un percorso del tipo $q_1 \xrightarrow{\sigma} q_{n+1}$ con $\sigma = \alpha_1 \alpha_2 \dots \alpha_n$ va bene $\iff \exists q_2, \dots, q_n$ t.c. $q_1 \xrightarrow{\alpha_1} q_2 \xrightarrow{\alpha_2} \dots q_n \xrightarrow{\alpha_n} q_{n+1}$.

4.1. Tracce weak

Preso un LTS $TS = (Q, A \cup \{\tau\}, \rightarrow)$, con $\tau \notin A$. Un traccia debole di $q \in Q$ è una sequenza $\sigma \in A^*$ t.c. $q \xrightarrow{\sigma} q'$.

$$WTr(q) = \left\{ \sigma \in A^* \mid \exists q' \in Q. q \xrightarrow{\sigma} q' \right\}$$

L'equivalenza a tracce weak fra due stati si ha se $WTr(q_1) = WTr(q_2)$ e viene denotata come $q_1 =_{wtr} q_2$. Vale lo stesso per il LTS-rooted ma con l'aggiunta del nodo principale.

Weak trace preorder

$$q_1 \leq_{wtr} q_2 \iff WTr(q_1) \leq WTr(q_2)$$

Equivalenza strong weak \Rightarrow equivalenza weak

Può esservi LTS che fa azioni τ non osservabili rispetto a quello che si sta equiparando.

4.2. Tracce weak complete

Preso un LTS $TS = (Q, A \cup \{\tau\}, \rightarrow)$, con $\tau \notin A$, è un insieme di $q \in Q$ t.c.

$$WCTr(q) = \left\{ \sigma \in A^* \mid \exists q' \in Q. q \xrightarrow{\sigma} q' \wedge q' \xrightarrow{\alpha} \text{per ogni osservabile } \alpha \in A \right\}$$

L'equivalenza a tracce weak completa fra due stati si ha se $WCTr(q_1) = WCTr(q_2)$ e viene denotata come $q_1 =_{wctr} q_2$. Vale lo stesso per il LTS-rooted ma con l'aggiunta del nodo principale.

Non si può fare qualcosa di non osservabile, non necessita di deadlock.

Esempio

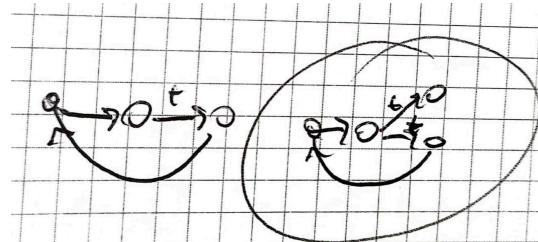
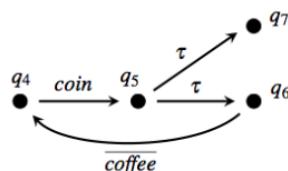


Figura 32: Non equivalent a tracce weak complete

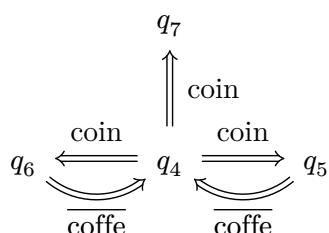
4.3. LTS τ -free

È un LTS con sole azioni osservabili in A .

Esempio



è equivalente a



ma si può minimizzare togliendo q_5

4.4. Simulazione weak

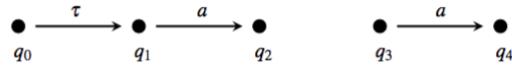
Preso un LTS $TS = (Q, A \cup \{\tau\}, \rightarrow)$, con $\tau \notin A$. Una simulazione weak è una relazione $R \subseteq Q \times Q$ t.c. $(q_1, q_2) \in R \Rightarrow \forall \alpha \in A$

- $\forall q'_1.q_1 \xrightarrow[\tau]{\alpha} q'_1, \exists q'_2.q_2 \xrightarrow[\varepsilon]{\alpha} q'_2 \wedge (q'_1, q'_2) \in R$
- $\forall q'_1.q_1 \rightarrow q'_1, \exists q'_2.q_2 \Rightarrow q'_2 \wedge (q'_1, q'_2) \in R$

Uno stato q è simulato weak da q' se denotato con $q \lessapprox q'$ ed esiste una simulazione weak R t.c. $(q, q') \in R$. L'equivalenza è denotata come $q \approx q'$.

$$\lessapprox = \bigcup \{R \subseteq Q \times Q \mid R \text{ è simulazione weak}\}$$

Esempio



è simulato da

1. $S_1 = \{(q_0, q_3), (q_1, q_4), (q_2, q_4)\}$ e quindi $q_0 \lessapprox q_3$.
2. $S_2 = \{(q_3, q_0), (q_4, q_2)\}$ e quindi $q_3 \lessapprox q_0$.

Nel punto 1 si ha che in $q_0 \xrightarrow{\tau} q_1$ il q_3 risponde stando fermo con $q_3 \xrightarrow{\varepsilon} q_3$.

Nel punto 2 si ha che in $q_3 \xrightarrow{a} q_4$, il q_0 risponde con $q_0 \xrightarrow{a} q_2$.

4.5. Simulazione strong

Preso un LTS $TS = (Q, A \cup \{\tau\}, \rightarrow)$. Una simulazione strong weak è una relazione $R \subseteq Q \times Q$ t.c. $(q_1, q_2) \in R \Rightarrow \forall \mu \in A \cup \{\tau\}$

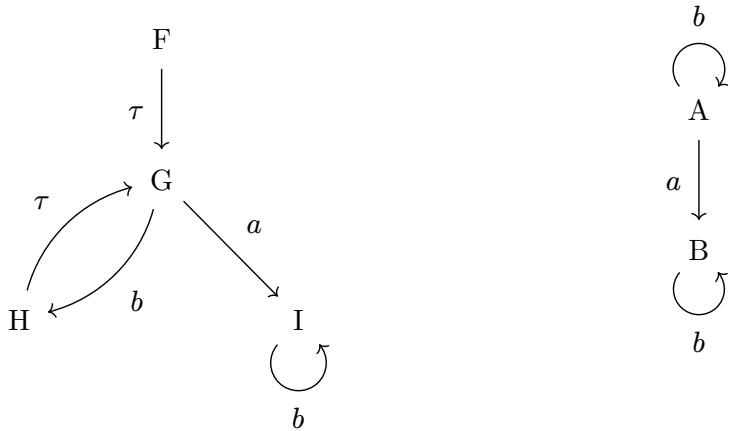
- $\forall q'_1.q_1 \xrightarrow{\mu} q'_1, \exists q'_2.q_2 \xrightarrow{\mu} q'_2 \wedge (q'_1, q'_2) \in R$

Uno stato q è simulato strong weak da q' se denotato con $q \lesssim q'$

Il τ viene trattata come se osservabile.

Se q è simulato strong da q' , allora q è simulato anche weak da q' . $q \lesssim q' \Rightarrow q \lessapprox q'$

Esempio



$\text{Id} = \{(F, F), (G, G), (H, H), (I, I)\}$ è una strong simulation (\Rightarrow weak simulation).

$R = \{(F, G), (G, G), (H, H), (I, I)\}$ è una weak simulation (ma non strong) perché:

1. gli ultimi tre elementi sono strong simulation (ovvi perché sono cappi).
2. in (F, G) se G risponde stando fermo, allora (G, G) è weak simulation.

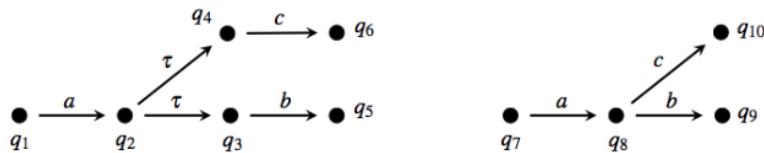
Si vede che $F \approx A$ perché da S_1 si vede $F \lesssim A$ e da S_2 si vede $A \lesssim F$.

$$S_1 = \{(F, \underbrace{A}_{\text{sta fermo}}), (G, A), (H, \underbrace{A}_{\text{sta fermo}}), (I, B)\}$$

$$S_2 = \{(A, F), (A, \underbrace{H}_{\text{resta fermo in loop con tau}}), (B, I)\}$$

Proprietà Guardando la lunghezza di \Rightarrow si può provare che, preso $TS = (Q, A \cup \{\tau\}, \rightarrow)$ con $\tau \notin A$ e data una simulaztion weak $R \subseteq Q \times Q$, si ha che se $(q_1, q_2) \in R \wedge q_1 \xrightarrow{\delta} q'_1$ allora $\exists q'_2$ t.c. $q_2 \xrightarrow{\delta} q'_2$ con $(q'_1, q'_2) \in R$ per un $\delta \in A \cup \{\epsilon\}$.

Esempio



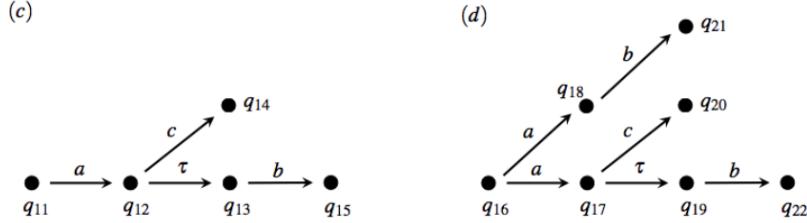
Vi è una weak simulation

$$R_1 = \{(q_1, q_7), (q_2, q_8), (q_3, q_8), (q_4, q_8), (q_5, q_9), (q_6, q_{10})\}$$

ma esiste anche una weak simulation

$$R_2 = \{(q_7, q_1), (q_8, q_2), (q_9, q_5), (q_{10}, q_6)\}$$

Esempio



Si vede come (q_{11}, q_{16}) sono equivalenti per simulazione weak perché se preso $q_{16} \rightarrow q_{17}$ si ha un LTS isomorfo a quello di q_{11} .

Il viceversa è dato da $\{(q_{16}, q_{11}), (q_{17}, q_{13}), (q_{18}, q_{13}), (q_{20}, q_{14}), (q_{19}, q_{13}), (q_{21}, q_{15}), (q_{22}, q_{15})\}$.

4.6. Simulazione weak completa

Una simulazione R è simulazione weak completa se $\forall (q_1, q_2) \in R$, se $q_1 \not\rightarrow$ allora $q_2 \xrightarrow{\alpha} \not\Rightarrow$ per ogni α osservabile.

Una equivalenza di questo tipo è $q_1 \cong_c q_2$ se $q_1 \lesssim_c q_2 \wedge q_2 \lesssim_c q_1$.

4.7. Bisimulazione weak

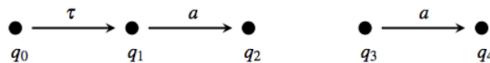
Preso un LTS $TS = (Q, A \cup \{\tau\}, \rightarrow)$, con $\tau \notin A$. Una bisimulazione weak è una relazione $R \subseteq Q \times Q$ t.c. R e R^{-1} sono simulazioni weak. Ovvero se $(q_1, q_2) \in R$ allora $\forall \alpha \in A$

- $\forall q'_1.q_1 \xrightarrow{\alpha} q'_1, \exists q'_2.q_2 \xrightarrow{\alpha} q'_2 \wedge (q'_1, q'_2) \in R$
- $\forall q'_1.q_1 \xrightarrow{\tau} q'_1, \exists q'_2.q_2 \xrightarrow{\epsilon} q'_2 \wedge (q'_1, q'_2) \in R$
- $\forall q'_2.q_2 \xrightarrow{\alpha} q'_2, \exists q'_1.q_1 \xrightarrow{\alpha} q'_1 \wedge (q'_1, q'_2) \in R$
- $\forall q'_2.q_2 \xrightarrow{\tau} q'_2, \exists q'_1.q_1 \xrightarrow{\epsilon} q'_1 \wedge (q'_1, q'_2) \in R$

Una bisimulazione weak fra due stati q e q' è denotata come $q \approx q'$.

$$\approx = \bigcup \{R \subseteq Q \times Q \mid R \text{ è bisimulazione weak}\}$$

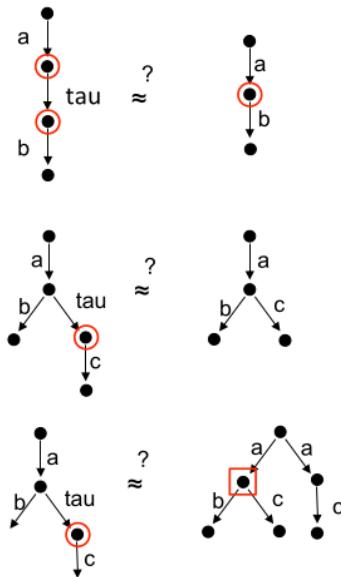
Esempio



è bisimulazione weak perché

- $S = \{(q_0, q_3), (q_1, q_3), (q_2, q_4)\}$ è simulazione weak, ma anche
 - $S^{-1} = \{(q_3, q_0), (q_3, q_1), (q_4, q_2)\}$
- e quindi $q_0 \approx q_3$.

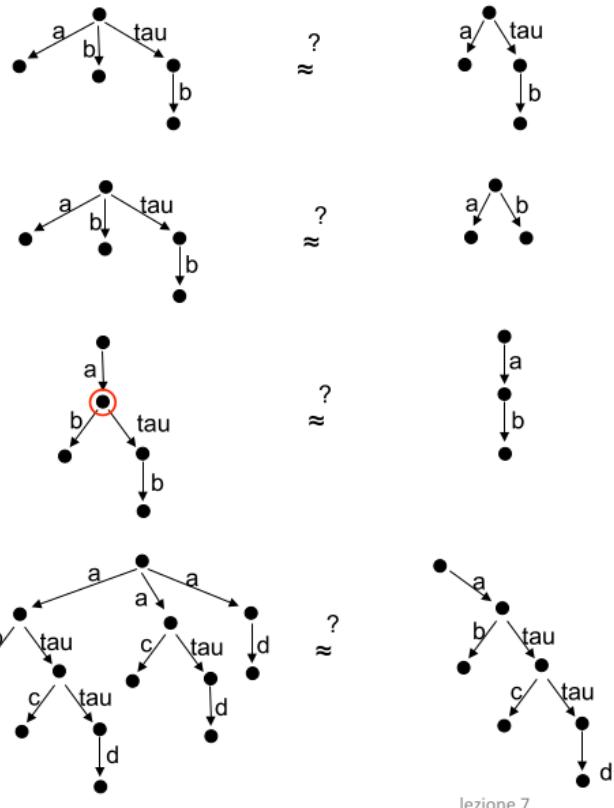
Esempio



Il primo è bisimulazione weak perché gli stati cerchiati in rosso sono equivalenti.

Il secondo non è bisimulazione weak perché lo stato cerchiato non ha equivalenti in quello a dx.

Il terzo non è bisimulazione weak perché lo stato quadrato non ha equivalenti.

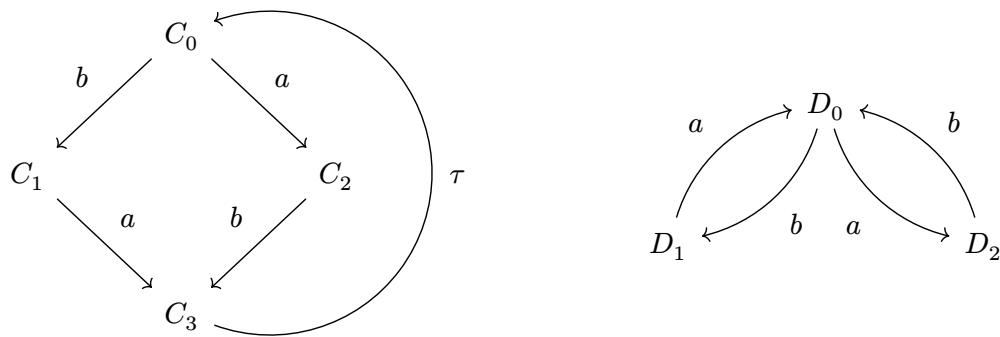


lezione 7

Figura 39: Solo il secondo non è bisimulazione weak

Proprietà Bisimulazione strong \Rightarrow bisimulazione debole. $q_1 \sim q_2 \Rightarrow q_1 \approx q_2$

Esempio

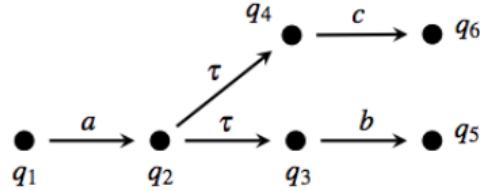


Si ha $S = \{(C_0, D_0), (C_1, D_1), (C_2, D_2), (C_3, D_0)\}$ è bisimulazione weak, ma non forte.

4.7.1. LTS τ free

Un LTS è di tipo τ free se nessuna transizione è etichettata τ .

Ad esempio non esiste un LTS senza transizione τ weak bisimile a questo sotto



perché senza lo stato q_2 non avremmo un LTS eequivalente (potrebbe stare solo fermo). Dunque le transizioni τ non sono sempre inosservabili e quindi non possono essere sempre tolte.

4.8. LTS astratto per bisimulazione weak

Dato un $TS = (Q, A \cup \{\tau\}, \rightarrow)$ dove $\tau \notin A$ si può definire un LTS associato $ATS = (Q, A \cup \{\epsilon\}, \Rightarrow'')$ con $\Rightarrow'' = \{(q, \delta, q') \mid \delta \in A \cup \{\epsilon\} \wedge (q, \delta, q') \in \Rightarrow\}$.

Per dimostrare che $q \approx q' \iff q \sim q'$ in ATS basta manipolare gli LTS (e transformarlo da LTS ad ALTS) e poi studiare la bisimulazione strong. La conversione LTS \rightarrow ALTS viene fatta mediante chiusura riflessiva, simmetrica e transitiva.

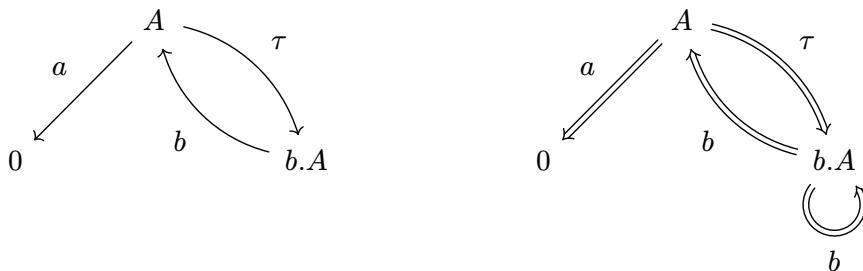
Dall'ALTS possiamo definire una bisimulazione weak come una relazione R tale che se $(q_1, q_2) \in R$ allora $\forall \delta \in A \cup \{\epsilon\}$ si ha che

- $\forall q'_1. q_1 \xrightarrow{\delta} q'_1, \exists q'_2. q_2 \xrightarrow{\delta} q'_2 \wedge (q'_1, q'_2) \in R$
- $\forall q'_2. q_2 \xrightarrow{\delta} q'_2, \exists q'_1. q_1 \xrightarrow{\delta} q'_1 \wedge (q'_1, q'_2) \in R$

Questa definizione eredita anche le proprietà per bisimulazione strong (come la più grande bisimulazione weak è relazione d'equivalenza oppure caratterizzazione di bisimilarità weak come massimo punto fisso di opportuna definizione ricorsiva).

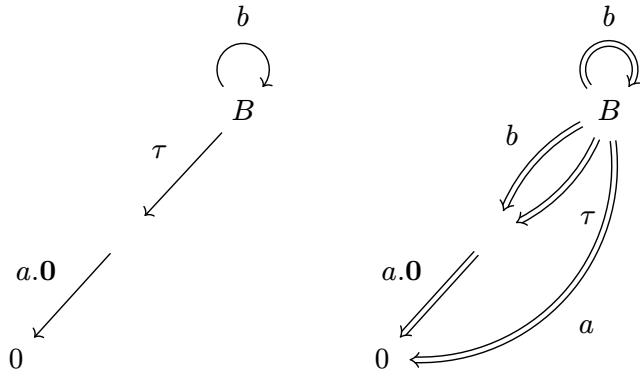
Esempio

Termine CCS $A = a.\mathbf{0} + \tau.b.A$, con a sx il suo LTS e a dx il suo ALTS.



Esempio

Termine CCS $B = \tau.a.\mathbf{0} + b.B$ non è weak bisimile ad A .



4.9. Proprietà varie

1. L'identità è bisimulazione weak.
2. L'inversa di una bisimulazione weak è una bisimulazione weak.
3. La composizione relazionale di due bisimulazioni weak è una bisimulazione weak. Se $(q_1, q_2) \in R$ e $q_1 \xrightarrow{k} q'_1$ allora $\exists q'_2. q_2 \xrightarrow{k} q'_2$ con $(q'_1, q'_2) \in R$.
4. L'unione arbitraria di bisimulazione weak è una bisimulazione weak. Quindi \approx è una relazione d'equivalenza e la più grande weak bisimulazione.

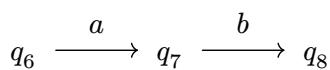
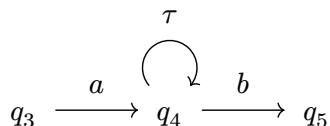
4.10. Divergenza

Presi ad esempio



q_1 è un livelock; q_2 è un deadlock; (q_1, q_2) è una coppia di bisimulazione weak.

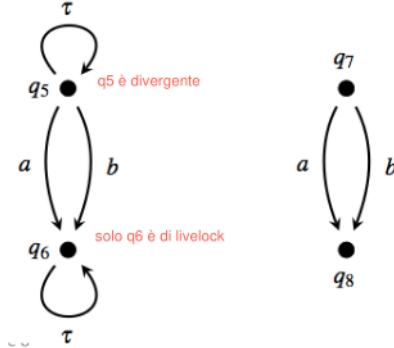
Un livelock può eseguire un τ e gli stati che raggiunge sono sempre elementi di S . Un $TS = (Q, A, \rightarrow)$ si può vedere come il più grande punto fisso del funzionale $G : \mathcal{P}(Q) \rightarrow \mathcal{P}(Q)$.



q_4 potenzialmente diverge, però è bisimulazione weak valida perché prima o poi potrebbe uscire dal loop τ .

La bisimulazione weak non distingue divergenza/livelock da deadlock. Si assume che ogni computazione sia *fair*: se b è possibile “infinitely often”, allora b prima o poi sarà scelta ed eseguita.

Definizione Uno stato q è divergente e denotato $q \uparrow$ se \exists un percorso infinito $q_1 \xrightarrow{\tau} q_2 \xrightarrow{\tau} \dots$ di transizioni τ con $q_1 = q$. Un LTS $TS = (Q, A \cup \{\tau\}, \rightarrow)$ è divergente-free se non c'è stato $q \in Q$ che è divergente. Uno stato q è un livelock se tutte le computazioni a partire da quello stato sono fatte da τ e non completano mai.



Può anche essere visto come il sottoinsieme di stati divergenti presi come il più grande punto fisso del funzionale $F : \mathcal{P}(Q) \rightarrow \mathcal{P}(Q)$

$$F(S) = \left\{ q_1 \mid \exists q_2. q_1 \xrightarrow{\tau} q_2 \wedge q_2 \in S \right\}$$

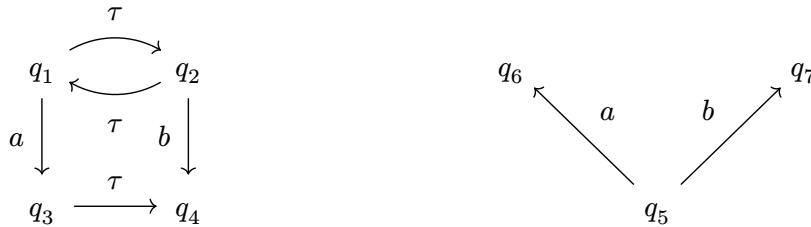
Esempio

Con $Q = \{q_1, q_2, q_3, q_4\}$ gli stati divergenti sono:

$$F^0(Q) = Q$$

$$F^1(Q) = \{q_1, q_2, q_3\} \text{ ovvero quelli che raggiungono uno stato facendo un } \tau.$$

$$F^2(Q) = \{q_1, q_2\} = F^3(Q) \text{ sono quelli che facendo un } \tau \text{ raggiungono gli stati sopra.}$$



Se volessimo trovare una weak bisimulation con la coppia (q_1, q_5) può essere data da

$$P = \{(q_1, q_5), (q_2, q_5), (q_3, q_6), (q_4, q_7), (q_4, q_6)\}$$

4.11. Bisimulazione weak up to \approx

Dato un LTS $TS = (Q, A \cup \{\tau\}, \rightarrow)$ con $\tau \notin A$, una bisimulazione weak up to \approx è $R \subseteq Q \times Q$ tale che se $(q_1, q_2) \in R$ allora $\forall \alpha \in A$:

- $\forall q'_1. q_1 \xrightarrow{\alpha} q'_1, \exists q'_2. q_2 \xrightarrow{\alpha} q'_2 \Rightarrow q'_1 \sim q'_2 \approx R \approx q'_2$
- $\forall q'_1. q_1 \xrightarrow{\tau} q'_1, \exists q'_2. q_2 \xrightarrow{\epsilon} q'_2 \Rightarrow q'_1 \sim q'_2 \approx R \approx q'_2$
- $\forall q'_2. q_2 \xrightarrow{\alpha} q'_2, \exists q'_1. q_1 \xrightarrow{\alpha} q'_1 \Rightarrow q'_1 \sim q'_2 \approx R \approx q'_2$

- $\forall q'_2.q_2 \xrightarrow{\tau} q'_2, \exists q'_1.q_1 \xrightarrow{\epsilon} q'_1 \wedge q'_1 \sim R \approx q'_2$

$q'_1 \sim R \approx q'_2$ è uguale a dire

1. $q'_1 \sim q''_1$
2. $(q''_1, q''_2) \in R$
3. $q'_2 \approx q'_2$

$\approx R \approx \subseteq \approx$ che poi anche $\text{Id } R \text{ Id } \subseteq \approx R \approx \subseteq \approx$ e questo vale perché $\text{Id } \subseteq \approx$.

4.12. Rooted weak bisimilarity

Dando per buono che la relazione $q \xrightarrow{\tau} q'$ è $q \xrightarrow{\epsilon} q_1 \xrightarrow{\tau} q_2 \xrightarrow{\epsilon} q'$.

Dato un LTS $TS = (Q, A \cup \{\tau\}, \rightarrow)$ due stati q_1, q_2 sono rooted weak bisimili $q_1 \approx^c q_2$ se $\forall \mu \in A \cup \{\tau\}$

- $\forall q'_1.q_1 \xrightarrow[\mu]{\mu} q'_1, \exists q'_2.q_2 \xrightarrow[\mu]{\mu} q'_2 \wedge q'_1 \approx q'_2$
- $\forall q'_2.q_2 \xrightarrow{\tau} q'_2, \exists q'_1.q_1 \xrightarrow{\epsilon} q'_1 \wedge q'_1 \approx q'_2$

q_2 non può star fermo, ma deve fare almeno un τ . Dopo il primo passo la relazione diviene una normale bisimilarità weak, quindi si può rispondere stando fermo. Usato per garantire la congruenza dell'operatore CCS +.

$\tau.a.\mathbf{0} \approx a.\mathbf{0}$ ma $\tau.a.\mathbf{0} + b.\mathbf{0} \not\approx a.\mathbf{0} + b.\mathbf{0}$ perché $\tau.a.\mathbf{0} + b.\mathbf{0} \xrightarrow{\tau} a.\mathbf{0}$ ma $a.\mathbf{0} + b.\mathbf{0}$ può rispondere solo con $a.\mathbf{0} + b.\mathbf{0} \xrightarrow{\epsilon} a.\mathbf{0} + b.\mathbf{0}$ ma gli stati $a.\mathbf{0}$ e $a.\mathbf{0} + b.\mathbf{0}$ non sono weak bisimili. $\tau.a.\mathbf{0} \not\approx^c a.\mathbf{0}$

Se $p \xrightarrow{\tau} q$ e $q \xrightarrow{\tau} p$ allora

- se $p \xrightarrow{\mu} p'$ allora $q \xrightarrow{\tau} p \xrightarrow{\mu} p'$ e $p \approx p'$
- se $q \xrightarrow{\mu} q'$ allora $p \xrightarrow{\tau} q \xrightarrow{\mu} q'$ e $q \approx q'$

$$p \approx^c q$$

$$p \approx q \iff \mu.p \approx^c \mu.q$$

$\forall \mu$.

1. Se $p \approx q$ allora $\mu.p \approx^c \mu.q$
2. Se $\mu.p \approx^c \mu.q$ allora $p \approx q$

5. Calculus of Communicating Systems (CCS)

È un linguaggio usato per descrivere modelli di un sistema in modo compositivo. Viene usato per fare prototipizzazione e analisi compositiva. Fornisce anche un supporto al ragionamento equazionale: invece di usare simili, basta eguagliare due termini. Se lo sono, sono anche bisimili.

Vi è una semantica operazionale capace di associare un CCS ad un LTS.

5.1. Sintassi

Azioni a, b, c, \dots come input in un insieme contabile \mathcal{L} .

Coazioni $\bar{a}, \bar{b}, \bar{c}, \dots$ come output in un insieme contabile \mathcal{L}' .

$\mathcal{L} \cup \mathcal{L}'$ insieme di azioni osservabili $\alpha, \beta, \gamma, \dots$. La coazione di α è $\bar{\alpha}$.

L'insieme delle azioni μ con l'azione non osservabile τ è definito da $\text{Act} = \mathcal{L} \cup \mathcal{L}' \cup \{\tau\}$.

Cons è l'insieme contabile di costanti di un processo disgiunto da Act .

Possiamo definire i processi separando quelli sequenziali da quelli generali:

$$P ::= \mathbf{0} \mid \mu.Q \mid P + P$$

$$Q ::= P \mid Q|Q \mid (\nu a)Q \mid C$$

Vi si applicano alcune convezioni perché una sintassi ambigua del tipo $a.b.\mathbf{0} + c.\mathbf{0}$ si rappresenta come $(a.(b.\mathbf{0})) + (c.\mathbf{0})$ oppure $a.((b.\mathbf{0}) + (c.\mathbf{0}))$. Pertanto si applica una priorità nel parsing:

$$(\nu a)p > \mu.p > x|y > x + y$$

Dunque quella sintassi ambigua denota $(a.(b.\mathbf{0})) + (c.\mathbf{0})$.

La composizione parallela e la scelta sono associative. Si possono usare gli operatori n -ari.

$$p_1 + p_2 + \dots + p_n = \sum_{1 \leq k \leq n} p_k$$

$$p_1|p_2|\dots|p_n = \prod_{1 \leq k \leq n} p_k$$

La restrizione invece si ha

$$(\nu a_1)(\nu a_2)\dots(\nu a_n)p = (\nu a_1 a_2 \dots a_n)p = (\nu L)p \text{ con } L = \{a_1, a_2, \dots, a_n\}$$

L'operatore nil è spesso omesso.

$$a.\mathbf{0} \mid b.\mathbf{0} = a|b$$

La somma con nil è omessa.

$$(\mathbf{0} + p) + \mathbf{0} = p$$

Si assume la somma “guardata”, dove entrambi gli addendi sono sequenziali. Pertanto, alcune operazioni non valide in CCS sono:

$$a.(A + B)$$

$$(b.\mathbf{0} + c.\mathbf{0}).a.\mathbf{0}$$

$$(\nu a)(b.\mathbf{0}) + c.\mathbf{0}$$

$$(\nu \tau)(a.\tau.\mathbf{0})$$

$$(A|c.\mathbf{0}) + a.B$$

$$\mathbf{0}.a.\mathbf{0}$$

$$(a.A + \bar{a}.\mathbf{0}).B$$

$$a.A.B$$

Esempi che vanno bene invece sono:

$$(\nu a)(a.\mathbf{0})$$

$$(A \mid (\nu c)B)$$

$$a.(\nu a)(a.\mathbf{0} \mid b.\mathbf{0})$$

5.2. Semantica

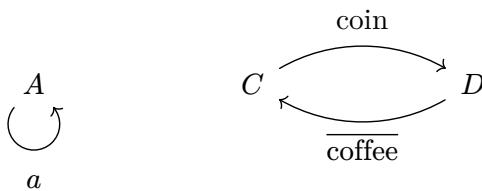
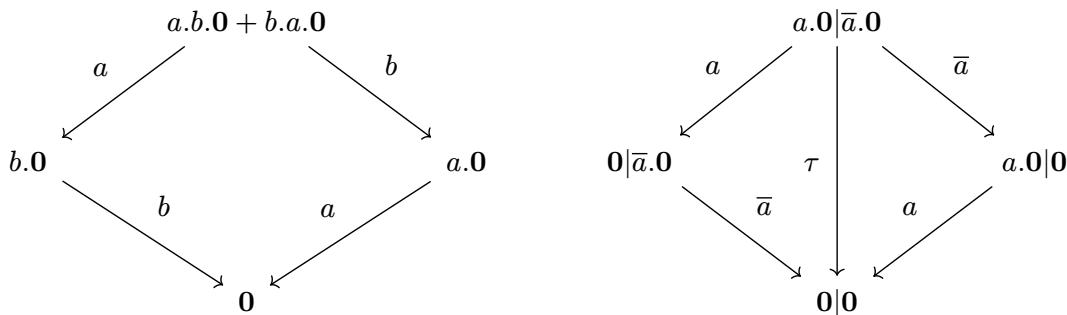
Vi sono operatori **dinamici** perché scompaiono durante l'esecuzione del processo:

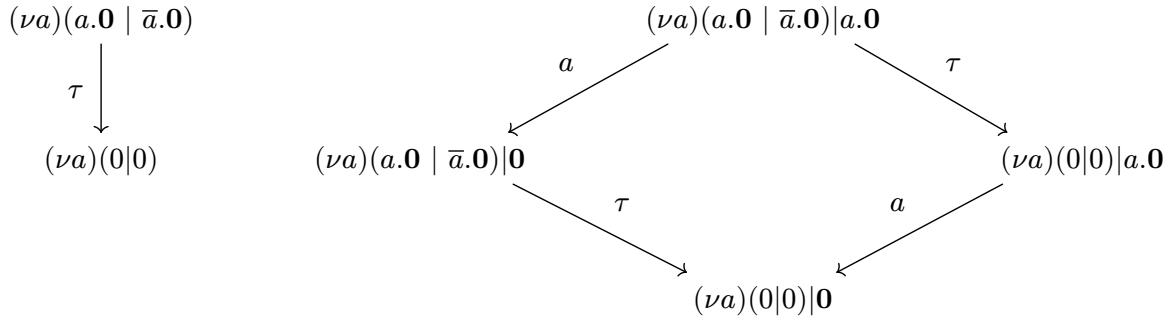
- **nil $\mathbf{0}$** : processo vuoto;
- **prefisso $a.p$** : a è l'azione e p il processo.
- **alternativa $p + q$** : si può eseguire l'azione da p o da q .

Oppure **statici** perché non svaniscono:

- **composizione parallela $p|q$** : processi indipendenti che girano in parallelo in modo asincrono o interagendo eseguendo sincronamenti I/O. La sincronizzazione è strettamente binaria. Le azioni τ vengono usate per stabilire comunicazione, ma qui nessuno può parteciparvi, dunque è strettamente lineare. Per fare τ le due azioni devono essere complementari. $a.\mathbf{0}|b.\mathbf{0} \equiv a.b.\mathbf{0} + b.a.\mathbf{0}$
- **restrizione $(\nu a)p$** : si rende privata un'azione a di un processo p . Quindi si dichiara che l'azione che il processo potrebbe eseguire non può essere offerta per interazione all'ambiente esterno. Dentro il processo, l'azione, può essere usata solo per sincronizzazioni interne.

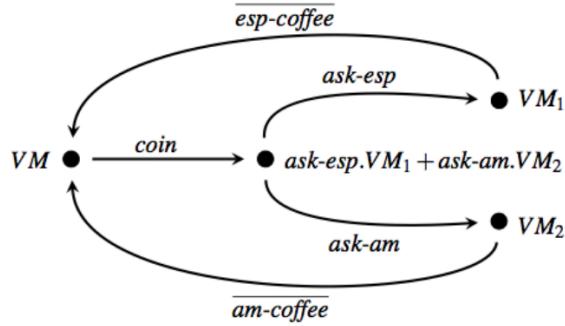
$$\mathbf{0} \xrightarrow{a} a.\mathbf{0} \quad a.b.\mathbf{0} \xrightarrow{a} b.\mathbf{0} \xrightarrow{b} \mathbf{0}$$





Un esempio più articolato per una macchinetta del caffè potrebbe essere quella di seguito.

$$\begin{aligned}
 VM &\stackrel{\text{def}}{=} \text{coin.}(\text{ask-esp.}VM_1 + \text{ask-am.}VM_2) \\
 \text{where } VM_1 &\stackrel{\text{def}}{=} \overline{\text{esp-coffee}.VM} \text{ and } VM_2 \stackrel{\text{def}}{=} \overline{\text{am-coffee}.VM}.
 \end{aligned}$$



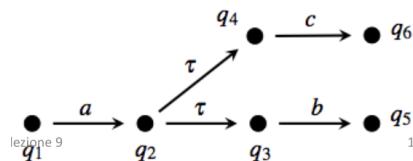
Esempio

- **Competizione**

Si esprime la competizione tra processi indipendenti per l'uso di risorse condivise (una forma di scelta interna). Preso ad esempio

$$(\nu d)(a.(d.b.\mathbf{0} \mid d.c.\mathbf{0}) \mid \bar{d}.\mathbf{0})$$

C'è una biforcazione dopo la a , ovvero l'iniziale processo sequenziale. Quella esecuzione attiva due processi concorrenti in competizione per la risorsa condivisa $\bar{d}.\mathbf{0}$.



- **Sequenzializzazione**

Esprime un'azione che precede un processo.

Con parallelo e restrizione si può esprimere una forma di “un processo che precede un processo”.

Ad esempio, $(\nu d)(a.(b.d.\mathbf{0} + c.d.\mathbf{0}) \mid \bar{d}.q)$ si descrive un processo in cui q è attivato solo quando quello a sx viene completato (con ab o ac).

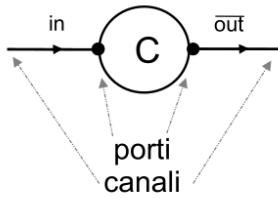
Ad esempio, $(\nu d)(b.d.0 \mid c.d.0) \bar{d}.\bar{d}.q$ si descrive un processo in cui q è attivato solo dopo che i due processi $b.d.0$ e $c.d.0$ sono terminati.

Si definire un nuovo operatore $p; q$.

5.3. Flow graph

Gli operatori statici determinano l'architettura di interconnessione. Un processo sequenziale p è rappresentato da un cerchio; un'azione determina un port su cui è possibile instaurare un canale di comunicazione con porti su altri processi sequenziali. Se il nome è ristretto tra due processi sequenziali, allora il canale instaurato tra i due è privato.

Ad esempio, un buffer ad una posizione è rappresentato mediante il flow graph seguente



in cui in è il canale di input, e $\overline{\text{out}}$ quello di output. In CCS lo si esprime come

$$C = \text{in}.C'$$

$$C' = \overline{\text{out}}.C$$

ma potrebbe essere espresso con una sola costante dato che è un solo processo sequenziale.

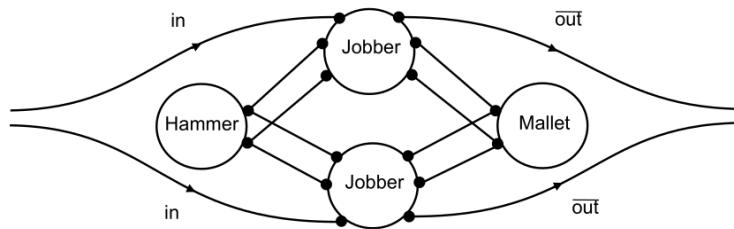
$$C = \text{in}.\overline{\text{out}}.C$$

Oppure una versione di CCS “value-passing” definita come:

$$C = \text{in}(x).C'(x)$$

$$C'(x) = \overline{\text{out}(x)}.C$$

Esempio



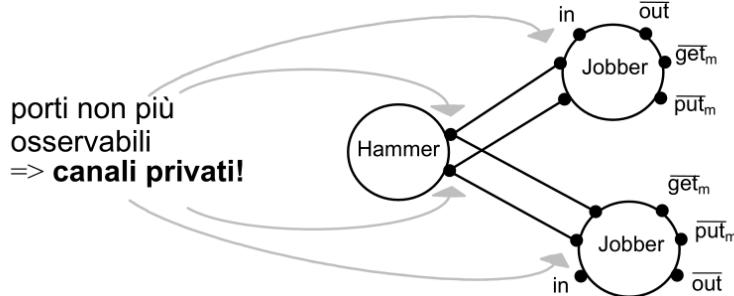
Due “jobber” possono competere per l’uso di una mazza e martello.

Si potrebbe usare una versione di CCS “value-passing” con if-then-else per reappresentare i continui get e put da parte dei processi.

Facendo $\text{Jobber} \mid \text{Hammer}$ si legano insieme i porti complementari, creando dei canali di comunicazione. Quest’azione è commutativa.

Per impedire che dei jobber utilizzino altri hammer bisogna restringere le azioni.

$$(\nu \text{get}_h)(\nu \text{put}_h)(\text{Jobber} \mid \text{Hammer} \mid \text{Jobber})$$



quindi l’esempio può essere risolto come

$$\text{Jobshop} = (\nu \{\text{get}_h, \text{put}_h, \text{get}_m \mid \text{put}_m\})(\text{Jobber} \mid \text{Jobber} \mid \text{Hammer} \mid \text{Mallet})$$

Esso è weak bisimile a $\text{StrongJobber} = \text{in(job).out(done(job))}.\text{StrongJobber}$

5.3.1. Operatore di linking / pipeline

Mette in parallelo due processi con connessione privata su nomi non complementari: l’output del primo diviene input del secondo.

$$p \hat{\wedge} q$$

Per p e q è necessaria la ridenominazione, con un nuovo nome d .

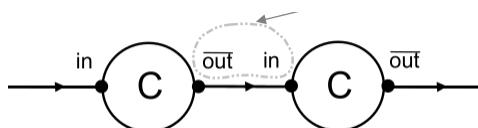
$$p \hat{\wedge} q = (\nu d)(p\{d/a\} \mid q\{d/b\})$$

Esempio

Con questo operatore si può definire un buffer a due posizioni

$$A = \text{in}(x).\text{in}(y).\overline{\text{out}}(x).\overline{\text{out}}(y).A$$

il quale riempie il buffer e poi lo svuota rispettando l’ordine.



$$C \hat{\wedge} C = (\nu d)(C\{d/\text{out}\} \mid C\{d/\text{in}\})$$

$$C_{\{d/\text{out}\}} \stackrel{\text{def}}{=} \text{in}.d.C_{\{d/\text{out}\}}$$

$$C_{\{d/\text{in}\}} \stackrel{\text{def}}{=} d.\overline{\text{out}}.C_{\{d/\text{in}\}}$$

5.4. Costanti

L'insieme delle costanti in un processo CCS viene calcolato usando una funzione definita come:

$$\text{Const}(\mathbf{0}) = \emptyset$$

$$\text{Const}(\mu.p) = \text{Const}(p)$$

$$\text{Const}((\nu a)p) = \text{Const}(p)$$

$$\text{Const}(p_1 + p_2) = \text{Const}(p_1) \cup \text{Const}(p_2)$$

$$\text{Const}(p_1 | p_2) = \text{Const}(p_1) \cup \text{Const}(p_2)$$

$$\text{Const}(A) = \begin{cases} \{A\} & \text{if } A \text{ undefined} \\ \{A\} \cup \text{Const}(q) & \text{if } A = q \end{cases}$$

Un termine $\text{Const}(p)$ finito è chiamato finitario, ed usa molto i processi finiti con solo costanti.

Esempio

$$C_1 \stackrel{\text{def}}{=} \text{coin}.C_2$$

$$C_2 \stackrel{\text{def}}{=} \overline{\text{coffee}}.C_1$$

Si trovano

$$\text{Const}(C_1) = \{C_1\} \cup \text{Const}(\text{coin}.C_2) = \{C_1\} \cup \text{Const}(C_2) = \{C_1, C_2\} \cup \text{Const}(\overline{\text{coffee}}.C_1) = \{C_1, C_2\} \cup \text{Const}(C_1)$$

ma questo è ricorsivo!

Quando il processo per cui si calcolano le costanti è finitario c'è un algoritmo per il calcolo della funzione che usa una funzione ausiliaria δ che prende come parametro anche un insieme I formato da costanti note:

$$\delta(\mathbf{0}, I) = \emptyset$$

$$\delta(\mu.p, I) = \delta(p, I)$$

$$\delta((\nu a)p) = \delta(p, I)$$

$$\delta(p_1 + p_2, I) = \delta(p_1, I) \cup \delta(p_2, I)$$

$$\delta(p_1 | p_2, I) = \delta(p_1, I) \cup \delta(p_2, I)$$

$$\delta(A, I) = \begin{cases} \emptyset & A \in I \\ \{A\} & A \notin I \wedge A \text{ undef} \\ \{A\} \cup \delta(p, I \cup \{A\}) & A \notin I \wedge A \stackrel{\text{def}}{=} p \end{cases}$$

Una costante A è detta **definita** se possiede un'equazione di definizione $A = p$.

Un termine CCS q è detto **fully defined** se tutte le sue costanti in $\text{Const}(q)$ sono definite.

5.4.1. Costanti guardate

Una occorrenza di una costante A in un termine p è detta **fortemente guardata** se occorre in un sottotermine $\mu.q$ di p . Ad esempio, l'unica occorrenza di B in $a.\mathbf{0}|a.B$ è fortemente guardata, mentre in $A|a.A$ quella a sinistra non lo è.

Una costante definita $A = p$ è guardata se ogni occorrenza di A in p è fortemente guardata e, inoltre, ogni altra costante B che occorre in p è fortemente guardata in p oppure è guardata. Ad esempio, $B = b.B$ e $A = (a.A|B)|c.C$ sono guardate; $C = a.\mathbf{0}|C$, $D = F$ e $F = D$ non lo sono.

Un termine guardato genera un LTS finitely-branching. Ad esempio, $C = a.\mathbf{0}|C$ ne genera uno, addirittura, image finite.

Le costanti guardate garantiscono unicità di soluzione, modulo di equivalenza per bisimulazione \sim , di equazioni di processi del tipo $X \sim E(X)$ con $E(_)$ un contesto. Ad esempio, $X \sim \tau.X + a.\mathbf{0}$ dove X occorre guardata da τ , ha unica soluzione $A = \tau.A + a.\mathbf{0}$

5.5. Processi

L'insieme \mathcal{P} dei processi CCS è dato da tutti i termini p tali che ogni costante A in $\text{Const}(p)$ è definita e guardata. (*Una costante definita è anche guardata*)

Un processo CCS p è finitario se $\text{Const}(p)$ è finito **Finitary CCS**. In questo caso il processo può essere visto come sistema formale finito (un po' come un DFA). Però, il linguaggio finitary CCS, non è un sistema formale finito.

In realtà, la sintassi CCS dovrebbe essere definita su un insieme finito di costanti C e azioni L nel modo $\text{CCS}_{C,L}$. In questo caso si ha un processo al suo interno che è un sistema formale finito, ma anche l'insieme infinito di tutti i processi in esso costituisce un sistema formale finito.

Un **calcolo finito** è dunque il linguaggio $\text{CCS}_{C,L}$ con C ed L finite ma molto grandi. Se non è importante il nome delle azioni e costanti, si può indicare $\text{CCS}(h,k)$ il linguaggio con h costanti e k azioni al massimo.

5.6. Semantica Operazionale Strutturata

Definita mediante albero di inferenza, ogni premessa/conclusione è una transizione e side-condition è un predicato che deve essere vero per poter applicare la regola. Con $n = \mathbf{0}$ si ha un assioma.

$$\begin{array}{c} \text{(Side effect)} \quad \text{Premessa 1} \quad \text{Premessa n} \\ \hline & \text{Conclusione} \end{array}$$

Il **transition system** per CCS è la tripla $(\mathcal{P}, \text{Act}, \rightarrow)$ con $\rightarrow \subseteq \mathcal{P} \times \text{Act} \times \mathcal{P}$ ed è la minima relazione generata dall'assioma e dalle regole d'inferenza.

- **Somma**

$$\begin{aligned} (\text{Sum}_1) \quad & \frac{p \xrightarrow{\mu} p'}{p + q \xrightarrow{\mu} p'} \\ (\text{Sum}_2) \quad & \frac{q \xrightarrow{\mu} q'}{p + q \xrightarrow{\mu} p'} \end{aligned}$$

In caso di $\mathbf{0} + a.b \xrightarrow{a} b$ il processo $\mathbf{0}$ non può comunque essere scelto.

- **Prefisso**

$$(\text{Pref}) \frac{}{\mu.p \xrightarrow{\mu} p}$$

Il quale è un operatore dinamico, ecco perché scompare.

- **Costante**

$$(\text{Cons}) \frac{p \xrightarrow{\mu} p'}{C \xrightarrow{\mu} p'}$$

con $C \stackrel{\text{def}}{=} p$. Non viene definita induttivamente.

- **Parallelo**

$$(\text{Par}_1) \frac{p \xrightarrow{\mu} p'}{p|q \xrightarrow{\mu} p'|q}$$

$$(\text{Par}_2) \frac{q \xrightarrow{\mu} q'}{p|q \xrightarrow{\mu} p|q'}$$

Si ha l'**interleaving** perché i due processi mescolano le loro azioni in tutti i modi possibili perché non si fa ipotesi sulla velocità di essi.

- **Sincronizzazione**

$$(\text{Com}) \frac{p \xrightarrow{\alpha} p' \quad q \xrightarrow{\bar{\alpha}} q'}{p|q \xrightarrow{\tau} p'|q'}$$

È strettamente binaria (point-to-point) perché la transizione risultante è τ , e dunque non può essere usata da un terzo processo per fare sync. Non è possibile in questo CCS fare sync multi-way. La comunicazione async viene fatta con buffer intermedio.

- **Restrizione**

$$(\text{Res}) \frac{p \xrightarrow{\mu} p'}{(\nu a)p \xrightarrow{\mu} (\nu a)p'}$$

con $\mu \neq (\nu a)p'$.

Non ha effetto su transizioni di p ma impedisce qualunque transizione etichettata a o \bar{a} .

Esempio

Si calcola l'albero di derivazione per

$$(\nu a)((a.E + b.\mathbf{0})|\bar{a}.F) \xrightarrow{\tau} (\nu a)(E|F)$$

è il seguente

$$\frac{\overline{\frac{a.E \xrightarrow{a} E}{a.E + b.\mathbf{0} \xrightarrow{a} E} \quad \frac{\overline{\bar{a}.F \xrightarrow{\bar{a}} F}}{(a.E + b.\mathbf{0})|\bar{a}.F \xrightarrow{\tau} E|F}}}{(\nu a)((a.E + b.\mathbf{0})|\bar{a}.F) \xrightarrow{\tau} (\nu a)(E|F)}$$

Esempio

$$\begin{array}{c}
 \text{(Pref)} \frac{}{a.b.\mathbf{0} \xrightarrow{a} b.\mathbf{0}} \\
 (\text{Sum}_1) \frac{}{a.b.\mathbf{0} + b.a.\mathbf{0} \xrightarrow{a} b.\mathbf{0}}
 \end{array}
 \quad
 \begin{array}{c}
 \text{(Pref)} \frac{}{b.a.\mathbf{0} \xrightarrow{b} a.\mathbf{0}} \\
 (\text{Sum}_2) \frac{}{a.b.\mathbf{0} + b.a.\mathbf{0} \xrightarrow{b} a.\mathbf{0}}
 \end{array}$$

$$\begin{array}{c}
 \text{(Pref)} \frac{}{\bar{a}.\mathbf{0} \xrightarrow{\bar{a}} \mathbf{0}} \\
 (\text{Sum}_1) \frac{}{\bar{a}.\mathbf{0} + c.\mathbf{0} \xrightarrow{\bar{a}} \mathbf{0}}
 \end{array}
 \quad
 \begin{array}{c}
 \text{(Pref)} \frac{}{a.c.\mathbf{0} \xrightarrow{a} c.\mathbf{0}} \\
 (\text{Com}) \frac{}{a.c.\mathbf{0} | (\bar{a}.\mathbf{0} + c.\mathbf{0}) \xrightarrow{\tau} c.\mathbf{0} | \mathbf{0}}
 \end{array}$$

$$\begin{array}{c}
 \text{(Res)} \frac{}{(vc)(a.c.\mathbf{0} | (\bar{a}.\mathbf{0} + c.\mathbf{0})) \xrightarrow{\tau} (vc)(c.\mathbf{0} | \mathbf{0})}
 \end{array}$$

La risoluzione è creata a ritroso dal basso.

Teorema (SOS ben formata) Per un $p \in \mathcal{P}$ se $p \xrightarrow{\mu} p'$ allora $p' \in \mathcal{P}$

Proof Per induzione sulla prova $p \xrightarrow{\mu} p'$ con p un termine di processo CCS, allora anche p' è un termine di processo. Oppure, $\text{Const}(p') \subseteq \text{Const}(p)$ tale che se p soddisfa la condizione di guardabilità, allora regge la cosa anche per p' . ■

Proposizione Per ogni $p \in \mathcal{P}$, l'LTS $\mathcal{C}_p = (\mathcal{P}_p, \text{sort}(p), \rightarrow_p, p)$ raggiungibile da p è ridotto da un LTS rooted.

Problema dell'esplosione dello stato nello spazio Se vi sono porcessi $p_1|p_2|...|p_n$ dove per ogni p_i si genera un LTS con 10 stati, allora $p_1|p_2|...|p_n$ ha 10^n stati.

Proposizione Per ogni processo CCS q , l'nsieme delle transizioni in uscita da q è finito. Vi è un upper-bound $\gamma(q)$ sul numero di transizioni che lasciano uno stato/processo q .

$\gamma : \mathcal{P} \rightarrow \mathbb{N}$ t.c.

$$\gamma(\mathbf{0}) = \mathbf{0} \quad \gamma(\mu.p) = 1 \quad \gamma(p_1 + p_2) = \gamma(p_1) + \gamma(p_2)$$

$$\gamma(p_1|p_2) = \gamma(p_1) + \gamma(p_2) + \gamma(p_1) \times \gamma(p_2) \quad \gamma(A) = \gamma(p) \text{ se } A \stackrel{\text{def}}{=} p \quad \gamma((\nu a)p) = \gamma(p)$$

La guardabilità di $\gamma(A)$ garantisce il numero finito.

Corollario Un LTS raggiungibile da un processo CCS p è finitely branching (ogni stato raggiungibile ha un numero finito di transizioni in uscita).

Tutti i $p \sim E\{\frac{p}{X}\}$ sono soluzione di $X \sim E(X)$.

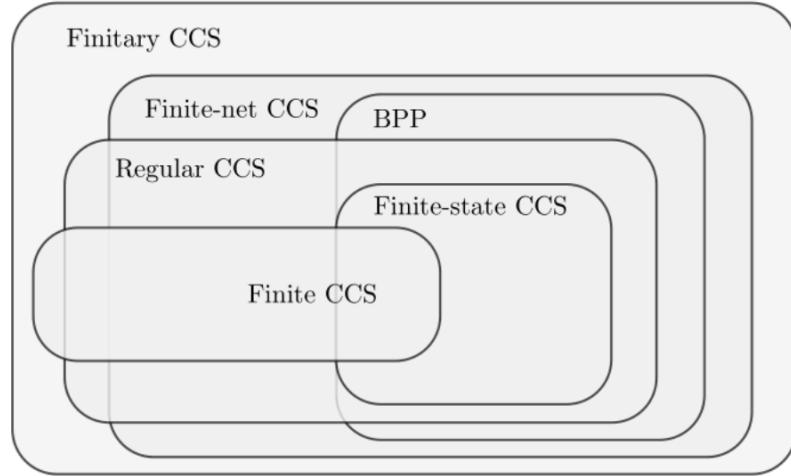
Proposizione Guardatezza \Rightarrow soluzione unica.

Con X processo di una costante guardata in E e $\text{var}(E) \subseteq \{X\}$. La soluzione è unica up to \sim quando $p \sim E\{\frac{p}{X}\}$ e $q \sim E\{\frac{q}{X}\} \Rightarrow p \sim q$.

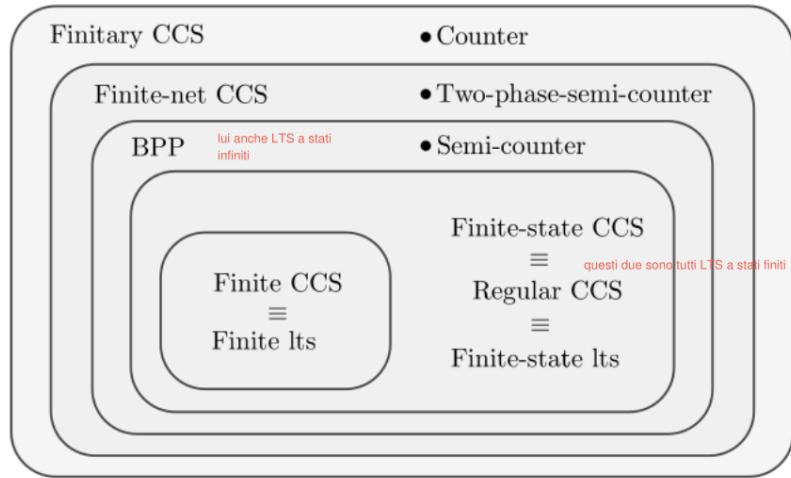
5.7. Gerarchie

Possiamo definire CCS all'interno di sottoclassi gerarchiche le quali vanno ad intersecarsi su alcuni costrutti sintattici e/o semanticci.

A livello di **sintassi** abbiamo che



mentre, a livello di **semantica**, abbiamo che



in cui, da come si capisce dallo screenshot, abbiamo una separazione tra gli LTS a stati finiti e infiniti.

5.7.1. CCS Finite

Non si usano costanti.

$$p ::= \mathbf{0} \mid \mu.q \mid p + p$$

$$q ::= p \mid q|q \mid (\nu a)q$$

oppure anche scritto come

$$p ::= \sum_{j \in J} \mu_j \cdot p_j \mid p|p \mid (\nu a)p$$

Non vi sono comportamenti ciclici o infiniti. Viene usato per descrivere esempi semplici.

Proposizione Finite CCS \Rightarrow Finite LTS

Dimostrazione Il linguaggio non offre nessun meccanismo di iterazione o ricorsione. ■

Si può definire una funzione $\text{size}(p)$ per un processo p la quale ritorna il numero di prefissi che occorrono in p .

$$\text{size}(\mathbf{0}) = 0$$

$$\text{size}(\mu.p) = 1 + \text{size}(p)$$

$$\text{size}(p_1 + p_2) = \text{size}(p_1) + \text{size}(p_2)$$

$$\text{size}(p_1 \mid p_2) = \text{size}(p_1) + \text{size}(p_2)$$

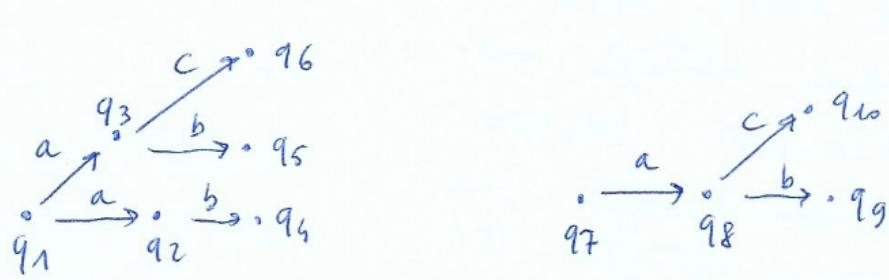
$$\text{size}((\nu a)p) = \text{size}(p)$$

Osservazione Per un $p \in \mathcal{P}_{\text{fin}}$ la funzione $\text{size}(p)$ ritorna un numero $n \in \mathbb{N}$ t.c. $\text{size}(p) = 0$ e dunque $p \not\rightarrow$.

Dimostrazione Per ogni $p \in \mathcal{P}_{\text{fin}}$ se $p \xrightarrow{\mu} p'$, allora $\text{size}(p') < \text{size}(p)$. La transizione deve essere aciclica perché $q_1 \xrightarrow{\mu_1} q_2 \xrightarrow{\mu_2} \dots q_n \xrightarrow{\mu_n} q_{n+1}$ determina una catena decrescente t.c. $\text{size}(q_1) > \text{size}(q_2) > \dots > \text{size}(q_{n+1})$ con 0 come lower bound. Quindi, se $\text{size}(q_1) = k$ allora il percorso non può essere più lungo di k .

Esempio

Dati i due LTS seguenti



si possono ricavare due processi p e q di Finite CCS con semantica isomorfa a questi due e poi si può dimostrare che non sono strong bisimili.

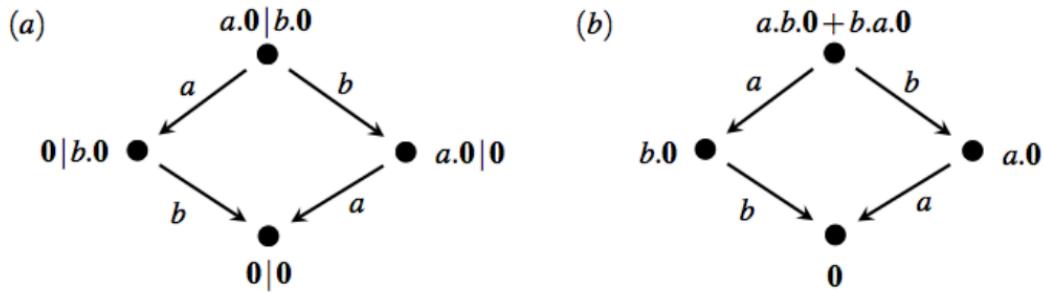
Ad ogni deadlock si rappresenta un ristretto, tipo per q_6 si ha $(\nu a)\mathbf{0}$.

$$p = a.(c.(\nu c)\mathbf{0} + b.(\nu b)\mathbf{0}) + a.b.(\nu a)\mathbf{0}$$

$$q = a.(c.(\nu d)\mathbf{0} + b.(\nu e)\mathbf{0})$$

Per ogni LTS finito TS ridotto si ha una rappresentazione in finite CCS di un processo p t.c. che l'LTS associato al processo p generato è isomorfo a TS .

Modello di interleaving Il problema della scelta del modello, causato dall'enorme astrazione, non riesce a distinguere la concorrenza dalla sequenzialità.



5.7.2. CCS finite-state

È il sottolinguaggio definito solo con nil, prefisso d'azione, operatore di scelta e numero finito di costanti. Non è molto espressivo perché non capace di modellare sottoprocessi che compongono un sistema complesso (alias modellazione compositiva). Mancano infatti gli operatori di restrizione e composizione parallela.

Con infinite costanti possono essere espressi dei semicontatori, come ad esempio

$$\begin{aligned} \text{SCount}_0 &\stackrel{\text{def}}{=} \text{inc.SCount}_1 \\ \text{SCount}_n &\stackrel{\text{def}}{=} \text{inc.SCount}_{n+1} + \text{dec.SCount}_{n+1} \quad n > 0 \end{aligned}$$

Questo perché ogni LTS finitely-branching con infiniti stati può essere rappresentato con infinite costanti (con somma e prefisso).

Formalmente la sintassi è definita come

$$p ::= \mathbf{0} \mid \mu.q \mid p + p$$

$$q ::= p \mid C$$

con costanti $C = q$ sempre definite e guardate, con q finito.

Una sintassi alternativa può essere espressa come

$$p ::= \sum_{j \in J} \mu_j \cdot p_j \mid C$$

Proposizione finite-state CCS \Rightarrow finite-state LTS

Un LTS finite-state ha numero di stati e numero di transizioni finite.

Teorema Per ogni processo p CCS finite-state, l'insieme \mathcal{P}_p degli stati raggiungibili, è finito.

Dimostrazione Definendo $\chi(p, \emptyset)$ come upper-bound del numero possibile di processi/stati raggiungibili da p si ha un insieme I di costanti già incontrate.

$$\chi(\mathbf{0}, I) = 1$$

$$\chi(p_1 + p_2, I) = \chi(p_1, I) + \chi(p_2, I) + 1$$

$$\chi(\mu.p, I) = 1 + \chi(p, I)$$

$$\chi(A, I) = \begin{cases} 0 & \text{if } A \in I \\ 1 + \chi(p, I \cup \{A\}) & \text{if } A \stackrel{\text{def}}{=} p \wedge A \notin I \end{cases}$$

L'insieme delle costanti $\text{Const}(p)$ è finito per il processo p , pertanto anche $\chi(p, \emptyset)$ ritorna un numero finito senza andare in loop.

Corollario (LTS finite-state) Per ogni processo p CCS finite-state, l'LTS raggiungibile da p , $\mathcal{C}_p = (\mathcal{P}_p, \text{sort}(p), \rightarrow_p, p)$ è finite-state.

Teorema (Rappresentabilità) Per ogni rooted LTS finite-state TS , esiste un processo p CCS finite-state, tale che l'LTS raggiungibile $\mathcal{C}_p = (\mathcal{P}_p, \text{sort}(p), \rightarrow_p, p)$ è isomorfo a TS .

Dimostrazione Preso $TS = (Q, A, \rightarrow_1, q_0)$ con $Q = \{q_0, q_1, \dots, q_n\}$, si definisce un processo costante C_i in corrispondenza dello stato q_i per $i = 0, 1, \dots, n$ definito come:

- Se q_i è deadlock, allora $C_i \stackrel{\text{def}}{=} 0$;
- Se $T(q_i) = \{(q_i, \mu, q_k) \mid \exists \mu \in A, \exists q_k \in Q. q_i \xrightarrow{\mu} q_k\}$, allora $C_i = \sum_{(q_i, \mu, q_k) \in T(q_i)} \mu.C_k$

Preso $\mathcal{C}_{C_0} = (\mathcal{P}_{C_0}, \text{sort}(C_0), \rightarrow_2, C_0)$, si vece che è $= \{C_0, C_1, \dots, C_n\}$ perché TS è ridotto.

Allora, la biiezione che cerchiamo è $f : Q \rightarrow \mathcal{P}_{C_0}$ definita come $f(q_i) = C_i$. Due condizioni di isomorfismo sono soddisfatte:

1. $C_0 = f(q_0)$
2. $q \xrightarrow{a} \Leftrightarrow f(q) \xrightarrow{a} f(q')$

quindi f è un LTS isomorfo. ■

Linguaggio CCS finite-state Un linguaggio $L \subseteq (\mathcal{L} \cup \overline{\mathcal{L}})^*$ è un linguaggio CCS finite-state se esiste un processo p CCS finite-state t.c. le tracce weak completed sono L , o meglio, $WCTr(p) = L$

I linguaggi CCS finite-state coincidono con i linugaggi regolari.

Esempio

Definire un processo finite-state per una macchina del caffè tale capace di accettare monete da 1€ o 2€, vendere espresso per 2€ e caffè americano per 1€ e capace di tenere credito fino a 4€. Più della somma massima, il processo deve rifiutare l'immissione di nuove monete.

$$0VM \doteq 1\text{€}.1VM + 2\text{€}.2VM$$

$$1VM \doteq \text{ask-am.am-coffee}.0VM + 1\text{€}.2VM + 2\text{€}.3VM$$

$$2VM \doteq \text{ask-am.am-coffee}.1VM + \text{ask-esp.esp-coffee}.0VM + 1\text{€}.3VM + 2\text{€}.4VM$$

$$3VM \doteq \text{ask-am.am-coffee}.2VM + \text{ask-esp.esp-coffee}.1VM + 1\text{€}.4VM$$

$$4VM \doteq \text{ask-am.am-coffee}.3VM + \text{ask-esp.esp-coffee}.2VM$$

Sintatticamente, non è contenuto in CCS regular. Ad esempio, $a.(a.\mathbf{0} \mid b.\mathbf{0})$

Sintatticamente, non è contenuto in CCS finite-net. Ad esempio, $a.((\nu a)a.\mathbf{0} \mid b.\mathbf{0})$

5.7.3. CCS Regular

Operatori statici non vengono usati e i suoi processi sono ancora a stati finiti. Con parallelo e restrizioni fuori dalle costanti avremmo CCS completo.

$$s ::= \mathbf{0} \mid \mu.t \mid s + s$$

$$t ::= s \mid C$$

$$p ::= t \mid (\nu a)p \mid p|p$$

oppure anche

$$t ::= \sum_{j \in J} \mu_j \cdot t_j \mid C$$

$$p ::= t \mid (\nu a)p \mid p|p$$

Le costanti sono definite e guardate. $\text{Const}(p)$ finito. C ha corpo in t , quindi restrizione e parallelo non occorrono nel corpo di una costante.

Proposizione Regular CCS \Rightarrow finite-state LTS

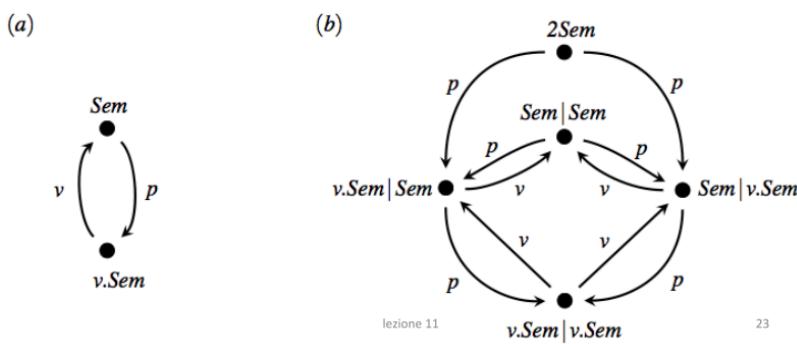
Un processo di regular CCS p è ottenuto come composizione parallela o restrizione di alcuni processi CCS finite-state p_1, \dots, p_n . Ogni p_i genera un LTS a stati finiti.

Se p_i ha k stati e p_j ha h stati, allora $p_i|p_j$ ha $k \cdot h$ stati; $(\nu a)p_i$ ha al massimo k stati.

Il processo regular CCS p deve avere dunque numero finito di stati ma, dato che l'LTS per p è finitely-branching (conseguenza della guardatezza delle costanti) $\Rightarrow p$ è finite-state.

Esempio

Presi un semaforo $Sem = p.v.Sem$ e $2Sem = Sem \mid Sem$, quello che avviene tra p e v è quello che fa l'altro processo: 1 solo processo può accedere alla sezione critica.

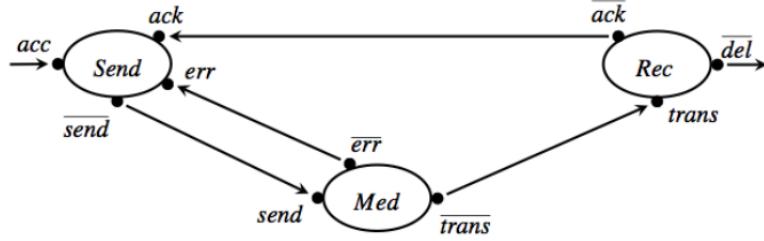


si può estendere come $nSem = \prod_{1 \leq n} Sem$ ed esso avrà $2^n + 1$ stati.

Visto che la proprietà di essere deadlock-free è composizione (basta che uno dei componenti sia deadlock-free), basta verificare che un singolo semaforo sia deadlock, ma Sem lo è (quello con 2 stati), quindi anche $nSem$ è deadlock-free.

Esempio

Preso un protocollo di rete definito dal seguente flow-graph



può essere definito mediante CCS come:

$$\text{Protocol} \stackrel{\text{def}}{=} (\nu \text{ send, error, trans, ack})((\text{Send}|\text{Med})|\text{Rec})$$

$$\text{Send} \stackrel{\text{def}}{=} \text{acc}. \text{Sending}$$

$$\text{Sending} \stackrel{\text{def}}{=} \overline{\text{send}}. \text{Wait}$$

$$\text{Wait} \stackrel{\text{def}}{=} \text{ack}. \text{Send} + \text{error}. \text{Sending}$$

$$\text{Med} \stackrel{\text{def}}{=} \text{send}. \text{Med}'$$

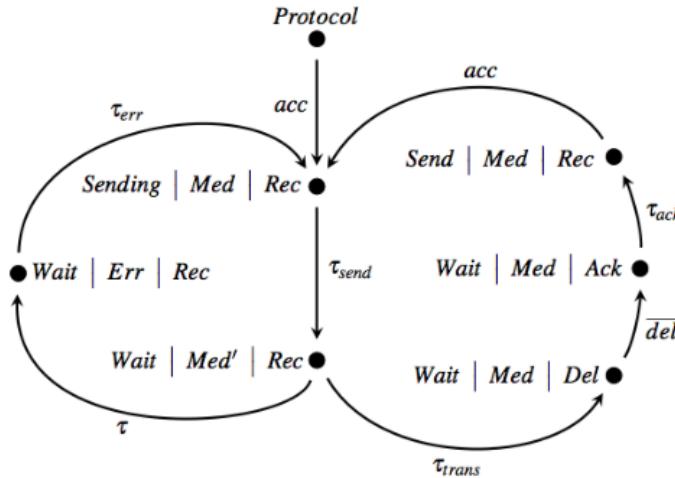
$$\text{Med}' \stackrel{\text{def}}{=} \tau. \text{Err} + \overline{\text{trans}}. \text{Med}$$

$$\text{Err} \stackrel{\text{def}}{=} \overline{\text{error}}. \text{Med}$$

$$\text{Rec} \stackrel{\text{def}}{=} \text{trans}. \text{Del}$$

$$\text{Del} \stackrel{\text{def}}{=} \overline{\text{del}}. \text{Ack}$$

$$\text{Ack} \stackrel{\text{def}}{=} \overline{\text{ack}}. \text{Rec}$$



Con la specifica sequenziale

$$\text{ProtocolSpec} = \text{acc}. \overline{\text{del}}. \text{ProtocolSpec}$$

si può avere un check di equivalenza perché, definita la bisimulazione weak R come

$$\begin{aligned}
R = \{ & (\text{Protocol}, \text{ProtocolSpec}), \\
& (\text{Sending}|\text{Med}|\text{Rec}, \overline{\text{del}}.\text{ProtocolSpec}), \\
& (\text{Wait}|\text{Med}'|\text{Rec}, \overline{\text{del}}.\text{ProtocolSpec}), \\
& (\text{Wait}|\text{Err}|\text{Rec}, \overline{\text{del}}.\text{ProtocolSpec}), \\
& (\text{Wait}|\text{Med}|\text{Del}, \overline{\text{del}}.\text{ProtocolSpec}), \\
& (\text{Wait}|\text{Med}|\text{Ack}, \text{ProtocolSpec}), \\
& (\text{Send}|\text{Med}|\text{Rec}, \text{ProtocolSpec}) \}
\end{aligned}$$

si ha

$$\text{ProtocolSpec} \approx \text{Protocol}$$

entrambi possono fare solo acc all'inizio.

Esempio

Si definisce un buffer a due posizioni: una semplice struttura che ha due input in due buffer separati.

La specifica sequenziale può essere

$$B_0 = \text{in}.B_1$$

$$B_1 = \text{in}.B_2 + \overline{\text{out}}.B_0$$

$$B_2 = \overline{\text{out}}.B_1$$

Qui però non si capisce quale output stia facendo e per risolverlo serve la versione CCS value passing.

Mentre quella parallela in cui non si rispetta l'ordine è

$$B \mid B$$

con

$$B = \text{in}.B'$$

$$B' = \overline{\text{out}}.B$$

Si ha l'equivalenza $B_0 \sim B|B$.

Un'implementazione della specifica sequenziale ma che rispetta l'ordine è fatta con pipeline come

$$\text{Buff} = (\nu d)(\text{Buf}_1 \mid \text{Buf}_2)$$

$$\text{Buf}_1 = \text{in}.\overline{d}.\text{Buf}_1$$

$$\text{Buf}_2 = d.\overline{\text{out}}.\text{Buf}_2$$

Si ha l'equivalenza $B_0 \approx \text{Buff}$.

Dato che non vengono trasmessi valori non si vede una vera differenza tra le due implementazioni.

Esempio

2 produttori - 1 consumatore

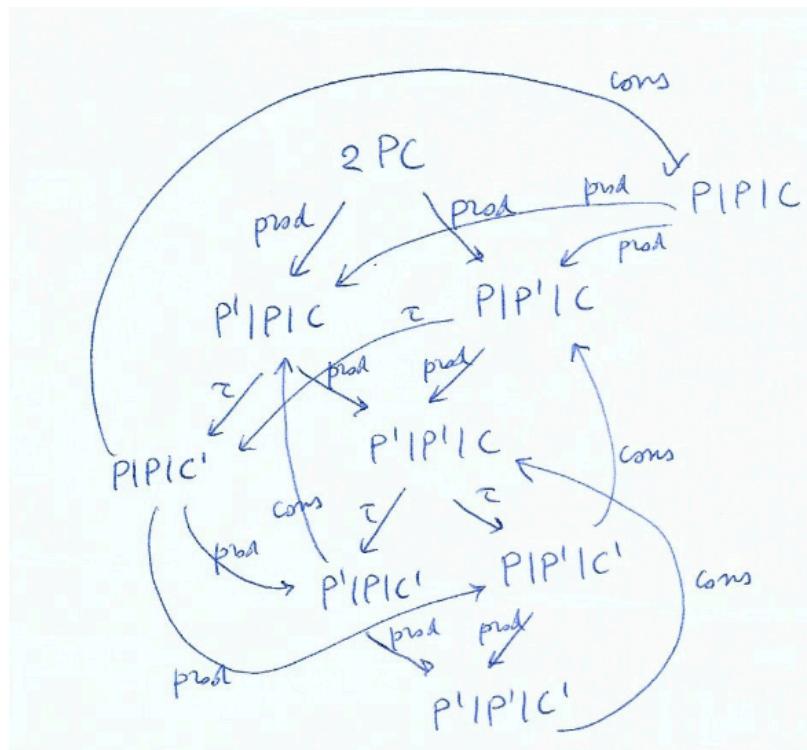
$$2PC = (\nu \text{ send})((P|P) \mid C)$$

$$P = \text{produce}.\overline{\text{send}}.P$$

$$C = \text{send}.consume.C$$

è la restrizione su send a garantire che solo il consumer è capace di consumare dati dai due produttori.

L'LTS associato ad esso è



con 9 stati e 18 transizioni. 3 produzioni possono essere fatte prima che il consumo diventi obbligatorio.

Un LTS con 3 produttori e 2 consumatori ha circa 30 stati e 70 transizioni.

Estendendo questo esempio con un **buffer ad una posizione** si ha

$$PBC = (\nu \text{ in.out})((P_1|B) \mid C_1)$$

$$P_1 = \text{produce.in.} \overline{P}_1$$

$$C_1 = \text{out.consume.} C_1$$

$$B = \text{in.} \overline{\text{out.}} B$$

- $PBC \approx 2PC$ perché la traccia strong “produce produce” c’è solo per il secondo.
- $PBC \approx 2PC$, anche se è complesso da fare a mano.

Estendendo questo esempio con un **buffer a due posizioni** si ha

$$P2BC = (\nu \text{ in.out})((P_1|B_0) \mid C_1)$$

$$P_1 = \text{produce.in.} \overline{P}_1$$

$$C_1 = \text{out.consume.} C_1$$

$$B_0 = \text{in.} B_1$$

$$B_1 = \text{in.} B_2 + \overline{\text{out.}} B_0$$

$$B_2 = \overline{\text{out.}} B_1$$

- $PBC \not\approx P2BC$ perché avere buffer a 1 o 2 posizioni crea flowgraph diversi che poi non sono visibili.

$$P2BC' = (\nu \text{ in.out})((P_1 \mid (B|B)) \mid C_1)$$

- $PBC \sim P2BC'$

$$P2BC'' = (\nu \text{ in.out})((P_1 \mid \text{Buff}) \mid C_1)$$

$$\text{Buff} = (\nu d)(\text{Buf}_1 \mid \text{Buf}_2)$$

$$\text{Buf}_1 = \text{in.} \overline{d}. \text{Buf}_1$$

$$\text{Buf}_2 = d. \overline{\text{out.}} \text{Buf}_2$$

- $P2BC \approx P2BC''$

5.7.3.1. Buffer a n posizioni

Presi gli esempi sopra definiti possiamo avere

- Specifica sequenziale

$$B_0 \stackrel{\text{def}}{=} \text{in.} B_1$$

$$B_i \stackrel{\text{def}}{=} \text{in.} B_{i+1} + \overline{\text{out.}} B_{i-1} \quad 0 < i < n$$

$$B_n \stackrel{\text{def}}{=} \overline{\text{out.}} B_{n-1}$$

- Implementazione parallela

$$B^n = \prod_{1 \leq i \leq n} B$$

$$B = \text{in}.\overline{\text{out}}.B$$

Sfruttando l'associatività e commutatività del parallelo si può definire la bisimulazione strong up to \sim come

$$R = \left\{ \left(B_k, \left(\prod_{i=1}^k \overline{\text{out}}.B \right) \mid B^{n-k} \right) \mid 0 \leq k \leq n \right\}$$

B^n sono n B in parallelo, con B_0 versione iniziale.

per $k = 0$ si ha che la coppia in R è $(B_0, \mathbf{0} \mid B^n)$.

Se R è bisimulazione up to \sim , allora si ha $B_0 \sim \mathbf{0} \mid B^n$.

Con $\mathbf{0} \mid B^n \sim B^n$, per transitività, si ha $B_0 \sim B^n$.

Ora si prova che R è bisimulazione strong up to \sim provando

1. Se $B_k \xrightarrow{\alpha} p$ per qualche azione α e un processo p , allora ci sono q e q' tale che $(\prod_{i=1}^k \overline{\text{out}}.B) \mid B^{n-k} \xrightarrow{\alpha} q$ con $q' \sim q$ e $(p, q') \in R$ (e questo va bene visto che per applicazioni up-to, si ha \sim riflessivo, quindi $p \sim p$).
2. Simmetricamente, se $(\prod_{i=1}^k \overline{\text{out}}.B) \mid B^{n-k} \xrightarrow{\alpha} q$, allora si ha che anche i processi q' e p tali che $q \sim q'$, $B_k \xrightarrow{\alpha} p \wedge (p, q') \in R$.

Sfruttando proprietà associativa e commutativa del parallelo si ha

- (Caso 1) Se $k < n$, il processo B_k può fare $B_k \xrightarrow{\text{in}} B_{k+1}$ e $(\prod_{i=1}^k \overline{\text{out}}.B) \mid B^{n-k}$ può rispondere al in raggiungendo $(\prod_{i=1}^k \overline{\text{out}}.B) \mid \overline{\text{out}}.B \mid B^{-(k+1)}$ che è strong bisimili a $(\prod_{i=1}^{k+1} \overline{\text{out}}.B) \mid B^{n-(k+1)}$ e la coppia $(B_{k+1}, (\prod_{i=1}^{k+1} \overline{\text{out}}.B) \mid B^{n-(k+1)})$ è in R .

Se $k > 0$, B_k può anche fare $B_k \xrightarrow{\overline{\text{out}}} B_{k-1}$ e $(\prod_{i=1}^k \overline{\text{out}}.B) \mid B^{n-k}$ può rispondere raggiungendo $(\prod_{i=1}^{k-1} \overline{\text{out}}.B) \mid B^{n-(k-1)}$ e la coppia $(B_{k-1}, (\prod_{i=1}^{k-1} \overline{\text{out}}.B) \mid B^{n-(k-1)})$ è in R .

- (Caso 2) $(\prod_{i=1}^k \overline{\text{out}}.B) \mid B^{n-k}$ ha altre $n - (k - 1)$ transazioni in con stati strong bisimili a $(\prod_{i=1}^{k+1} \overline{\text{out}}.B) \mid B^{n-(k+1)}$ per associatività e commutatività della composizione parallela. Per esempio, uno degli stati stati $\prod_{i=1}^k \overline{\text{out}}.B \mid B^j \mid \overline{\text{out}}.B \mid B^{n-(k+j+1)}$ per $1 \leq j, k \leq n, j + k < n$. Ad ognuna di queste transizioni, B_k può rispondere con $B_k \xrightarrow{\text{in}} B_{k+1}$ e la coppia $(B_{k+1}, (\prod_{i=1}^{k+1} \overline{\text{out}}.B) \mid B^{n-(k+1)})$ è in R .

Se $k > 0$, $(\prod_{i=1}^k \overline{\text{out}}.B) \mid B^{n-k}$ ha altre $k - 1$ transazioni $\overline{\text{out}}$ con stati strong bisimili a $(\prod_{i=1}^{k-1} \overline{\text{out}}.B) \mid B^{n-(k-1)}$. Ad ognuna di queste transizioni B_k può rispondere con $B_k \xrightarrow{\overline{\text{out}}} B_{k-1}$ e la coppia $(B_{k-1}, (\prod_{i=1}^{k-1} \overline{\text{out}}.B) \mid B^{n-(k-1)})$ è in R . ■

Una implementazione, invece, usando l'operatore di pipeline può essere descritta con PB in modo da poter essere B oppure $\overline{\text{out}}.B$. $PB_\pi^{(n,k)}$ è usato per denotare $PB_1 \cap PB_2 \cap \dots \cap PB_n$.

Qui, per i due termini PB_i e PB_{i+1} per $i = 1, \dots, n - 1$, l' out del primo è collegato all' in del secondo. π è un vettore binario di lunghezza n con solo k elementi impostati a 1.

$$\pi(i) = \begin{cases} 1 & \text{if } PB_i = \overline{\text{out}}.B \\ 0 & \text{if } PB_i = B \end{cases}$$

Se $\pi(i) = 1$ e $\pi(i+1) = 0$ allora con $\pi[i+1/i]$ è il vettore $(\pi(1), \dots, \pi(i-1), 0, 1, \pi(i+2), \dots, \pi(n))$.

Da notare però che $PB_0^{(n,0)}$ e $PB_1^{(n,n)}$ sono composti da soli componenti B .

Vi è una bisimulazione weak grazie al linking associativo.

$$R = \left\{ \left(B_k, PB_{\pi}^{(n,k)} \right) \mid 0 \leq k \leq n \wedge |\pi| = n \text{ con } k \text{ elementi} = 1 \right\}$$

$$\overline{\text{out}}.B \cap B \xrightarrow{\tau} B \cap \overline{\text{out}}.B$$

$$PB_{\pi}^{(n,k)} \xrightarrow{\tau} PB_{\pi[i+1/i]}^{(n,k)}$$

Ho due relzioni possibili:

- R_1 se $n \geq 1$, $PB_1^{(n,n)} \cap B \xrightarrow{\tau} B \cap PB_1^{(n,n)}$
- R_2 se $n \geq 1$, $\overline{\text{out}}.B \cap PB_0^{(n,0)} \xrightarrow{\tau} PB_0^{(n,0)} \cap \overline{\text{out}}.B$

Sintatticmaente, non è contenuto in CCS Finite-net. Ad esempio, $a.\mathbf{0} \mid (\nu a)b.\mathbf{0}$

5.7.4. BPP: Basic Parallel Processes

Estensione di CCS finite-state che permette l'uso del parallelo.

$$s ::= \mathbf{0} \mid \mu.p \mid s + s$$

$$p ::= s \mid p|p \mid C$$

Oppure anche

$$p ::= \sum_{j \in J} \mu_j \cdot p_j \mid C \mid p|p$$

Il parallelo non consente la sincronizzazione, è puramente asincrono. Le costanti sono sempre definite e guardate per ogni processo p , e $\text{Const}(p)$ è finito.

Sintatticamente, è una superclasse di CCS finite-state ma non contiene CCS regular perché

1. la restrizione può essere usata da CCS regular;
2. il parallelo può essere usato in definizioni di costanti per BPP (eg. $C = a.\mathbf{0} \mid b.C$).

Semanticamente è una superclasse di CCS finite-state e regular perché esprime tutti gli LTS finiti + molti infiniti.

Esempio

In CCS si ha un semi-counter definito come

$$\text{SCount}_0 \stackrel{\text{def}}{=} \text{inc}.\text{SCount}_1$$

$$\text{SCount}_n \stackrel{\text{def}}{=} \text{inc}.\text{SCount}_{n+1} + \text{dec}.\text{SCount}_{n-1} \quad n > 0$$

ma qui SCount_i non può essere uguale a SCount_{i+1} perché il primo non può eseguire dec .

In CCS BPP possiamo esprimere come

$$\text{SC} \stackrel{\text{def}}{=} \text{inc}.(\text{SC}|\text{dec}.\mathbf{0})$$

Il quale è bisimile strong a SCount_0 .

Dimostrazione Presa la relazione $R = \{((\text{SCount}_n, \text{SC} \mid \prod_{i=1}^n \text{dec}.\mathbf{0}) \mid n \geq 0 \}$, si vede come sia strong bisimile up to \sim e lo si fa grazie al fatto che la composizione parallela è associativa, commutativa, neutrale con $\mathbf{0}$. Inoltre \sim è una congruenza per la composizione parallela (eg. se $p \sim q$, allora $p|r \sim q|r$ per tutte le r). Se $n = 0$ la coppia $(\text{SCount}_n, \text{SC}|\mathbf{0})$ è in R . Se R è una bisimulazione up to \sim , allora $\text{SCount}_0 \sim \text{SC}|\mathbf{0}$. Per $\text{SC}|\mathbf{0} \sim \text{SC}$ abbiamo, per transitività, $\text{SCount}_0 \sim \text{SC}$. Bisogna dimostrare che R è una bisimulazione strong up to \sim . Leggere il resto dal libro. ■

L è un linguaggio BPP se esiste un processo BPP p t.c. $WCTr(p) = L$. Visto che include CCS finite-state, allora tutti i linguaggi regolari sono esprimibili in BPP, ma può anche generare alcuni linguaggi context-free.

BPP può generare linguaggi non context-free.

Esempio

Le tracce complete di $A \stackrel{\text{def}}{=} a.(A \mid b.\mathbf{0}) + c.\mathbf{0}$ non costituiscono un linguaggio regolare perché se interseco il linguaggio a^*cb^* (che è regolare) ottengo un linguaggio context dependent.

$$CTr(A) \cap a^*bc^* = \{a^kcb^k \mid k \geq 0\}$$

$B \stackrel{\text{def}}{=} a.(B \mid b.\mathbf{0}) + c.(B \mid d.\mathbf{0}) + e.\mathbf{0}$ è tale che $CTr(B)$ non è context-free perché l'intersezione col linguaggio regolare $a^*c^*b^*d^*e$ è

$$CTr(B) \cap a^*c^*b^*d^*e = \{a^kc^n b^k d^n e \mid k, n \geq 0\}$$

ovvero un linguaggio context-dependent.

L'equivalenza per bisimulazione forte su processi BPP è decidibile (PSPACE completo). Essa è l'unica equivalenza decidibile su classe di sistemi a stati infiniti.

Per bisimilarità weak ci sono solo risultati parziali.

Per processi BPP che possono sempre terminare vale la decidibilità di branching bisimilarity.

5.7.5. CCS Finite-net

Estensione di BPP che permette di usare anche restrizione (non nel corpo della costante) e sincronizzazione. Non estende CCS regular (permette di usare parallelo nel corpo di una costante, ma CCS regular permette di fare mix di parallelo e restrizione).

$$s ::= \mathbf{0} \mid \mu.t \mid s + s$$

$$t ::= s \mid t|t \mid C$$

$$p ::= t \mid (\nu a)p$$

oppure anche

$$t ::= \sum_{j \in J} \mu_j.t_j \mid t|t \mid C$$

$$p ::= t \mid (\nu a)p$$

Semanticamente, ci sono processi che non possono essere rappresentati in BPP. Ogni suo processo può essere associato ad una rete di Petri finita e viceversa. Grazie alle reti di Petri vi si ha: raggiungibilità (presi i processi CCS finite-net p e q si può verificare che q è raggiungibile da p), bisimilarità strong tra processo CCS finite-net q e processo CSS finite-state p , regolarità strong (dato processo CCS finite-net q si può verificare se esiste processo p CCS finite-state ad esso bisimile strong). La bisimulazione equivalence è indecidibile fra due processi finite-net.

Un esempio di linguaggio è quello context-dependent $L = \{a^n b^m c^m \mid 0 \leq m \leq n\}$.

5.7.6. CCS Finitary

Sintatticamente, è il sotto-linguaggio più grande che include tutti gli altri. L'unico vincolo è che il processo p deve avere un insieme $\text{Const}(p)$ finito.

Semanticamente, è strettamente più espressivo di CCS finite-net. È Turing-completo. Esistono processi CCS full che non hanno equivalenza con un processo CCS finitary.

Esempio

Esempio di contatore con CCS full che può fare test su $\mathbf{0}$ è

$$\text{Counter}_0 \stackrel{\text{def}}{=} \text{zero}.\text{Counter}_0 + \text{inc}.\text{Counter}_1$$

$$\text{Counter}_n \stackrel{\text{def}}{=} \text{inc}.\text{Counter}_{n+1} + \text{dec}.\text{Counter}_{n-1} \quad n > 0$$

Nessun processo CCS finite-net può essere equivalente perché un processo CCS finite-net genera sempre una rete Petri finita (dimostrato che una rete di Petri finita non può rappresentare contatore con test su zero).

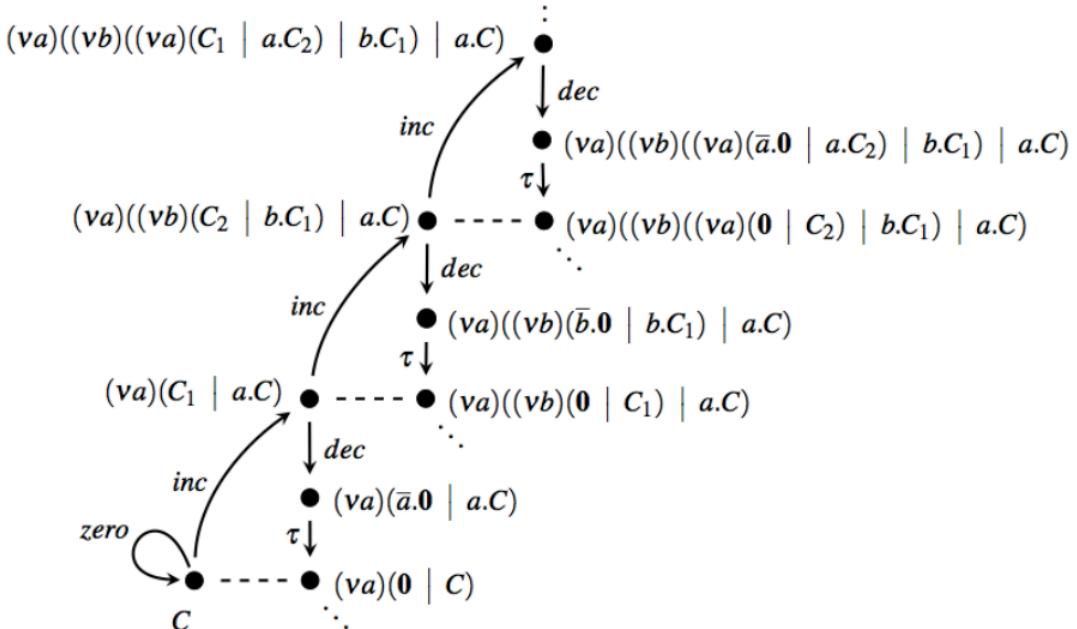
Questo contatore può essere scritto in CCS finitary usando 3 costanti

$$C \stackrel{\text{def}}{=} \text{zero}.C + \text{inc}.((\nu a)(C_1 \mid a.C))$$

$$C_1 \stackrel{\text{def}}{=} \text{dec}.\bar{a}.\mathbf{0} + \text{inc}.((\nu b)(C_2 \mid b.C_1))$$

$$C_2 \stackrel{\text{def}}{=} \text{dec}.\bar{b}.\mathbf{0} + \text{inc}.((\nu a)(C_1 \mid a.C_2))$$

in cui si alternano le attivazioni di istanze di C_1 (dispari) e C_2 (pari) dentro nuove restrizioni create ogni volta che si esegue inc. Il numero rappresentato da un termine è dato dal numero di restrizioni attive presenti.



$\text{Counter}_0 \approx C$ perché

$$p_0 = C$$

$$p_1 = (\nu a)(x \mid a.C)$$

$$p_{2n} = p_{2n-1}[(\nu b)(x \mid b.C_1)/x] \text{ per } n > 0$$

$$p_{2n+1} = p_{2n}[(\nu b)(x \mid b.C_1)/x] \text{ per } n > 0$$

e la relazione

$$\begin{aligned} R = & \{(C, \text{Counter}_0)\} \\ & \cup \{(p_{2n}[C_2/x], \text{Counter}_{2n}) \mid n > 0\} \\ & \cup \{(p_{2n+1}[C_1/x], \text{Counter}_{2n+1}) \mid n \geq 0\} \\ & \cup \{(p_{2n+1}[\bar{a}.\mathbf{0}/x], \text{Counter}_{2n}) \mid n \geq 0\} \\ & \cup \{(p_{2n}[\bar{b}.\mathbf{0}/x], \text{Counter}_{2n-1}) \mid n > 0\} \end{aligned}$$

è bisimulazione weak up to \approx .

Se $C \xrightarrow{0} C$ allora $\text{Counter}_0 \xrightarrow{0} \text{Counter}_0$ e $(C, \text{Counter}_0) \in R$

Se $C \xrightarrow{\text{inc}} (\nu a)(C_1 \mid a.C)$ allora $\text{Counter}_0 \xrightarrow{\text{inc}} \text{Counter}_1$ e, quando $n = 0$, $(p_1[C_1/x], \text{Counter}_1)$ appartiene al terzo gruppo di R .

Se $p_{2n+1}[C_1/x] \xrightarrow{\text{inc}} p_{2n+1}[(\nu b)(C_2 \mid b.C_1)/x]$ allora $\text{Counter}_{2n+1} \xrightarrow{\text{inc}} \text{Counter}_{2n+2}$.

Se $p_{2n+1}[C_1/x] \xrightarrow{\text{dec}} p_{2n+1}[\bar{a}.\mathbf{0}/x]$ allora $\text{Counter}_{2n+1} \xrightarrow{\text{dec}} \text{Counter}_{2n}$.

Con $p_1[\bar{a}.\mathbf{0}/x] = (\nu a)(\bar{a}.\mathbf{0} \mid a.C)$ si ha una sola transizione $p_1[\bar{a}.\mathbf{0}/x] \xrightarrow{\tau} (\nu a)(\mathbf{0} \mid C)$. Inoltre $(\nu a)(\mathbf{0} \mid C) \sim C$ e $\text{Counter}_0 \xrightarrow{\varepsilon} \text{Counter}_0 \approx \text{Counter}_0$ e $(C, \text{Counter}_0) \in R$.

Da Counter_0 si hanno due transizioni:

1. $\text{Counter}_0 \xrightarrow{0} \text{Counter}_0$ allora $p_1[\bar{a}.\mathbf{0}/x] \xrightarrow{\tau} (\nu a)(\mathbf{0} \mid C) \xrightarrow{0} (\nu a)(\mathbf{0} \mid C) \sim C$ e $(C, \text{Counter}_0) \in R$
2. $\text{Counter}_0 \xrightarrow{\text{inc}} \text{Counter}_1$ allora $p_1[\bar{a}.\mathbf{0}/x] \xrightarrow{\tau} (\nu a)(\mathbf{0} \mid C) \xrightarrow{\text{inc}} (\nu a)(\mathbf{0} \mid (\nu a)(C_1 \mid a.C)) \sim (\nu a)(C_1 \mid a.C) = p_1[C_1/x]$ e $(p_1[C_1/x], \text{Counter}_0) \in R$

[...] e così via per gli altri.

5.7.6.1. Turing completezza

Un formalismo è Turing-completo quando i programmi in quel formalismo possono calcolare tutte le funzioni calcolabili con macchine di Turing. Una conseguenza di questo è che le equivalenze comportamentali sono indecidibili per CCS finitary; se così non fosse, allora potremmo risolvere il problema della fermata (Halting problem).

Un esempio di formalismo Turing-completo sono le **Counter Machine**. Usano registri/contatori dove memorizzare valori $\mathbb{N} : r_1, \dots, r_n$. Inizialmente hanno i valori v_1, \dots, v_n . Il programma è un insieme indicizzato di istruzioni $\{(1, I_1), \dots, (m, I_m)\}$. Si inizia dall'istruzione di indice 1 e si va avanti a meno di salti; in genere si termina quando l'indice non è presente e/o $> m$. Quando il programma termine, il risultato si trova nei registri specificati come output.

Una classe speciale di CM ha 2 tipi di operazioni $1 \leq i \leq m, 1 \leq j \leq n$:

- $(i, \text{Inc}(r_j))$ incrementa il registro r_j e poi va all'istruzione $i + 1$
- $(i, \text{DecJump}(r_j, s)) = \begin{cases} \text{decrementa } r_j \text{ e va all'istruzione } i+1 \text{ se } r_j \neq 0 \\ \text{salta all'istruzione di indice } s \quad \text{se } r_j = 0 \end{cases}$

Lo stato iniziale ha $i = 1$ e i vari v_i sono quelli forniti in input.

La transizione $(i, v_1, \dots, v_n) \rightarrow_m (i', v'_1, \dots, v'_n)$ avviene se

- $I_i = \text{Inc}(r_j) \wedge i' = i + 1, v'_j = v_j + 1, v'_p = v_p$ per un qualcuno $p \neq j$;
- $I_i = \text{DecJump}(R_j, s), v_j > 0 \wedge i' = i + 1, v'_j = v_j - 1, v'_p = v_p$ per un qualcuno $p \neq j$;
- $I_i = \text{DecJump}(R_j, s), v_j = 0 \wedge i' = s, v'_p = v_p$ per un qualcuno $p = 1, \dots, n$.

Dati gli input v_1, \dots, v_n , per la CM $M = (I, n)$, lo stato interno (i, v'_1, \dots, v'_n) è detto terminale se $(1, v_1, \dots, v_n) \rightarrow_M^* (i, v'_1, \dots, v'_n)$ con $i > m$. I valori v'_1, \dots, v'_n sono detti gli output.

Se da $(1, v_1, \dots, v_n)$ non si raggiunge nessuna configurazione terminale, allora diciamo che la macchina, per quegli input, diverge.

Se la computazione termina in (i, v'_1, \dots, v'_n) con $i > m$, la funzione calcolata da M è

$$f_M(v_1, \dots, v_n) = (v'_1, \dots, v'_n)$$

sennò si dice che è indefinita.

Esempio

Calcolare la somma dei valori contenuti nei registri r_1 e r_2 , output va in r_1 .

Si usano 3 registri, in cui r_3 è a 0.

$$\{(1, \text{DecJump}(r_2, 4)), (2, \text{Inc}(r_1)), (3, \text{DecJump}(r_3, 1))\}$$

Useremo le **3CM**, ovvero counter machine con 3 contatori (classe universale Turing-completa) come formalismo universale. Non usiamo quella 2 contatori perché dovremmo godelizzare gli argomenti in I/O.

Data una CM $M = (I, 3)$ con input v_1, v_2, v_3 si definisce un processo CCS $\text{CM}_{M(v_1, v_2, v_3)}$ che modella il comportamento della CM M .

$$\text{CM}_{M(v_1, v_2, v_3)} \stackrel{\text{def}}{=} (\nu L)(P_1 \mid \dots \mid P_m \mid R_1 \mid R_2 \mid R_3 \mid B_{(v_1, v_2, v_3)})$$

- P_i sono le costanti che definiscono le istruzioni I_i .
- R_j sono le costanti che definiscono i registri r_j .
- B è il programmino di bootstrapping che inizializza registri e attiva prima istruzione.
- L è insieme delle azioni $\{\text{inc}_j, \text{zero}_j, \text{dec}_j \mid 1 \leq j \leq 3\}$ e $\{p_i \mid 1 \leq i \leq m + 1\}$.

L'istruzione $(i, \text{Inc}(r_j))$ è definita come

$$P_i \stackrel{\text{def}}{=} p_i.P'_i \quad P'_i \stackrel{\text{def}}{=} \overline{\text{inc}}_j \cdot \overline{p}_{i+1} \cdot P_i$$

dove p_i è l'azione che accetta l'abilitazione dell'istruzione; inc_j corrisponde all'incremento operato sul registro r_j ; \overline{p}_{i+1} abilita l'istruzione di indice $i + 1$.

L'istruzione $(i, \text{DecJump}(r_j, s))$ è definita come

$$P_i \stackrel{\text{def}}{=} p_i.P'_i \quad P'_i \stackrel{\text{def}}{=} \overline{\text{zero}}_j \cdot \overline{p}_s \cdot P_i + \overline{\text{dec}}_j \cdot \overline{p}_{i+1} \cdot P_i$$

dove la scelta tra zero_j e dec_j è guidata dallo stato del registro. Se $r_j = 0$ allora la sync su zero_j è possibile (e si attiva l'istruzione s), altrimenti, la sync su dec_j è possibile (si decrementa r_j e si attiva l'istruzione $i + 1$).

R_j , che modella il registro r_j , vengono definite come

$$R_j \stackrel{\text{def}}{=} \text{zero}_j \cdot R_j + \text{inc}_j \cdot ((\nu a)(R_{j_1} \mid a \cdot R_j))$$

$$R_{j_1} \stackrel{\text{def}}{=} \text{dec}_j \cdot \bar{a} \cdot \mathbf{0} + \text{inc}_j \cdot ((\nu b)(R_{j_2} \mid b \cdot R_{j_1}))$$

$$R_{j_2} \stackrel{\text{def}}{=} \text{dec}_j \cdot \bar{b} \cdot \mathbf{0} + \text{inc}_j \cdot ((\nu a)(R_{j_1} \mid a \cdot R_{j_2}))$$

Infine, per gli input v_1, \dots, v_n si ha

$$B_{(v_1, \dots, v_n)} \stackrel{\text{def}}{=} \underbrace{\overline{\text{inc}}_1 \cdots \overline{\text{inc}}_1}_{v_1 \text{ volte}} \cdots \underbrace{\overline{\text{inc}}_n \cdots \overline{\text{inc}}_n}_{v_n \text{ volte}} \cdot \bar{p}_1 \cdot 0$$

Per $i = 1, \dots, m$ si ha $\langle \text{CM}_{(i, v_1, v_2, v_3)} \rangle$ come l'insieme di tutti i termini della forma

$$(\nu L)(P_1 \mid \dots \mid P_{i-1} \mid P'_i \mid P_{i+1} \mid \dots \mid P_m \mid R'_1 \mid R'_2 \mid R'_3 \mid 0)$$

in cui, per $j = 1, 2, 3$ si ha $R'_j \approx \text{Counter}_{v_j}$ e R'_j non può fare τ inizialmente.

Ogni cambio di stato nella CM M determina una sequenza di sync che portano da un processo $\langle \text{CM}_{(i, v_1, v_2, v_3)} \rangle$ ad una $\langle \text{CM}_{(i', v'_1, v'_2, v'_3)} \rangle$.

Data una CM M con gli input v_1, v_2, v_3 si ha il processo CCS

$\text{CM}_{M(v_1, v_2, v_3)} \xrightarrow{*} Q \in \langle \text{CM}_{(1, v_1, v_2, v_3)} \rangle$ alcune cose che valgono sono

- $(1, v_1, v_2, v_3) \rightsquigarrow_M^* (i, v'_1, v'_2, v'_3) \iff \forall Q \in \langle \text{CM}_{(1, v_1, v_2, v_3)} \rangle, \exists Q' \in \langle \text{CM}_{(i', v'_1, v'_2, v'_3)} \rangle$ t.c. $Q \xrightarrow{*} Q'$
- Se $Q \in \langle \text{CM}_{(i, v'_1, v'_2, v'_3)} \rangle \wedge Q \xrightarrow{*} Q' \Rightarrow \exists Q'' \in \langle \text{CM}_{(i', v''_1, v''_2, v''_3)} \rangle$ t.c. $Q' \xrightarrow{*} Q''$
- $(1, v_1, v_2, v_3) \uparrow \iff \text{CM}_{M(v_1, v_2, v_3)} \uparrow$

Per rendere la terminazione osservabile possiamo definire una nuova istruzione

$$P_{m+1} \stackrel{\text{def}}{=} p_{m+1} \cdot \checkmark \cdot \mathbf{0}$$

in cui \checkmark è un'azione osservabile. Questo lo si fa perché potrebbero essere azioni infinite τ le quali renderebbero il processo weak bisimile ad uno che effettivamente termina.

$$\text{TCM}_{M(v_1, v_2, v_3)} \stackrel{\text{def}}{=} (\nu L') \left(P_1 \mid \dots \mid P_m \mid P_{m+1} \mid R_1 \mid R_2 \mid R_3 \mid B_{(v_1, v_2, v_3)} \right)$$

Viene modificata l'istruzione $(i, \text{DecJump}(r_j, s))$ come

$$p_i \stackrel{\text{def}}{=} p_i \cdot P'_i \quad P'_i \stackrel{\text{def}}{=} \begin{cases} \overline{\text{zero}}_j \cdot \bar{p}_s \cdot P_i + \overline{\text{dec}}_j \cdot \bar{p}_{i+1} \cdot P_i & \text{se } s \leq m \\ \overline{\text{zero}}_j \cdot \bar{p}_{m+1} \cdot P_i + \overline{\text{dec}}_j \cdot \bar{p}_{i+1} \cdot P_i & \text{altrimenti} \end{cases}$$

Con questa modifica la CM M termina \iff il processo CCS TCM_M è visimile weak a $\checkmark \cdot \mathbf{0}$.

Le bisimilarità weak e strong sono indecidibili. Lo sono neanche $\text{sort}(p)$ e la raggiungibilità. Addirittura la bisimulation equivalence per CCS finite-net perché lo è anche per le reti di Petri finite.

Definiamo **calcolabile** quando esiste un algoritmo; **effettivamente calcolabile** se capiamo com'è fatto questo algoritmo.

Provando a definire una **2CM** abbiamo

$$\text{CM}_{M(v_1, v_2)} = (\nu L)(R_1 \mid R_2 \mid B(v_1, v_2))$$

Dove

- $B(v_1, v_2)$ inizializza i due contatori con valori godelizzati e termina con la costante P_1 .
- $(i, \text{Inc}(r_j))$ è definita come $P_i = \overline{\text{inc}}_j.P_{i+1}$
- $(i, \text{DecJump}(r_j, s))$ è definita come $P_i = \overline{\text{zero}}_j.P_s + \overline{\text{dec}}_j.P_{i+1}$

I due contatori/registri sono definiti nel medesimo modo di 3CM.

$$R_j \stackrel{\text{def}}{=} \text{zero}_j.R_j + \text{inc}_j.((\nu a)(R_{j_1} \mid a.R_j))$$

$$R_{j_1} \stackrel{\text{def}}{=} \text{dec}_j.\bar{a}.\mathbf{0} + \text{inc}_j.((\nu b)(R_{j_2} \mid b.R_{j_1}))$$

$$R_{j_2} \stackrel{\text{def}}{=} \text{dec}_j.\bar{b}.\mathbf{0} + \text{inc}_j.((\nu a)(R_{j_1} \mid a.R_{j_2}))$$

In totale bastano 8 azioni per modellare una qualunque 2CM. Bastano $(m + 1) + 6$ costanti per modellare un qualunque Turing machine universale (UTM) con una 2CM (m = numero di istruzioni della 2CM che simula la UTM e 6 sono le costanti per i due contatori).

Quindi CCS(h, k) è Turing completo, dove $h = m + 7$ e $k = 8$, ma con m grande.

Un risultato noto è CCS(25, 12) attraverso una simulazione diretta di UTM con 15 stati e 2 simboli.

5.7.7. CCS Full

Non si ha nessun vincono sintattico. È più espressivo di CCS Finitary perché se p è un processo di CCS Finitary, allora il suo $\text{sort}(p)$ è finito (è il sotto insieme di azioni che sintatticamente occorrono libere in p). Il processo CCS Full A_0 , dove le infinite costanti A_k sono definite come $A_k = a_k.A_{k+1}$ per $k \geq 0$ è tale che $\text{sort}(A_0)$ è infinito = $\{a_0, a_1, \dots\}$.

5.8. CCS Value Passing

È un'estensione tale che i valori di input e output possono “passare” dei valori.

$$p ::= \mathbf{0} \mid a(x).q \mid \bar{a}(e).q \mid p + p \mid \text{if } b \text{ then } p \text{ else } p$$

$$q ::= p \mid q|q \mid (\nu a)q \mid c(x_1, \dots, x_n)$$

Un termine q è un processo se ogni occorrenza di una variabile è legata da input o definizione di costante.

Ad esempio, in $A(x) = \overline{\text{out}}(x + y).B$ la y non è legata.

Esempio

Un buffer ad una posizione che fa l'output $n!$ del fattoriale del suo input n :

$$\text{Fact} \stackrel{\text{def}}{=} \text{in}(x).F(x, 1)$$

$$F(x, y) \stackrel{\text{def}}{=} \text{if } x = 0 \text{ then } \overline{\text{out}}(y).\text{Fact} \text{ else } \tau.F(x - 1, x \times y)$$

A livello di semantica si ha

- Per un $\nu \in D$, chiamata *early* si ha

$$(\text{In}) \frac{}{a(x).p \xrightarrow{a(\nu)} p[v/x]}$$

mentre, in caso di *late*, diviene

$$(\text{In}) \frac{}{a(x).p \xrightarrow{a(\nu)} p}$$

$$\bullet \quad (\text{Tau}) \frac{}{\tau.p \xrightarrow{\tau} p}$$

- Se e ha valore ν

$$(\text{Out}) \frac{}{\bar{a}(e).p \xrightarrow{\bar{a}(\nu)} p}$$

ed avviene durante la comunicazione.

- Se $C(x_1, \dots, x_n) \stackrel{\text{def}}{=} p$ e per ogni e_i ha valore ν_i

$$(\text{Cons}) \frac{p[\nu_1/x_1, \dots, \nu_n/x_n] \xrightarrow{\mu} p'}{C(e_1, \dots, e_n) \xrightarrow{\mu} p'}$$

- Se b ha valore **true**

$$(\text{Then}) \frac{p \xrightarrow{\mu} p'}{\text{if } b \text{ then } p \text{ else } q \xrightarrow{\mu} p'}$$

- Se b ha valore **false**

$$(\text{Else}) \frac{q \xrightarrow{\mu} q'}{\text{if } b \text{ then } p \text{ else } q \xrightarrow{\mu} q'}$$

Dato un processo p in VP-CSS, è possibile ottenere un p' in CCS t.c. che i due generino LTS isomorfi.

L'idea dell'encoding è quella di avere un input $a(x)$ che genera un'azione $a_{\nu'}$ per ogni valore ν o costante nei dati.

- $\llbracket 0 \rrbracket = \mathbf{0}$
- $\llbracket a(x).p \rrbracket = \sum_{\nu \in D} a_{\nu} \cdot \llbracket p[v/x] \rrbracket$
- $\llbracket p_1 + p_2 \rrbracket = \llbracket p_1 \rrbracket + \llbracket p_2 \rrbracket$
- $\llbracket \tau.p \rrbracket = \tau \cdot \llbracket p \rrbracket$
- $\llbracket \bar{a}(e).p \rrbracket = \bar{a}_{\nu} \cdot \llbracket p \rrbracket$ se e ha valore ν
- $\llbracket (\nu a)p \rrbracket = (\nu \{a_{\nu} \mid \nu \in D\}) \llbracket p \rrbracket$
- $\llbracket \text{if } b \text{ then } p \text{ else } q \rrbracket = \begin{cases} \llbracket p \rrbracket & \text{if } b \text{ ha valore true} \\ \llbracket q \rrbracket & \text{if } b \text{ ha valore false} \end{cases}$

Per ogni definizione di costante $A(x_1, \dots, x_n) \stackrel{\text{def}}{=} p$ si ha una famiglia di costanti per ogni vettore $(\nu_1, \dots, \nu_n) \in D^n$ come $A_{\nu_1, \dots, \nu_n} \stackrel{\text{def}}{=} \llbracket p[\nu_1/x_1, \dots, \nu_n/x_n] \rrbracket$.

Se l'insieme dei dati D non fosse finito, avremmo somma infinita in corrispondenza di input, infinite costanti per implementarne una di VP-CCS e restrizioni su insiemi infiniti.

Proposizione Dato un insieme di valori finiti D , l'LTS per il VP-CCS p è isomorfo all'LTS di CCS puro $\llbracket p \rrbracket$.

Dimostrazione Si può vedere come $p \xrightarrow{a(\nu)} p' \Rightarrow \llbracket p \rrbracket \xrightarrow{a_{\nu}} \llbracket p' \rrbracket$ e $\llbracket p \rrbracket \xrightarrow{a_{\nu}} q \Rightarrow \exists p'.q = \llbracket p' \rrbracket \wedge p \xrightarrow{a(\nu)} p'$. Si fa la cosa analoga per $\bar{a}(\nu)/\bar{a}_{\nu}$ e τ . Poi si prova per induzione sulla prova delle transizioni. Alla fine, la biiezione f mappa un VP-CCS processo p' raggiungibile da p con $\llbracket p' \rrbracket$. ■

In un VP-CCS valido come $a(x).\bar{x}(b).C$ si vede come il valore ν ricevuto da a viene poi usato come nome di canale per trasmettere il valore b . Se i nomi dei canali diventano valori trasmissibili, allora il linguaggio permette mobilità dei canali e struttura a flow-graph riconfigurabili dinamicamente. Questi aspetti sono trattati nel π -calcolo.

La differenza è che in VP-CCS l'insieme delle azioni I/O non ha nulla a che fare con i valori dei canali che supponiamo nulli; nel π -calcolo sono lo stesso insieme.

Esempio

Stack Una specifica sequenziale può essere

$$\text{Stack}(\epsilon) \doteq \text{empty.Stack}(\epsilon) + \text{push}(x).\text{Stack}(x)$$

$$\text{Stack}(x\sigma) \doteq \overline{\text{pop}}(x).\text{Stack}(\sigma) + \text{push}(y).\text{Stack}(yx\sigma)$$

Il parametro σ può crescere indefinitamente, quindi servono infinite costanti in CCS puro. Con D di k elementi possiamo usare $2k + 1$ costanti in CCS puro definendo

$$S \stackrel{\text{def}}{=} \text{empty}.S + \text{push}(x).((\nu a)(S_1(x)|a.S))$$

$$S_1(x) \stackrel{\text{def}}{=} \overline{\text{pop}}(x).\bar{a}.\mathbf{0} + \text{push}(y).((\nu b)(S_2(y)|b.S_1(x)))$$

$$S_2(x) \stackrel{\text{def}}{=} \overline{\text{pop}}(x).\bar{b}.\mathbf{0} + \text{push}(x).((\nu a)(S_1(y)|a.S_2(x)))$$

Esempio

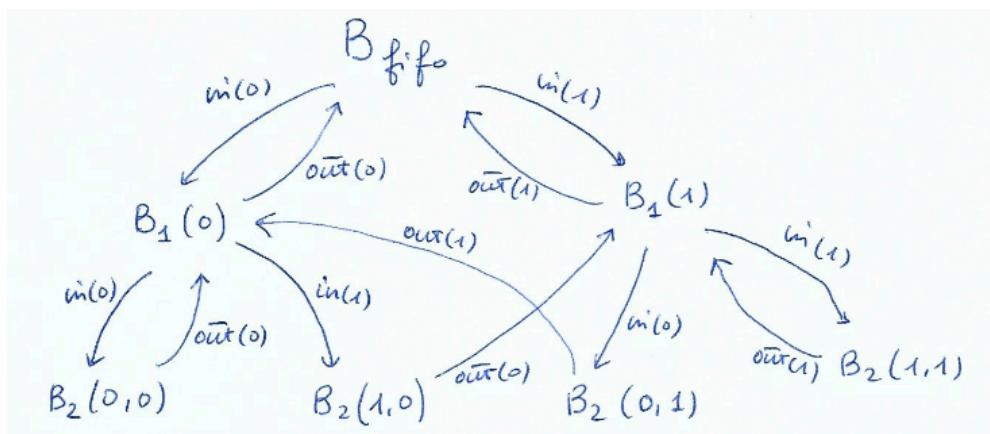
Si definiscono due tipologie di buffer diversi:

1.

$$B_{\text{fifo}} \stackrel{\text{def}}{=} \text{in}(x).B_1(x)$$

$$B_1(x) \stackrel{\text{def}}{=} \text{in}(y).B_2(y, x) + \overline{\text{out}}(x).B_{\text{fifo}}$$

$$B_2(x, y) \stackrel{\text{def}}{=} \overline{\text{out}}(y).B_1(x)$$

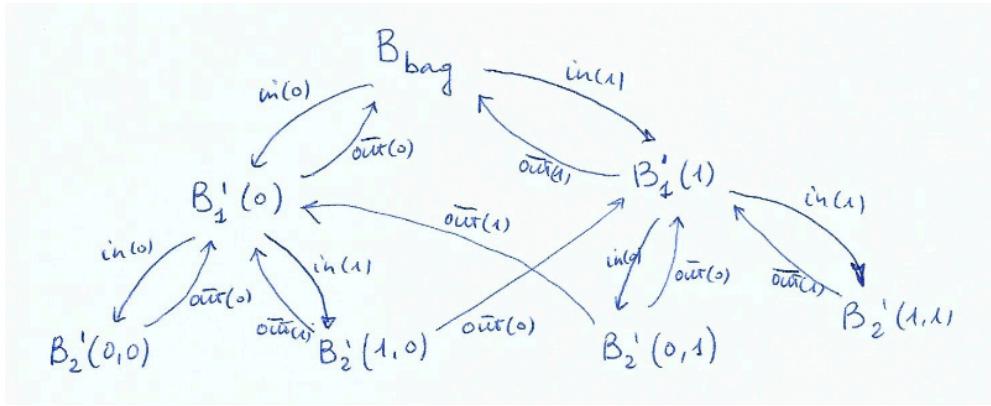


2.

$$B_{\text{bag}} \stackrel{\text{def}}{=} \text{in}(x).B'_1(x)$$

$$B'_1(x) \stackrel{\text{def}}{=} \text{in}(y).B'_2(y, x) + \overline{\text{out}}(x).B_{\text{bag}}$$

$$B'_2(x, y) \stackrel{\text{def}}{=} \overline{\text{out}}(x).B'_1(y) + \overline{\text{out}}(y).B'_1(x)$$



5.9. CCS[⊕]

Utilizzando questo operatore si ha la possibilità di fare

$$\frac{}{p \oplus q \xrightarrow{\tau} p}$$

$$\frac{}{p \oplus q \xrightarrow{\tau} q}$$

la scelta di p o q è presa internamente, senza impattare sull'ambiente. Essa si basa su diverse proprietà algebriche come

- commutatività $p \oplus q \sim q \oplus p$
- associatività $p \oplus (q \oplus r) \cong (p \oplus q) \oplus r$
- idempotenza $p \oplus p \approx p$
- assorbimento $p \oplus 0 \cong p$

dunque le bisimulazioni strong e weak sono congruenze per la scelta interna.

L'encoding in CCS avviene, in aggiunta alle regole viste prima, come

$$[p_1 \oplus p_2] = \tau.[p_1] + \tau.[p_2]$$

5.10. CCS^{hide}

È il linguaggio

$$p ::= \mathbf{0} \mid \mu.p \mid p + p \mid p|p \mid (\iota a)p \mid (\nu a)p \mid C$$

con l'aggiunta dunque di

$$\frac{p \xrightarrow{\mu} p'}{(\iota a)p \xrightarrow{\mu} (\iota a)p'}$$

con $\mu, \bar{\mu} \neq a$

$$\frac{p \xrightarrow{\mu} p'}{(\iota a)p \xrightarrow{\mu} (\iota a)p'}$$

con $\mu = a \vee \bar{\mu} = a$.

- (i) $(\iota a)p \sim p$ if $a \notin fn(p)$
- (ii) $(\iota a)((\iota b)p) \sim (\iota b)((\iota a)p)$
- (iii) $(\iota a)(\mu.p) \sim \begin{cases} \tau.(\iota a)p & \text{if } \mu = a \text{ or } \mu = \bar{a} \\ \mu.(\iota a)p & \text{otherwise} \end{cases}$
- (iv) $(\iota a)(p+q) \sim (\iota a)p + (\iota a)q$
- (v) $(\iota a)p|q \sim (\iota a)(p|q)$ if $a \notin fn(q)$
- (vi) $p|(\iota a)q \sim (\iota a)(p|q)$ if $a \notin fn(p)$
- (vii) $(\iota a)p \sim (\iota b)(p\{b/a\})$ if $b \notin fn(p) \cup bn(p)$
- (viii) $(\iota a)p+q \sim (\iota a)(p+q)$ if $a \notin fn(q)$
- (ix) $p+(\iota a)q \sim (\iota a)(p+q)$ if $a \notin fn(p)$
- (x) $(\iota a)((\nu a)p) \sim (\nu a)p$
- (xi) $(\nu a)((\iota a)p) \sim (\iota a)p$
- (xii) $(\iota a)((\nu b)p) \sim (\nu b)((\iota a)p)$ if $a \neq b$
- (xiii) $(\iota a)A \sim \begin{cases} A & \text{if } a \notin fn(A) \\ A^a & \text{otherwise, where } A^a \stackrel{\text{def}}{=} (\iota a)q \text{ if } A \stackrel{\text{def}}{=} q \end{cases}$

L'operatore hide è derivato, dunque si può mappare ogni processo p di CCS^{hide} in un q di CCS tale che sia up to $p \sim q$.

$$[(\iota a)p] = (\nu a)([p] \mid A_a)$$

$$\text{con } A_a \stackrel{\text{def}}{=} a.A_a + \bar{a}.A_{\bar{a}}$$

Un'alternativa è quella di trovare un processo q tale up to $p \approx^c q$.

$$[(\iota a)p] = [p]\{\tau/a\}$$

5.11. CCS^{rel}

Usa un operatore di scoprimento che converte tutte le azioni a di p in b (idem per \bar{a} e \bar{b}) chiamato relabeling $p[b/a]$.

$$\frac{p \xrightarrow{\mu} p'}{p[b/a] \xrightarrow{\mu} p'[b/a]}$$

con $\mu \neq a \wedge \mu \neq \bar{a}$

$$\frac{p \xrightarrow{\alpha} p'}{p[b/a] \xrightarrow{\beta} p'[b/a]}$$

con $(\alpha = a \wedge \beta = b) \vee (\alpha = \bar{a} \wedge \beta = \bar{b})$.

L'unica regola che non vale è $(p|q)[b/a] \sim p[b/a] \mid q[b/a]$.

Come derivazione, sempre up to \sim si ha

$$[p[b/a]] = [p]\{b/a\}$$

però non vale il parallelo.

5.12. CSS Full relabeling

Una funzione $f : L \mapsto L$ con L insieme di azioni visibili estesi ad Act come $f(\bar{a}) = \overline{f(a)}$ e $f(\tau) = \tau$.

$$\frac{p \xrightarrow{\mu} p'}{p[f] \xrightarrow{f(\mu)} p'[f]}$$

È più potente perché può essere usato per generare un processo che non è bisimulazione equivalente ad un processo CCS finitario. Assumendo $\text{Act} = \{a_i \mid i \in \mathbb{N}\}$ e $f(a_i) = a_{i+1}$. $A \stackrel{\text{def}}{=} a_0.(A[f])$ genera

$$A \xrightarrow{a_0} A[f] \xrightarrow{a_1} ((A[f])[f]) \xrightarrow{a_2} (((A[f])[f])[f]) \xrightarrow{a_3} \dots$$

però non è equivalente al processo CCS finitario p , dato che $\text{sort}(p)$ è sempre finito. Quindi questa versione di CCS con full relabeling è più potente.

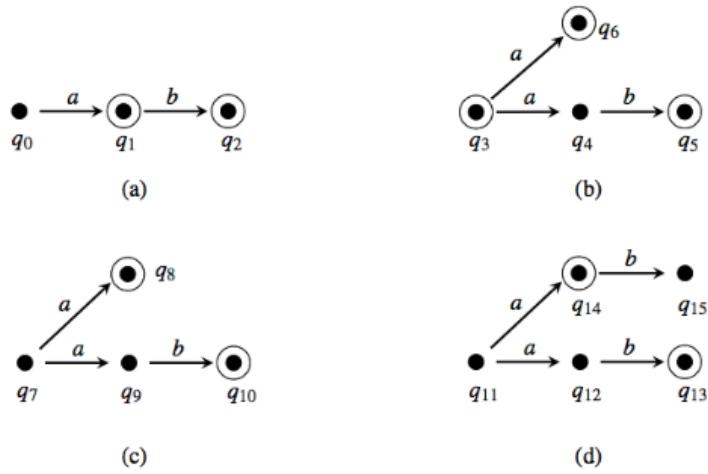
5.13. Composizione sequenziale

Presi p e q , si costruisce un terzo processo $p \cdot q$ il cui comportamento è dato dall'esecuzione di p come primo e successivamente dal q , ma questo solo se e quando p riesce a terminare con successo.

$p \downarrow$ vuol dire che p potrebbe terminare con successo.

$$\frac{\begin{array}{c} p \xrightarrow{\mu} p' \\ p \xrightarrow{\mu} p' \cdot q \end{array}}{p \downarrow \quad \frac{p \xrightarrow{\mu} q'}{p \cdot q \xrightarrow{\mu} q'}}$$

Preso un LTS con stati finali $TS = (Q, A, \rightarrow, F)$ con F come stati finali, un $p \in F$ è equivalente a dire $p \downarrow$.



5.13.1. F-trace

Una f-trace è una traccia che finisce su di uno stato finale. Gli esempi, sopra, (a), (c) e (d) lo sono.

Un f-trace completo è un f-trace che finisce su di un deadlock. Gli esempi, sopra, (a) e (d) lo sono.

Un weak f-trace è un weak trace che finisce su di uno stato finale.

Un weak f-trace completo è un weak f-trace che finisce su di uno stato che può eseguire nessun'azione osservabile.

5.13.2. F-bisimulation

Per ogni coppia (q, q') in una bisimulazione R , una f-bisimulazione è una bisimulazione tale che vale $q \in F \iff q' \in F$.

$$\sim_f = \bigcup \{R \subseteq Q \times Q \mid R \text{ è una f-bisimulazione}\}$$

5.13.3. Weak F-bisimulation

Per ogni LTS $TS = (Q, A \cup \{\tau\}, \rightarrow, F)$ una weak f-bisimulation è una relazione $R \subseteq (Q \times Q)$ t.c.

- R è weak bisimulation
- $(q, q') \in R \wedge q \in F \Rightarrow \exists q'' \in F \text{ t.c. } q' \xrightarrow{\varepsilon} q'' \wedge (q, q'') \in R$
- $(q, q') \in R \wedge q' \in F \Rightarrow \exists q'' \in F \text{ t.c. } q \xrightarrow{\varepsilon} q'' \wedge (q'', q) \in R$

q weak f-bisimile a q' è segnato come $q \approx_f q'$.

5.13.4. Rooted weak f-bisimulation

Denotato come $q_1 \approx_f^c q_2$ se $\forall \mu \in A \cup \{\tau\}$

- $\forall q_1'. q_1 \xrightarrow{\mu} q_1', \exists q_2'. q_2 \xrightarrow{\mu} q_2' \wedge q_1' \approx_f q_2'$
- $\forall q_2'. q_2 \xrightarrow{\mu} q_2', \exists q_1'. q_1 \xrightarrow{\mu} q_1' \wedge q_1' \approx_f q_2'$
- $q_1 \in F \iff q_2 \in F$

5.14. Finite BPA

$$p ::= \mathbf{0} \mid \mathbf{1} \mid \mu \mid p + p \mid p \cdot p$$

$$\frac{}{\mu \xrightarrow{\mu} \mathbf{1}}$$

$$\frac{p \xrightarrow{\mu} p'}{p \cdot q \xrightarrow{\mu} p' \cdot q}$$

$$\frac{p \downarrow \quad q \xrightarrow{\mu} q'}{p \cdot q \xrightarrow{\mu} p' \cdot q'}$$

$$\frac{p \xrightarrow{\mu} p'}{p + q \xrightarrow{\mu} p'}$$

$$\frac{q \xrightarrow{\mu} q'}{p + q \xrightarrow{\mu} q'}$$

$$\frac{}{\mathbf{1} \downarrow}$$

$$\frac{p \downarrow}{(p + q) \downarrow}$$

$$\frac{q \downarrow}{(p + q) \downarrow}$$

$$\frac{p \downarrow \quad q \downarrow}{(p \cdot q) \downarrow}$$

Esempio

1, 0 + 1, b · c + 1, (a + 1) · (b · c + 1) sono stati finali ↓.

0, a, 1 · 0, a + b · c non lo sono.

Proprietà parlando, si vede come **0 ~ 1** ma **0 ~f 1**. Idem **p + 1 ~ p** ma **p + 1 ~f p**. Ad esempio, **a + 1 ~f a** perché solo **a + 1** è finale.

Per la composizione sequenziale si distingue la f-bisimilarità

$$p \cdot (q \cdot r) \sim_f (p \cdot q) \cdot r$$

$$0 \cdot p \sim_f 0$$

$$1 \cdot p \sim_f p$$

$$(p + q) \cdot r \sim_f p \cdot r + q \cdot r$$

e l'equivalenza f-trace

$$p \cdot 0 =_{\text{trf}} 0$$

$$r \cdot (p + q) =_{\text{trf}} r \cdot p + r \cdot q$$

Dunque

\sim_f e \approx_f sono composizioni sequenziali.

\approx_f non è una congruenza “per scelta”. $\tau \cdot a \approx_f a$ ma $\tau \cdot a + b \not\approx_f a + b$.

\sim_f e \approx_f^c sono congruenze per scelta.

\approx_f^c è la congruenza più grossolana contenuta in \approx_f .

5.15. BPA*

È un finite BPA con i loop. È il linguaggio usato per le espressioni regolari.

$$p ::= 0 \mid 1 \mid \mu \mid p + p \mid p \cdot p \mid p^*$$

$$\frac{p \in \mathcal{P}_{\text{BPA}^*}}{\frac{p^* \downarrow}{\frac{p \xrightarrow{\mu} p'}{\frac{}{p^* \xrightarrow{\mu} p' \cdot p^*}}}}$$

Esempio

$$(a \cdot b)^* \xrightarrow{a} (1 \cdot b) \cdot (a \cdot b)^* \xrightarrow{b} 1 \cdot (a \cdot b)^* \xrightarrow{a} (1 \cdot b) \cdot (a \cdot b)^*$$

$$a \cdot (b \cdot a)^* \xrightarrow{a} \mathbf{1} \cdot (b \cdot a)^* \xrightarrow{b} (\mathbf{1} \cdot a) \cdot (b \cdot a)^* \xrightarrow{a} \mathbf{1} \cdot (b \cdot a)^*$$

Per congruenza, $p \sim_f q \Rightarrow p^* \sim_f q^*$

Alcune proprietà per l'iterazione sono:

- $p^* \sim_f p \cdot p^* + \mathbf{1}$
- $p^* \sim_f (p + \mathbf{1})^*$
- $(p + q)^* \sim_f p^* \cdot (q \cdot (p + q)^* + \mathbf{1})$

Con annotazione sugli stati che sono finiti o meno, ogni processo BPA^* può generare un LTS-f a stati finiti.

La dimostrazione avviene per induzione strutturale che mostra come $\text{s-size}(p)$, upper-bound sul numero degli stati raggiungibili, sia un numero finito.

- $\text{s-size}(\mathbf{0}) = 1$
- $\text{s-size}(\mathbf{1}) = 1$
- $\text{s-size}(\mu) = 2$
- $\text{s-size}(p_1 + p_2) = \text{s-size}(p_1) + \text{s-size}(p_2) + 1$
- $\text{s-size}(p_1 \cdot p_2) = \text{s-size}(p_1) \cdot \text{s-size}(p_2)$
- $\text{s-size}(p^*) = \text{s-size}(p) + 1$

Non tutti gli LTS-f a stati finiti possono essere dei processi BPA^* per up to \sim_f . Però possono farlo per equivalenza a tracce up to f .

$$\forall p \in \mathcal{P}_{\text{BPA}^*}, Tr_f(p) = \mathcal{L}[p]$$

5.16. BPA

È un BPA finito con ricorsione.

$$p ::= \mathbf{0} \mid \mathbf{1} \mid \mu \mid p + p \mid p \cdot p \mid C$$

$$\frac{p \downarrow}{C \downarrow}$$

con $C \stackrel{\text{def}}{=} p$.

$$\frac{p \xrightarrow{\mu} p'}{C \xrightarrow{\mu} p'}$$

con $C \stackrel{\text{def}}{=} p$.

Per quanto riguarda la semantica a tracce è equivalente tanto quanto i linguaggi CCS a stati finiti. BPA è un sistema a stati infiniti.

Un contatore può essere rappresentato senza restrizioni come

$$\text{BC} \stackrel{\text{def}}{=} \text{zero} \cdot \text{BC} + \text{inc} \cdot (S \cdot \text{BC})$$

$$S \stackrel{\text{def}}{=} \text{dec} + \text{inc} \cdot (S \cdot S)$$

Quindi ad ogni inc aggiungo una S .

L'equivalenza a tracce è indecidibile per BPA (idem per BPP), mentre l'equivalenza per bisimulazione sì (anche se algoritmo costoso). Non si è ancora risolto il problema della decidibilità per la bisimulazione weak.

5.17. Process Algebra

PA aggiunge la composizione asincrona parallela a BPA (quindi il parallelo non permette sync).

$$p ::= \mathbf{0} \mid \mathbf{1} \mid \mu \mid p + p \mid p \cdot p \mid p|p \mid C$$

$$\frac{\begin{array}{c} p \downarrow \quad q \downarrow \\ (p|q) \downarrow \\ p \xrightarrow{\mu} p' \\ \hline p|q \rightarrow p'|q \end{array}}{\begin{array}{c} q \xrightarrow{\mu} q' \\ \hline p|q \rightarrow p|q' \end{array}}$$

con le proprietà algebriche

- $p|(q|r) \sim_f (p|q)|r$
- $p|q \sim_f q|p$
- $p|\mathbf{1} \sim_f p$
- $p|\mathbf{0} \sim_f p \cdot \mathbf{0}$

Questo linguaggio non è Turing completo perché non si è ancora dimostrato il problema sulla reach. Non si sa se la bisimilarità è decidibile. La bisimilarità weak è indecidibile.

5.18. PAER

Estensione di PA.

$$p ::= \mathbf{0} \mid \mathbf{1} \mid \mu \mid p + p \mid p \cdot p \mid p|p \mid C$$

$$q ::= p \mid (\nu a)q$$

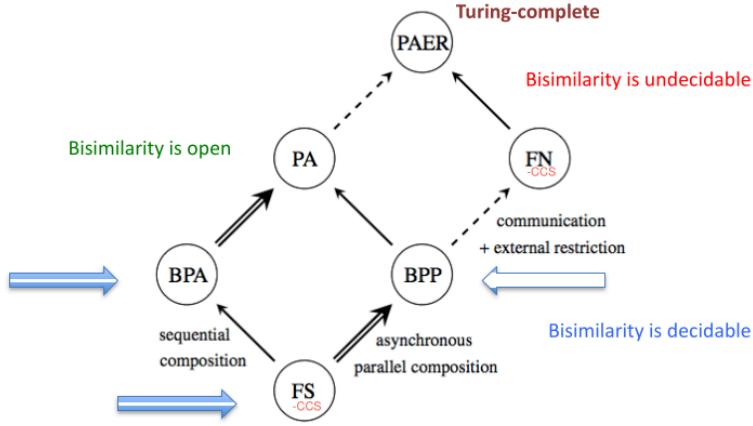
$$\frac{\begin{array}{c} p \xrightarrow{\alpha} p' \quad q \xrightarrow{\bar{\alpha}} q' \\ \hline p|q \xrightarrow{\tau} p'|q' \end{array}}{\begin{array}{c} p \downarrow \\ (\nu a)p \downarrow \\ p \xrightarrow{\mu} p' \\ \hline (\nu a)p \rightarrow (\nu a)p' \end{array}}$$

con $\mu \neq a, \bar{a}$

È Turing completo. Infatti, un contatore può essere visto come

$$\text{CM}_{M(\nu_1, \nu_2, \nu_3)} \stackrel{\text{def}}{=} (\nu L)(P_1 \mid \dots \mid P_m \mid R_1 \mid R_2 \mid R_3 \mid B_{(\nu_1, \nu_2, \nu_3)})$$

qui non si usano restrizioni ma solo una sequenza di composizioni.



5.19. CCS^{seq}

Estensione di CCS con composizione sequenziale (a meno di bisimilità f-weak).

$$p ::= \sum_{j \in J} \mu_j \cdot p_j \mid p|p \mid p \cdot \mid (\nu a)p \mid C$$

Gli addendi iniziano sempre con un prefisso. Non è consentito fare roba del tipo $a.\mathbf{0} + \mathbf{0}$ a causa della somma con $\mathbf{0}$.

Lo stato finale è $\mathbf{0}$ invece di $\mathbf{1}$. Non c'è una sintassi terminale per $p + q$.

$$\begin{array}{c} \overline{\mathbf{0} \downarrow} \\ \frac{p \downarrow \quad q \downarrow}{(p \cdot q) \downarrow} \\ \frac{p \downarrow \quad q \downarrow}{(p|q) \downarrow} \\ \frac{p \downarrow}{(\nu a)p \downarrow} \\ \frac{p \downarrow}{C \downarrow} \end{array}$$

con $C \stackrel{\text{def}}{=} p$.

Non c'è proprio uno stato non-finale base, ma un esempio di deadlock stato non-finale è $(\nu a)a.\mathbf{0}$.

Dato che p e q devono iniziare con un prefisso, $p + q$ non può essere finale.

$$\begin{array}{c} \frac{p \xrightarrow{\mu} p'}{p \cdot q \xrightarrow{\mu} p' \cdot q} \\ \frac{p \downarrow \quad q \xrightarrow{\mu} q'}{p \cdot q \xrightarrow{\mu} q'} \end{array}$$

$$\begin{aligned}
(i) \quad & p \cdot (q \cdot r) \sim_f (p \cdot q) \cdot r \\
(ii) \quad & p \cdot q \sim_f q \quad \text{if } p \downarrow \\
(iii) \quad & q \cdot p \sim_f q \quad \text{if } p \downarrow \\
(iv) \quad & (p + q) \cdot r \sim_f p \cdot r + q \cdot r \\
(v) \quad & p | q \sim_f q \quad \text{if } p \downarrow \\
(vi) \quad & (\forall a)p \sim_f p \quad \text{if } a \notin fn(p)
\end{aligned}$$

- 1) If $p \sim_f q$, then $p \cdot r \sim_f q \cdot r$ and $r \cdot p \sim_f r \cdot q$, for all $r \in \mathcal{P}_{CCS^{seq}}$,
2) If $p \approx_f q$, then $p \cdot r \approx_f q \cdot r$ and $r \cdot p \approx_f r \cdot q$, for all $r \in \mathcal{P}_{CCS^{seq}}$.

- 1) If $p \sim_f q$, then $p | r \sim_f q | r$ for all $r \in \mathcal{P}_{CCS^{seq}}$,
- 2) If $p \approx_f q$, then $p | r \approx_f q | r$ for all $r \in \mathcal{P}_{CCS^{seq}}$,
- 3) If $p \sim_f q$, then $(\forall a)p \sim_f (\forall a)q$ for all $a \in \mathcal{L}$,
- 4) If $p \approx_f q$, then $(\forall a)p \approx_f (\forall a)q$ for all $a \in \mathcal{L}$.

5.19.1. Da CCS^{seq} a CCS up to \approx_f

Esso è omomorfico (va su induzione strutturale de termine) per tutti gli operatori, ad eccezione della composizione sequenziale.

$$[\![p \cdot q]\!] = (\nu d)([\![p]\!]_d^e \mid \bar{d} \cdot ([\![q]\!]_d^e))$$

con $d, e \notin fn(p \cdot q) \cup bn(p \cdot q)$ non c'è in CCS per questo si può fare.

$[\![-]\!]_d^e$ quel d è un free name ed e è un bound name.

Dunque, quando p raggiunge lo stato finale p' , allora si ha $[\![p']\!]_d^e$ in cui può eseguire d come ultima azione e andare in deadlock.

$$[\![p_1 \mid p_2]\!] = (\nu e)(([\![p_1]\!]_d^e \mid [\![p_2]\!]_d^e) \mid \bar{e} \cdot \bar{e} \cdot d \cdot \mathbf{0})$$

qui p_1 è ristretto, dunque deve fare sync. Se termina con successo eseguirà e come ultima azione.

5.20. CSP sync multi-part

Un singolo step di sync può coinvolgere più parti (a differenza della sync in CCS).

$$\frac{\mu}{p \parallel_A q \xrightarrow{\mu} p' \parallel_A q}$$

con $\mu \notin A$.

$$\frac{\mu}{p \parallel_A q \xrightarrow{\mu} p \parallel_A q'}$$

con $\mu \notin A$.

$$\frac{p \xrightarrow{a} p' \quad q \xrightarrow{a} q'}{p \parallel_A q \xrightarrow{a} p' \parallel_A q'}$$

con $a \in A$.

Una sync tra p e q può avvenire solo se entrambe le parti eseguono la stessa azione $a \in A$. L'azione $p \parallel_A q$ eseguire l'azione a . Il risultato della sync è osservabile (a differenza di CCS). Si possono sync sulle azioni non appartenenti a A ma non possono fare azioni asincrone che appartengono ad A .

La composizione parallela di CSP non può essere trasformata in CCS: serve un'estensione del tipo Multi-CCS che aggiunge l'operatore di prefisso strong.

5.21. Problema dei filosofi a cena

Essere Turing-completo non garantisce la risoluzione di tutti i problemi concorrenti.

Ad esempio, non esiste una soluzione senza deadlock e/o senza divergenza per il *Problema dei filosofi a cena* perché CCS non ha un costrutto per atomicità che permetta una sincronizzazione per più parti. Esiste per CSP però, ad esempio

$$P_i \stackrel{\text{def}}{=} \text{think}.P_i + \text{up}_{(i,i+1)}. \text{eat}. \text{dn}_{(i,i+1)}.P_i \quad \forall i \in [0, 4]$$

$$\text{Phils} \stackrel{\text{def}}{=} P_0 \parallel_{\emptyset} P_1 \parallel_{\emptyset} P_2 \parallel_{\emptyset} P_3 \parallel_{\emptyset} P_4$$

visto da parte del filosofo e in cui $i + 1$ è computato mod 5 e le azioni up e dn hanno due parametri per la forchetta ad indice i .

$$F_i \stackrel{\text{def}}{=} \text{up}_{(i,i+1)}. \text{dn}_{(i,i+1)}.F_i + \text{up}_{(i-1,i)}. \text{dn}_{(i-1,i)}.F_i \quad \forall i \in [0, 4]$$

visto da parte della forchetta e in cui $i + 1$ e $i - 1$ sono computate mod 5.

$$\text{Forks} \stackrel{\text{def}}{=} F_0 \parallel_A F_1 \parallel_A F_2 \parallel_A F_3 \parallel_A F_4$$

con $A = \{\text{up}(i, i + 1), \text{dn}(i, i + 1) \mid i \in [0, 4]\}$.

L'intero sistema

$$\text{DP}_{\text{CSP}} \stackrel{\text{def}}{=} (\iota A)(\text{Phils} \parallel_A \text{Forks})$$

5.22. Multi-CCS

$$p ::= \mathbf{0} \mid \mu.q \mid \underline{\alpha}.p \mid p + p$$

$$q ::= p \mid q|q \mid (\nu a)q \mid C$$

È un'estensione di CCS.

Aggiunge il prefisso strong $\underline{\alpha}.p$ in cui α è la prima azione della transizione che continua con p . Le transizioni sono etichettate da sequenze di azioni osservabili. In più si ha la sync multiparti.

$$\frac{p \xrightarrow{\sigma} p'}{\underline{\alpha}.p \xrightarrow{\alpha \diamond \sigma} p'}$$

$$\text{con } \alpha \diamond \sigma = \begin{cases} \alpha & \text{if } \sigma = \tau \\ \alpha\sigma & \text{else} \end{cases}$$

La relazione di sincronizzazione Sync con $\beta \neq \alpha$ è definita come

$$\begin{array}{c} \overline{\text{Sync}}(\alpha, \bar{\alpha}, \tau) \\ \hline \sigma \neq \varepsilon \\ \overline{\text{Sync}}(\alpha\sigma, \bar{\alpha}, \sigma) \\ \hline \sigma \neq \varepsilon \\ \overline{\text{Sync}}(\bar{\alpha}, \alpha\sigma, \sigma) \end{array}$$

$$\frac{\text{Sync}(\alpha, \bar{\alpha}, \tau)}{\text{Sync}(\beta\sigma, \bar{\alpha}, \beta)}$$

$$\frac{\text{Sync}(\bar{\alpha}, \alpha, \tau)}{\text{Sync}(\bar{\alpha}, \beta\sigma, \beta)}$$

$$\frac{\text{Sync}(\sigma, \bar{\alpha}, \sigma_1)}{\text{Sync}(\beta\sigma, \bar{\alpha}, \beta\sigma_1)}$$

$$\frac{\text{Sync}(\bar{\alpha}, \sigma, \sigma_1)}{\text{Sync}(\bar{\alpha}, \beta\sigma, \beta\sigma_1)}$$

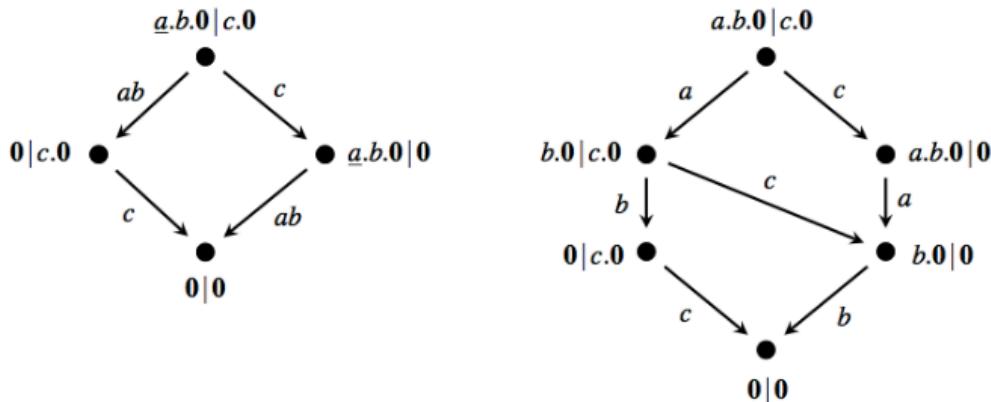
Sync è deterministica (per ogni $\sigma' \in \mathcal{A}$ contenente un'azione di α esiste una sola sequenza $\sigma'' \in \mathcal{A}$ t.c. $\text{Sync}(\sigma', \bar{\alpha}, \sigma'')$ e $\text{Sync}(\bar{\alpha}, \sigma', \sigma'')$)

Ad esempio,

$$\frac{\begin{array}{c} b.\mathbf{0} \xrightarrow{b} \mathbf{0} \\ \hline b.\mathbf{0} + c.\mathbf{0} \xrightarrow{b} \mathbf{0} \end{array}}{a.(b.\mathbf{0} + c.\mathbf{0}) \xrightarrow{ab} \mathbf{0}}$$

$a.\mathbf{0}$ non può far nulla.

Il prefisso strong garantisce la proprietà “tutto-o-niente” (eg: $\bar{a}.b.\mathbf{0}$ esegue solo la transizione a $\mathbf{0}$ con etichetta ab) e di “non-intralcio” (nessun processo interrompe l'esecuzione).



Il problema dei filosofi viene risolto come

$$P_i \stackrel{\text{def}}{=} \text{think}.P_i + \underline{\text{up}_i}.\underline{\text{up}_{i+1}}.\underline{\text{eat}}.\underline{\text{dn}_i}.\underline{\text{dn}_{i+1}}.P_i \quad \text{for } i \in [0, 1]$$

i $\underline{\text{up}_i}.\underline{\text{up}_{i+1}}$ devono stare in modo consecutivo.

think

$$\frac{\underline{dn_i.dn_{i+1}} \xrightarrow{} P_i \xrightarrow{\text{up}_i.up_{i+1}} \underline{up_i.up_{i+1}.eat.bn_i.bn_{i+1}.P_i}}{\underline{dn_i.bn_{i+1}.P_i}}$$

$$P_i \stackrel{\text{def}}{=} \underline{\text{think}.P_i + up_i.up_{i+1}.eat.bn_i.bn_{i+1}.P_i} \quad \text{for } i = 0, 1$$

$$F_i \stackrel{\text{def}}{=} \underline{\overline{up_i.bn_i}.F_i} \quad \text{for } i = 0, 1 \quad DP \stackrel{\text{def}}{=} (\nu L)((P_0 | P_1) | F_0) | F_1$$

$$\frac{\overline{up_1.P'_0 \xrightarrow{up_1} P'_0}}{\underline{up_0.up_1.P'_0 \xrightarrow{up_0.up_1} P'_0}}$$

$$\frac{\overline{\underline{\text{think}.P_0 + up_0.up_1.P'_0 \xrightarrow{up_0.up_1} P'_0}}}{\underline{P_0 \xrightarrow{up_0.up_1} P'_0}}$$

$$\frac{\overline{P_0 | P_1 \xrightarrow{up_0.up_1} P'_0 | P_1}}{\underline{(P_0 | P_1) | F_0 \xrightarrow{up_1} (P'_0 | P_1) | F'_0}}$$

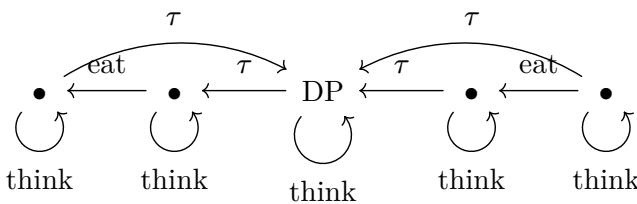
$$\frac{\overline{\overline{up_0}.F'_0 \xrightarrow{\overline{up_0}} F'_0}}{\underline{F_0 \xrightarrow{\overline{up_0}} F'_0}}$$

$$\frac{\overline{\overline{up_1}.F'_1 \xrightarrow{\overline{up_1}} F'_1}}{\underline{F_1 \xrightarrow{\overline{up_1}} F'_1}}$$

$$\frac{\overline{(P_0 | P_1) | F_0 \xrightarrow{\tau} ((P'_0 | P_1) | F'_0) | F'_1}}{\underline{(\nu L)((P_0 | P_1) | F_0) | F_1 \xrightarrow{\tau} (\nu L)((P'_0 | P_1) | F'_0) | F'_1}}$$

$$DP \xrightarrow{\tau} (\nu L)((P'_0 | P_1) | F'_0) | F'_1$$

24



la soluzione è corretta: totalmente distribuita (non usa nessuna memoria condivisa o coordinatore globale), simmetrica e priva di deadlock o divergenze.

In Multi CCS si ha anche la congruenza

$$\frac{p \equiv p' \xrightarrow{\sigma} q' \equiv q}{p \xrightarrow{\sigma} q}$$

di cui la composizione parallela non permette l'associatività però.

5.22.1. Interleaving

Due processi p e q sono bisimili interleaving $p \sim q$ se esiste una bisimulazione forte R su \mathcal{P}_M t.c. $(p, q) \in R$.

L'equivalenza per bisimulazione interleaving è una congruenza per quasi tutti gli operatori in Multi CCS.

Alcune proprietà sono:

- $\underline{\alpha}.p \sim \underline{\alpha}.q \quad \forall \alpha \in \mathcal{L} \cup \overline{\mathcal{L}}$
- $p + r \sim q + r \quad \forall \text{ sequential } r \in \mathcal{P}$
- $\mu.p \sim \mu.q \quad \forall \mu \in \text{Act}$
- $(\nu a)p \sim (\nu a)q \quad \forall a \in \mathcal{L}$

5.22.2. Semantica a step

Ogni transizione è etichettata da un numero definito di sequenze multiset che concorrono sottoprocessi che eseguono azioni allo stesso tempo. Sono bisimulazioni ordinarie su più ricchi LTS.

$$\frac{\overline{\mu.p \xrightarrow{s} p}}{\underline{\alpha}.p \xrightarrow{s} p'}$$

$$\frac{\overline{p \xrightarrow{s} p'}}{\underline{\alpha}.p \xrightarrow{s} p'}$$

$$\frac{\overline{p \xrightarrow{s} p'}}{p + q \xrightarrow{s} p'}$$

$$\frac{\overline{q \xrightarrow{s} q'}}{p + q \xrightarrow{s} q'}$$

$$\frac{\overline{p \xrightarrow{s} p'}}{p + q \xrightarrow{s} q'}$$

$$\frac{\overline{p \xrightarrow{s} p'}}{p|q \xrightarrow{s} p'|q}$$

$$\frac{\overline{q \xrightarrow{s} q'}}{p|q \xrightarrow{s} p'|q'}$$

$$\frac{\overline{p \xrightarrow{s} p'}}{C \xrightarrow{s} p'}$$

con $C \stackrel{\text{def}}{=} p$

$$\frac{p \xrightarrow{s} p'}{(\nu a)p \xrightarrow{s} (\nu a)p'}$$

con $a, \bar{a} \notin n(M)$

$$\frac{p \xrightarrow{s} p' \quad q \xrightarrow{s} q'}{p|q \xrightarrow{s} p'|q'}$$

con $\text{MSync}(M_1 \oplus M_2, M)$

$$\frac{\overline{\text{MSync}(M, M)} \quad \overline{\text{Sync}(\sigma_1, \sigma_2, \sigma)} \quad \overline{\text{MSync}(M \oplus \{\sigma\}, M')}}{\text{MSync}(M \oplus \{\sigma_1, \sigma_2\}, M')}$$

Un processo p che assicura durante la sua esecuzione di sincronizzare due sequenze atomiche anche non direttamente, è detto **ben formato**.

Presi $p, q \in \mathcal{P}_M$ processi ben formati. Se $p \sim_{\text{step}} q$ allora $p \sim q$.

Presi $p, q \in \mathcal{P}_M$ processi ben formati. Se $p \equiv q$ allora $p \sim_{\text{step}} q$.

Presi due processi Multi CCS p e q . Se $p \sim_{\text{step}} q$ allora

- $\mu.p \sim_{\text{step}} \mu.q \quad \forall \mu \in \text{Act}$
- $p|r \sim_{\text{step}} q|r \quad \forall r \in \mathcal{P}_M$
- $(\nu a)p \sim_{\text{step}} (\nu a)q \quad \forall a \in \mathcal{L}$

Presi due processi sequenziali p e q . Se $p \sim_{\text{step}} q$ allora

- $\underline{\alpha}.p \sim_{\text{step}} \underline{\alpha}.q \quad \forall \alpha \in \mathcal{L} \cup \overline{\mathcal{L}}$
- $p + r \sim_{\text{step}} q + r \quad \text{per ogni processo sequenziale } r$

6. Proprietà algebriche

6.1. Proprietà di \sim

6.1.1. +

Per quanto riguarda l'equivalenza strong \sim si ha che l'operatore di scelta $+$ forma un monoide commutativo con identità $\mathbf{0}$ ed è idempotente.

- $p + (q + r) \sim (p + q) + r$ (associatività)

Dimostrazione $R_1 = \{(p + (q + r), (p + q) + r) \mid p, q, r \in \mathcal{P}\} \cup \mathbf{Id}$ ■

- $p + q \sim q + p$ (commutativa)

Dimostrazione $R_2 = \{(p + q, q + r) \mid p, q \in \mathcal{P}\} \cup \mathbf{Id}$ ■

- $p + \mathbf{0} \sim p$ (neutralità)

Dimostrazione $R_3 = \{(p + \mathbf{0}, p) \mid p \in \mathcal{P}\} \cup \mathbf{Id}$ ■

- $p + p \sim p$ (idempotenza)

Dimostrazione $R_4 = \{(p + p, p) \mid p \in \mathcal{P}\} \cup \{(q, q) \mid q \in \mathcal{P}\}$ in cui si dimostra vedendo come se si muove la prima, allora si raggiunge uno stato q t.c. $(p', q') \in R$. Il primo membro è derivabile con premessa

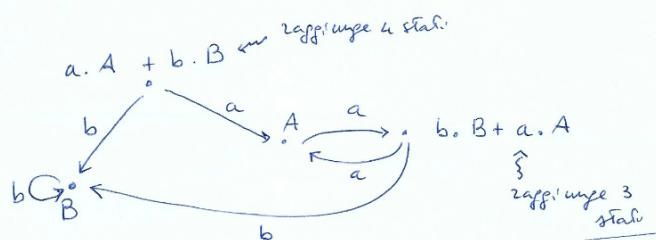
$$\frac{p \xrightarrow{\mu} p'}{p + p \xrightarrow{\mu} p'}$$

e il secondo è ovvio perché sono **Id**. ■

Queste leggi non valgono per l'isomorfismo:

- $p + q \xrightarrow{\text{iso}} q + p$

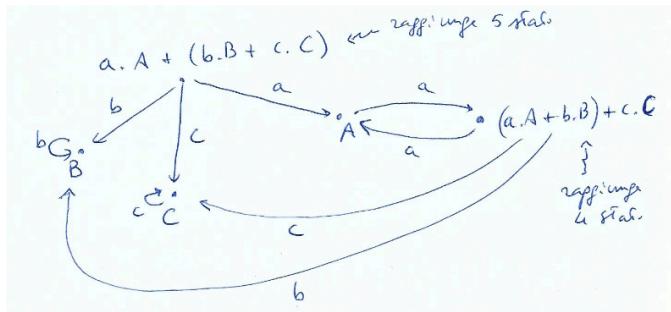
Dimostrazione Presi $A \doteq a.(b.B + a.A)$ e $B \doteq b.B$ si ha



■

- $p + (q + r) \xrightarrow{\text{iso}} (p + q) + r$

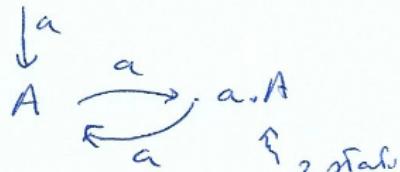
Dimostrazione Presi $A \doteq a.A$, $C \doteq c.C$ e $B \doteq b.B$ si ha



- $p + p \stackrel{\text{iso}}{\not\approx} p$

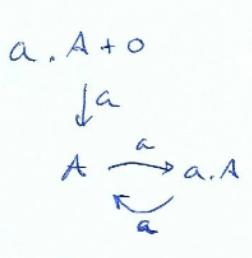
Dimostrazione Preso $p = a.A$ si ha

$$a \cdot A + a \cdot A \text{ con } 3 \text{ stati}$$



- $p + \mathbf{0} \stackrel{\text{iso}}{\not\approx} p$

Dimostrazione Preso $p = a.A$ si ha



Queste leggi valgono per l'equivalenze più deboli di bisimulazione (come l'equivalenza a tracce) e giustificano la scrittura $\sum_{1 \leq i \leq n} p_i$.

Distributività del prefisso rispetto alla somma La legge sopra descritta non vale per $\mu.(p + q) \not\approx \mu.p + \mu.q$ e neanche per simulation equivalence, ma solo per trace equivalence.

6.1.2. |

Per quanto riguarda l'equivalenza strong \sim si ha che l'operatore di parallelo $|$ forma un monoide commutativo con identità $\mathbf{0}$ e non è idempotente.

- $p|(q|r) \sim (p|q)|r$ (associatività)

Dimostrazione $R_1 = \{(p|(q|r), (p|q)|r) \mid p, q, r \in \mathcal{P}\} \blacksquare$

- $p|q \sim q|p$ (commutatività)

Dimostrazione $R_2 = \{(p|q, q|p) \mid p, q \in \mathcal{P}\} \blacksquare$

- $p|\mathbf{0} \sim p$ (neutralità)

Dimostrazione $R_3 = \{(p|\mathbf{0}, p) \mid p \in \mathcal{P}\}$ Si vede come se $p|\mathbf{0} \rightarrow p'|\mathbf{0}$ allora $p \rightarrow p'$, quindi $(p'|\mathbf{0}, p') \in R$ e viceversa se $p \rightarrow p'$ \blacksquare

- $p|p \not\sim p$

Dimostrazione Preso, ad esempio $a.0|a.0 \not\sim a.0 \blacksquare$

Queste leggi valgono per l'equivalenze più deboli di bisimulazione (come l'equivalenza a tracce) e per l'isomorfismo. Inoltre, giustificano la scrittura $\prod_{1 \leq i \leq n} p_i$.

Legge per la costante Se $C = p$ allora $C \sim p$.

Controesempio della legge per la costante con isomorfismo Preso $A = a.A$ si vede che l'LTS per A e per $a.A$ non sono isomorfi. Se $C \rightarrow p'$ allora $p \rightarrow p' \wedge (p', q') \in R$. Se $p \rightarrow p'$ allora $C \rightarrow p' \wedge (p', q') \in R$. (La legge di unfolding vale solo per le costanti).

6.1.3. (νa)

- $(\nu a)\mathbf{0} \sim \mathbf{0}$

Dimostrazione $R_1 = \{((\nu a)\mathbf{0}, \mathbf{0})\} \blacksquare$

- $(\nu a)((\nu b)p) \sim (\nu b)((\nu a)p) \quad \text{se } a \neq b$

Dimostrazione $R_2 = \{((\nu a)((\nu b)p'), (\nu b)((\nu a)p')) \mid p' \in \mathcal{P}\} \blacksquare$

- $(\nu a)((\nu a)p) \sim (\nu a)p$

Dimostrazione $R_3 = \{((\nu a)((\nu a)p'), (\nu a)p') \mid p' \in \mathcal{P}\} \blacksquare$

- $(\nu a)(\mu.p) \sim \begin{cases} \mathbf{0} & \text{se } \mu = a \vee \mu = \bar{a} \\ \mu.(\nu a)p & \text{altrimenti} \end{cases}$

Dimostrazione $R_4 = \{((\nu a)(\mu.p), \mu.(\nu a)p) \mid \mu \neq a, \mu \neq \bar{a}, p \in \mathcal{P}\} \cup \{((\nu a)(\mu.p), \mathbf{0}) \mid \mu \neq a, \mu \neq \bar{a}, p \in \mathcal{P}\} \cup \mathbf{Id} \blacksquare$

- $(\nu a)(p + q) \sim (\nu a)p + (\nu a)q$

Dimostrazione $R_5 = \{((\nu a)(p + q), (\nu a)p + (\nu a)q) \mid p, q \in \mathcal{P}\} \cup \mathbf{Id} \blacksquare$ Questa però non è del tutto "legale".

6.2. Legge di espansione / interleaving

Presi $p = \sum_{i=1}^n \mu_i \cdot p_i$ e $q = \sum_{j=1}^m \mu'_j \cdot q_j$ allora si ha

$$p|q \sim \sum_{i=1}^n \mu_i \cdot (p_i|q) + \sum_{j=1}^m \mu'_j \cdot (p|q_j) + \sum_{\overline{\mu}_i = \mu'_j} \tau \cdot (p_i|q_j)$$

Dimostrazione Se T_r è l'insieme finito di tutte le transizioni in uscita da r , allora $r \sim \sum_{(r, \mu_k, r_k) \in T_r} \mu_k \cdot r_k$ ■

Teorema (Non distributività del parallelo rispetto alla somma) $(p+q)|r \not\sim p|r + q|r$

Dimostrazione TODO

6.3. Free names $fn(p)$

I nomi liberi di un processo p , denotati come $fn(p)$, sono l'insieme $F(p, \emptyset)$ dove $F(p, I)$ l' I è l'insieme delle costanti del processo ed è definito da:

$$F(\mathbf{0}, I) = \emptyset$$

$$F(a.p, I) = F(\bar{a}.p, I) = F(p, I) \cup \{a\}$$

$$F(\tau.p, I) = F(p, I)$$

$$F(p+q, I) = F(p|q, I) = F(p, I) \cup F(q, I)$$

$$F((\nu a)p, I) = F(p, I) \setminus \{a\}$$

$$F(C, I) = \begin{cases} F(q, I \cup \{C\}) & \text{se } C \stackrel{\text{def}}{=} q \wedge C \notin I \\ \emptyset & \text{se } C \in I \end{cases}$$

Quindi $fn(p)$ è calcolabile solo per processi con $\text{Const}(p)$ finito.

Per ogni p in CCS finitary, l'insieme $fn(p)$ è finito.

Se $p \xrightarrow{\mu} p'$ allora $fn(p') \subseteq fn(p)$ e se $\mu \in \{a, \bar{a}\}$ allora $a \in fn(p)$.

$\text{sort}(p) \subseteq fn(p) \cup \overline{fn}(p) \cup \{\tau\}$ è superset decidibile di $\text{sort}(p)$ per tutti i p in CCS finitary.

Alcune proprietà che usano $fn(p)$ sono:

- $(\nu a)p \sim p$ se $a \notin fn(p)$

Dimostrazione $R_1 = \{((\nu a)p, p) \mid p \in \mathcal{P} \wedge a \notin fn(p)\}$ ■

- $(\nu a)p|q \sim (\nu a)(p|q)$ se $a \notin fn(q)$

Dimostrazione $R_2 = \{((\nu a)p|q, (\nu a)(p|q)) \mid p, q \in \mathcal{P} \wedge a \notin fn(q)\}$ ■

6.4. Bound names $bn(p)$

I nomi a bordo di un processo p , denotati come $bn(p)$, sono l'insieme $B(p, \emptyset)$ dove $B(p, I)$ l' I è l'insieme delle costanti del processo ed è definito da:

$$B(\mathbf{0}, I) = \emptyset$$

$$B(\mu.p, I) = B(p, I)$$

$$B(p+q, I) = B(p|q, I) = B(p, I) \cup B(q, I)$$

$$B((\nu a)p, I) = B(p, I) \cup \{a\}$$

$$B(C, I) = \begin{cases} B(q, I \cup \{C\}) & \text{se } C \not\equiv q \wedge C \notin I \\ \emptyset & \text{se } C \in I \end{cases}$$

Una proprietà che si lega anche ai nomi liberi è

- $(\nu a)p \sim (\nu b)(p\{b/a\})$ se $b \notin (fn(p) \cup bn(p))$

Dimostrazione $R_1 = \{((\nu a)p, (\nu b)(p\{b/a\})) \mid p \in \mathcal{P} \wedge b \notin (fn(q) \cup bn(p))\} \blacksquare$

6.5. Sostituzione

Insieme di associazioni $\{b_i/a_i\}_{i \in I}$. L'azione $a_i \in \mathcal{L}$ è rimpiazzata da $b_i \in \mathcal{L}$ quando applicata ad un termine.

Una sostituzione di questo tipo $\{b_i/a_i\}_{i \in I}$ è ammessa quando $\forall i \neq j. a_i \neq a_j \wedge \forall i, j. b_i \neq a_j$.

Esempi non ammessi sono: $\{a/a\}, \{b/a, c/a\}, \{b/a, a/b\}$.

Esempi ammessi sono: $\{b/a\}, \{b/a, b/c\}, \{b/a, d/c\}$.

Per iterare l'insieme delle sostituzioni ammesse si usa θ .

Una sostituzione è vuota ϵ se $|I| = 0$.

Una sostituzione è unaria se $|I| = 1$. Ad esempio $\{b/a\}$, ma è ammessa solo se $a \neq b$.

La composizione di una sostituzione ammessa è $\theta = \{b_1/a_1, \dots, b_n/a_n\}$ con una sostituzione unaria ammessa $\{b/a\}$ denotata da $\theta \circ \{b/a\}$, è definita t.c. $b \neq a_j \forall j = 1, \dots, n$ come

- $\{b_1/a_1, \dots, b_n/a_n\}$ se $\exists j. a = a_j$;
- $\{b'_1/a_1, \dots, b'_n/a_n, b/a\}$ se $\forall j = 1, \dots, n. a \neq a_j$ dove, se $a \neq b_i$ si ha $b' = b_i$, se $a = b_i$ si ha $b' = b$.

$\theta \circ \{b/a\}$ è una sostituzione ammessa.

Esempi di composizioni di sostituzioni sono: $\epsilon \circ \{b/a\} = \{b/a\}, \{b/a\} \circ \{b/a\} = \{b/a\}, \{b/a, d/c\} \circ \{d/b\} = \{d/a, d/c, d/b\}$.

Le costanti sono indicate da una sostituzione ammessa, che definiamo A_θ . A_ϵ sta per A . L'applicazione $A_{\theta \circ \{b/a\}}$ di una sostituzione unaria $\{b/a\}$ ad A_θ genera una costante $\theta \circ \{b/a\}$, cioè $A_{\theta \circ \{b/a\}}$.

La sostituzione sintattica $p\{b/a\}$ di un'azione b per un'altra a in un processo p è definita come

- $\mathbf{0}\{b/a\} = \mathbf{0}$
- $(a.p)\{b/a\} = b.(p\{b/a\})$
- $(\bar{a}.p)\{b/a\} = \bar{b}.(p\{b/a\})$
- $(\mu.p)\{b/a\} = \mu.(p\{b/a\})$ se $\mu \neq a, \bar{a}$
- $(p+q)\{b/a\} = p\{b/a\} + q\{b/a\}$
- $(p|q)\{b/a\} = p\{b/a\}|q\{b/a\}$
- $((\nu c)p)\{b/a\} = (\nu a)(p\{b/a\})$ se $c \neq a, b$
- $((\nu c)p)\{b/a\} = (\nu a)p$
- $((\nu b)p)\{b/a\} = \begin{cases} (\nu b)p & \text{se } a \notin fn(p) \\ (\nu c)((p\{\frac{c}{b}\})\{b/a\}) & \text{altrimenti, con } c \notin fn(p) \cup bn(p) \end{cases}$

Esempio

$A \doteq a.A + b.B$

$$B \doteq c.d.A + a.B$$

$$A\{c/a\} = A_{\{c/a\}} \text{ dove}$$

$$A_{\{c/a\}} \doteq c.A_{\{c/a\}} + b.B_{\{c/a\}}$$

$$B_{\{c/a\}} \doteq c.d.A_{\{c/a\}} + c.B_{\{c/a\}}$$

Esempio

$$A \doteq (\nu a)(a.b.A \mid \bar{a}.c.A)$$

$$A\{b/a\} = A \text{ perché } a \notin fn(A)$$

$$A\{a/b\} = A_{\{a/b\}} \text{ dove } A_{\{a/b\}} \doteq (\nu d)\left(d.a.A_{\{a/b\}} \mid \bar{d}.c.A_{\{a/b\}}\right)$$

6.6. Proprietà di \approx

$$\tau.p \approx p$$

Dimostrazione $R = \{(\tau.p, p) \mid p \in \mathcal{P}\} \cup \text{Id}$ è una weak bisimilarity ■

Vale anche per weak trace equivalence $=_{\text{wtr}}$ ma non vale per rooted weak bisimilarity \approx^c .

$$\forall \mu \text{ si ha che } \mu.p \approx^c \mu.q \iff p \approx q$$

Alcune proprietà che valgono sono:

- $\mu.\tau.p \approx^c \mu.p$

Dimostrazione Vale perché $\tau.p \approx p$ e allora $\mu.\tau.p \approx^c \mu.p$ ■

- $p + \tau.p \approx^c \tau.p$

Dimostrazione Vale perché

1. se $\tau.p \xrightarrow{\tau} p$, allora $p + \tau.p \xrightarrow{\tau} p \wedge p \approx p$
2. se $p + \tau.p \rightarrow p'$ allora $\tau.p \xrightarrow{\tau} p'$ con $p' \approx p'$ ■

- $\mu.(p + \tau.q) \approx^c \mu.(p + \tau.q) + \mu.q$

Dimostrazione Vale perché

1. se $\mu.(p + \tau.q) \xrightarrow{\mu} p.\tau.q$ allora $\mu.(p + \tau.q) + \mu.q \xrightarrow{\mu} p + \tau.q \wedge p + \tau.q \approx p + \tau.q$
2. se $\mu.(p + \tau.q) + \mu.q \xrightarrow{\mu} p$ allora $\mu.(p + \tau.q) \Rightarrow q \wedge q \approx q$
3. se $\mu.(p + \tau.q) + \mu.q \xrightarrow{\mu} p + \tau.q$ e questo è ovvio.

6.6.1. Lemma di Hennessy

Per ogni processo p e q , $p \approx q \iff (p \approx^c q \vee p \approx^c \tau.q \vee \tau.p \approx^c q)$

Dimostrazione La dimostrazione la sviluppiamo in due parti:

(\Leftarrow) Si hanno tre casi

- (1) Se $p \approx^c q$ allora $p \approx q$.
- (2) Se $p \approx^c \tau.q$ allora $p \approx \tau.q \approx q$.
- (3) Simmetrico al punto (2).

(\Rightarrow) Assumendo che $p \approx q$ si hanno tre casi

- (1) Se $\exists p'.p \xrightarrow{\tau} p' \approx q$ allora $p \approx^c \tau.q$.

Se $\tau.q \xrightarrow{\tau} q$ allora $p \xrightarrow{\tau} p'$ con $p' \approx q$.

Se $p \xrightarrow{\tau} p'$ con $p' \approx q$ allora $\tau.q \xrightarrow{\tau} q$.

Se $p \xrightarrow{\tau} p''$ con $p'' \approx q$ allora $\tau.q \xrightarrow{\tau} q \xrightarrow{\epsilon} q''$ con $p'' \approx q''$ perché $p \approx q$.

Se $p \xrightarrow{\alpha} p''$ allora $\tau.q \xrightarrow{\tau} q \xrightarrow{\alpha} q''$ con $p'' \approx q''$ perché $p \approx q$.

- (2) Simmetrico al punto (1), se $q \xrightarrow{\tau} q' \approx p$ allora $\tau.p \approx^c q$

- (3) Se né (1) né (2) valgono, allora si ha che

Se $p \xrightarrow{\alpha} p'$ allora $q \xrightarrow{\alpha} q'$ con $p' \approx q'$ (e qui la definizione di rooted weak bisimilarity \approx^c è rispettata).

Se $p \xrightarrow{\tau} p'$ allora $q \xrightarrow{\epsilon} q'$ con $p' \approx q'$, però $q' \neq q$, sennò saremmo nel primo sottopunto di (2).

Simmetricamente se è q a muovere per primo. ■

6.6.2. |

Per l'operatore di parallelo si ha

- $p|q \approx p|\tau.q$

Dimostrazione $R = \{(p'|q', p'|\tau.q') \mid p', q' \in \mathcal{P}\} \cup \mathbf{Id}$

$$(a) p|q \xrightarrow{\mu} s \begin{cases} p \xrightarrow{\mu} p' & s=p'|q \\ q \xrightarrow{\mu} q' & s=p|q' \\ p \xrightarrow{\alpha} p', q \xrightarrow{\bar{\alpha}} q' & s=p'|q' \end{cases}$$

I tre casi per (a) possono essere visti come

1. $p|\tau.q \xrightarrow{\mu} p'|\tau.q \wedge (p'|q, p'|\tau.q) \in R$
2. $p|\tau.q \xrightarrow{\tau} p|q \xrightarrow{\mu} p|q' \wedge (p|q', p|q') \in R(\mathbf{Id})$
3. $p|\tau.q \xrightarrow{\tau} p|q \xrightarrow{\tau} p'|q' \wedge (p'|q', p'|q') \in R(\mathbf{Id})$

$$(b) p|\tau.q \xrightarrow{\mu} s \begin{cases} p \xrightarrow{\mu} p' & s=p'|\tau.q \\ \tau.q \xrightarrow{\tau} q & s=p|q \end{cases}$$

I due casi per (b) possono essere visti come

1. $p|q \xrightarrow{\mu} p'|q \wedge (p'|q, p'|\tau.q) \in R$
2. $p|q \xrightarrow{\epsilon} p|q \wedge (p|q, p|q) \in R(\mathbf{Id})$

■

- $p|\tau.q \not\approx^c p|q$

Dimostrazione Se $p = a \wedge q = b$ allora $a|\tau.b \xrightarrow{\tau} a|b$ ma $a|b \not\xrightarrow{\tau}$ ■

- $p|\tau.q \approx^c \tau.(p|q)$

Dimostrazione

(a) Se $\tau.(p|q) \xrightarrow{\tau} p|q$ allora $p|\tau.q \xrightarrow{\tau} p|q \wedge p|q \approx p|q$

$$(b) \text{Se } p|\tau.q \xrightarrow{\mu} s \begin{cases} p \xrightarrow{\mu} p' & s=p'|\tau.q \\ \tau.q \xrightarrow{\tau} q & s=p|q \end{cases}$$

I due casi per (b) possono essere visti come

1. $\tau.(p|q) \xrightarrow{\tau} p|q \xrightarrow{\mu} p'|q \wedge p'|q \approx p'|\tau.q$
2. $\tau.(p|q) \xrightarrow{\tau} p|q \wedge p|q \approx p|q(\mathbf{Id})$

7. Congruenze comportamentali

Una relazione di equivalenza è una congruenza se è preservata dagli operatori (chiusa per contesti).

Se $P \propto Q$ allora

- $\mu.P \propto \mu.Q$
- $P + R \propto Q + R$
- $P|R \propto Q|R$
- $(\nu a)P \propto (\nu a)Q$

Se vale quanto sopra, la congruenza è detta comportamentale.

Equivalenza \Rightarrow Intercambiabilità

Congruenza \Rightarrow Intercambiabilità (ma in ogni contesto)

La congruenza non è compositiva rispetto al parallelo, dunque se $P \propto Q$ non è vero che $P|R \propto Q|R$ e il comportamento potrebbe cambiare.

7.1. Congruenza \Rightarrow Equivalence-checking compositivo

La congruenza è la base per poter fare ragionamento compositivo.

Per confrontare $p_1|p_2$ e $q_1|q_2$, se si dimostra che $p_1 \propto q_1$ e che $p_2 \propto q_2$, allora, per congruenza rispetto al parallelo, siamo sicuri che $p_1|p_2 \propto q_1|q_2$.

7.2. Congruenza \Rightarrow Minimizzazione compositiva

La congruenza è la base per avere minimizzazione compositiva.

Preso un processo $P = p_1 | p_2 | \dots | p_n$, per minimizzare rispetto alla congruenza scelta (ad esempio strong bisimilarity) da ogni p_j si ottiene un processo p'_j e quindi si compongono a due a due i processi sempre poi minimizzando il risultato.

Lo spazio degli stati risultante Q è spesso più piccolo di quello per P . Ma $P \propto Q$, quindi si possono fare le analisi su Q stesso.

7.3. Alcune congruenze e non

7.3.1. Isomorfismo

Esso non è una congruenza. Presi, ad esempio, $a.\mathbf{0}$ e $a.(\mathbf{0} + \mathbf{0})$ essi generano due LTS isomorfi. Preso il contesto $C[X] = X + a$. Si ha che $C[a.\mathbf{0}]$ e $C[a.(\mathbf{0} + \mathbf{0})]$ generano degli LTS non isomorfi (il primo ha 2 stati, il secondo ne ha 3). Quindi l'equivalenza per isomorfismo non è una congruenza per l'operatore di scelta $+$.

7.3.2. Strong bisimilarity \sim

Se $P \sim Q$ allora

- $\forall \mu \text{ ho } \mu.P \sim \mu.Q$
- $\forall R \text{ ho } P + Q \sim Q + R$
- $\forall R \text{ ho } P|R \sim Q|R$
- $\forall a \text{ ho } (\nu a)P \sim (\nu a)Q$

Dimostrazione Presa una bisimulazione S tale che $(P, Q) \in S$ si ha

- $S_1 = S \cup \{(\mu.P, \mu.Q) \mid \mu \in \text{Act}\}$ è bisimulazione.
- $S_2 = S \cup \{(P + R, Q + R) \mid R \text{ un processo CCS}\} \cup \mathbf{Id}$ è una bisimulazione.
- $S_3 = \{(P'|R', Q'|R') \mid (P', Q') \in S \wedge R \text{ un processo CCS}\}$ è una bisimulazione.

- $S_4 = \{((\nu a)P', (\nu a)Q') \mid (P', Q') \in S \wedge a \in L\}$ è una bisimulazione.

I casi simmetrici $R + P \sim R + Q$ e $R|P \sim R|Q$ derivano per commutatività da quelli operatori.

■

7.3.3. Trace equivalence $Tr(p)$

Se $Tr(p) = Tr(q)$ allora

- $\forall \mu \in \text{Act}$ ho $Tr(\mu.p) = Tr(\mu.a)$
- $\forall r \in \mathcal{P}$ ho $Tr(p+r) = Tr(q+r)$
- $\forall r \in \mathcal{P}$ ho $Tr(p|r) \sim Tr(q|r)$
- $\forall a \in \mathcal{L}$ ho $Tr((\nu a)p) = Tr((\nu a)q)$

TODO: aggiungere dimostrazione

7.3.4. Completed trace/simulation equivalence

La completed trace equivalence e la completed simulation equivalence non sono una congruenza per la restrizione.

TODO: aggiungere dimostrazione per $a.(b+c)$ e $a.b + a.c$

7.3.5. Weak bisimilarity \approx

Se $P \approx Q$ allora

- $\forall \mu$ ho $\mu.P \approx \mu.Q$
- $\forall R$ ho $P|R \approx Q|R$
- $\forall a$ ho $(\nu a)P \approx (\nu a)Q$

Dimostrazione Presa una bisimulazione weak S tale che $S \in (P, Q)$ si ha

- $S_1 = S \cup \{(\mu.P, \mu.Q) \mid \mu \in \text{Act}\}$ è bisimulazione weak.
- $S_2 = \{(P'|R', Q'|R') \mid (P', Q') \in S \wedge R$ un processo CCS $\}$ è una bisimulazione weak.
- $S_3 = \{((\nu a)P', (\nu a)Q') \mid (P', Q') \in S \wedge a \in L\}$ è una bisimulazione weak.

■

Non è una congruenza per l'operatore di scelta perché $\tau.a \approx a$ ma $\tau.a + b$ non è equivalente al processo $a + b$.

7.3.6. Rooted weak bisimilarity \approx^c

Se $P \approx^c Q$ allora

- $\forall \mu$ ho $\mu.P \approx^c \mu.Q$
- $\forall R$ ho $P + R \approx^c Q + R$
- $\forall R$ ho $P|R \approx^c Q|R$
- $\forall a$ ho $(\nu a)P \approx^c (\nu a)Q$

Dimostrazione Se $P \approx^c Q$ allora $P \approx Q$ allora $\mu.P \approx^c \mu.Q$.

Se $P + R \xrightarrow{\mu} S$ allora ho due possibilità:

1. $P \xrightarrow{\mu} S$, poiché $P \approx^c Q$, deve essere $Q \xrightarrow{\mu} Q'$ con $S \approx Q'$.
2. $R \xrightarrow{\mu} S$ ho $Q + R \xrightarrow{\mu} S$ con $S \approx S$.

Simmetricamente partendo $Q + R$.

TODO: Aggiungere la dimostrazione per $|$ e νa .

7.4. $\approx^c \subseteq \approx$

Assumendo che $fn(p) \cup fn(q) \notin \mathcal{L}$ allora $p \approx^c q \iff \forall r \in \mathcal{P}, p + r \approx q + r$.

Dimostrazione La si costruisce in due parti

(\Rightarrow) Si usa la dimostrazione della \approx^c sopra.

(\Leftarrow) Supposto $p + r \approx q + r \quad \forall r \in \mathcal{P}$. Presa un $a \in \text{Act}$ t.c. $a \notin fn(p) \cup fn(q)$ ² e assunto $p \xrightarrow{\mu} p'$. Allora $p + a \xrightarrow{\mu} p'$. Per $p + a \approx q + a$ si ha $q + a$ risposte alla transizione. Si prosegue guardando due casi:

1. $\mu = \tau$. Impossibile perché p' non può essere bisimile weak a $q + a$ e p' non può eseguire a .
2. $q + a \xrightarrow{\mu} q + a$. Vero, ma è proprio quello richiesto dalla rooted weak bisimulation. Se $p \xrightarrow{\mu} p'$ allora $q \xrightarrow{\mu} q'$ con $p' \approx q'$.

Vale anche il caso simmetrico in cui è q a muoversi per primo. ■

8. Assiomatizzazioni

Sono teorie equazionali che caratterizzano le congruenze. Visto che due LTS sono termini di un linguaggio, la loro congruenza può essere dimostrata sintatticamente mostrando che il primo può essere egualato al secondo per mezzo di una dimostrazione di deduzione equazionale.

Un'assiomatizzazione E che caratterizza una congruenza comportamentale R offre una prova alternativa puramente sintattica per dimostrare che i due processi sono congruenti secondo R . E può esistere solo per i sottocalcoli di CCS in cui R è decidibile.

Qui viene tenuto in considerazione solo CCS finite.

8.1. CCS finite “open”

In CCS Finite le costanti non possono essere usate ma si può estendere con termini aperti che considerano un insieme finito di azioni Act.

I termini aperti hanno occorrenze di variabili, mentre i termini chiusi sono quelli considerati processi “veri”.

$$p ::= \mathbf{0} \mid x \mid \mu.p \mid p + p \mid p|p \mid (\nu a)p$$

Un'assiomatizzazione per esso è dato da un insieme E di equazioni del tipo $t_1 = t_2$ dove t_1 e t_2 sono termini possibilmente aperti di CCS finite. A partire da queste equazioni è possibile derivare delle uguaglianze tra termini di CCS finite usando la deduzione equazionale.

8.2. Deduzione equazionale

$$1. \text{ (Riflessività)} \frac{}{t = t}$$

$$2. \text{ (Simmetria)} \frac{t_1 = t_2}{t_2 = t_1}$$

$$3. \text{ (Transitività)} \frac{t_1 = t_2 \quad t_2 = t_3}{t_1 = t_3}$$

4. Per ogni operatore f

$$\text{(Sostituzione)} \frac{\begin{array}{c} t_i = t'_i \\ f(t_1, \dots, t_i, \dots, t_k) = f(t_1, \dots, t'_i, \dots, t_k) \end{array}}{f(t_1, \dots, t_i, \dots, t_k) = f(t_1, \dots, t'_i, \dots, t_k)}$$

5. Per ogni sostituzione ρ

$$\text{(Istanziamento)} \frac{t_1 = t_2}{t_1[\rho] = t_2[\rho]}$$

6. Per ogni assioma $t_1 = t_2$ in E

$$\text{(Assiomi)} \frac{}{t_1 = t_2}$$

Una prova è una sequenza finita di uguaglianze

$$t_1 = t'_1, \dots, t_k = t'_k$$

t.c. ogni $t_j = t'_j$ o è una assioma (regole 1 e 6) oppure ottenuta usando una regola (regole 2, 3, 4, 5) con premesse delle uguaglianze precedenti nella sequenza.

Con $E \vdash t_1 = t_2$ si indica l'esistenza di una prova usando gli assiomi di E che termina con uguaglianza $t_1 = t_2$. La congruenza è $t_1 =_E t_2 \iff E \vdash t_1 = t_2$

8.3. Operatore +

- Associatività

$$x + (y + z) = (x + y) + z$$

- Commutatività

$$x + y = y + x$$

- Identità

$$x + \mathbf{0} = x$$

- Idempotenza

$$x + x = x$$

Esempio

$$\{A_1, A_2, A_4\} \vdash a.\mathbf{0} + (b.w + a.\mathbf{0}) = a.\mathbf{0} + b.w$$

$$A_1 \stackrel{\text{def}}{=} x + (y + z) = (x + y) + z$$

$$A_2 \stackrel{\text{def}}{=} x + y = y + x$$

$$A_4 \stackrel{\text{def}}{=} x + x = x$$

$$\begin{array}{c} \dfrac{x + y = y + x}{b.w + a.\mathbf{0} = a.\mathbf{0} + b.w} \\ \hline a.\mathbf{0} + (b.w + a.\mathbf{0}) = (a.\mathbf{0}) + (a.\mathbf{0} + b.w) \end{array} \quad \begin{array}{c} \dfrac{x + (y + z) = (x + y) + z}{a.\mathbf{0} + (a.\mathbf{0} + b.w) =} \\ \hline (a.\mathbf{0} + a.\mathbf{0}) + b.w \end{array} \quad \begin{array}{c} \dfrac{x + x = x}{a.\mathbf{0} + a.\mathbf{0} = a.\mathbf{0}} \\ \hline (a.\mathbf{0} + a.\mathbf{0}) + b.w = \end{array}$$

$$\dfrac{a.\mathbf{0} + (b.w + a.\mathbf{0}) = (a.\mathbf{0} + a.\mathbf{0}) + b.w}{a.\mathbf{0} + (b.w + a.\mathbf{0}) = a.\mathbf{0} + b.w} \quad \dfrac{a.\mathbf{0} + a.\mathbf{0} = a.\mathbf{0}}{a.\mathbf{0} + b.w}$$

8.4. Assiomatizzazioni (ground) sound e (ground) complete

Presi una relazione su processi CCS finite (closed) R .

L'assiomatizzazione E è detta (ground) sound per R se $E \vdash t_1 = t_2 \Rightarrow (t_1, t_2) \in R$

L'assiomatizzazione E è detta (ground) complete per R se $(t_1, t_2) \in R \Rightarrow E \vdash t_1 = t_2$

Quindi, E è sound e complete per $R \iff R$ è una congruenza e coincide con $=_E$ su termini chiusi.

8.5. Assiomatizzazione SB (non finita) di \sim per CCS finite

Si assumono altre regole, oltre agli assiomi visti sopra, ovvero quattro schemi di assioma (uno diverso per ogni scelta delle azioni a e μ che sono finite).

R1. $(\nu a)\mathbf{0} = \mathbf{0}$

R2. $(\nu a)\mu.x = \mu.(\nu a)x$ se $\mu \notin \{a, \bar{a}\}$

R3. $(\nu a)\mu.x = \mathbf{0}$ se $\mu \in \{a, \bar{a}\}$

R4. $(\nu a)(x + y) = (\nu a)x + (\nu a)y$

e uno non finito per ogni n e m .

$$\text{EXP. } x|y = \sum_i \mu_i.(x_i|y) + \sum_j \mu'_j.(x|y_j) + \sum_{i,j: \mu_i = \mu'_j} \tau.(x_i|y_j) \quad \text{se } x = \sum_{i=1}^n \mu_i.x_i \wedge y = \sum_{j=1}^m \mu'_j.y_j$$

Teorema (SB è sound) $\forall p, q$ in CCS finite (closed), si ha $\mathcal{SB} \vdash p = q \Rightarrow p \sim q$

Dimostrazione Per induzione su prova finita di $\mathcal{SB} \vdash p = q$. Gli assiomi usati sono quello sulla riflessività (ovvia perché vale per \sim) e quelli su SB (valgono per le proprietà algebriche). Le altre regole valgono assumendo che la tesi vale sulla premessa, perché \sim è un'equivalenza ed anche congruenza. ■

Teorema (SB è completo) Se due processi p e q sono in forma normale t.c. $p \sim q$ allora $\mathcal{SB} \vdash p = q$

Dimostrazione Un processo p è in forma normale se costruito solo con $\mathbf{0}$, prefisso e somma e rappresentato con una sommatoria: dunque del tipo $\sum_{i \in I} \mu_i.p_i$. Una misura di una forma normale è il massimo numero di prefissi annidati (assumendo che $\max(\emptyset) = 0$) si definisce

$$\text{depth}\left(\sum_{i \in I} \mu_i.p_i\right) = \max\{\text{depth}(\mu_i.p_i) \mid i \in I\}$$

$$\text{depth}(\mu.p) = 1 + \text{depth}(p)$$

Per ogni forma normale p , se $p \xrightarrow{\mu} p'$ allora $\mu.p'$ è una somma di p , p' è una forma normale e $\text{depth}(p') < \text{depth}(p)$.

La dimostrazione viene fatta per induzioni sulla somma delle profondità di p e q . Se la somma è 0 allora $p = q = 0$ e la tesi segue la regola di riflessività. Altrimenti, supposto $\mu.p'$ come somma di p , allora $p \xrightarrow{\mu} p'$. Per $p \sim q$ allora anche $q \xrightarrow{\mu} q'$ anche $p' \sim q'$. Visto che q è una forma normale, $\mu.q'$ deve essere una somma di q .

La somma delle profondità di p' e q' è strettamente decrescente, visto che l'induzione può essere applicata in ordine per ottenere $\mathcal{SB} \vdash p' = q'$. Per la regola di sostituzione allora anche $\mathcal{SB} \vdash \mu.p' = \mu.q'$ è derivabile.

Allora, per ogni somma $\mu.p'$ di p abbiamo la somma $\mu.q'$ di q così che due siano equiparate dagli assiomi.

Simmetricamente, possiamo provare che per ogni somma $\mu.q'$ di q , esiste un $\mu.p'$ di p t.c. $\mathcal{SB} \vdash \mu.p' = \mu.q'$ è derivabile.

Allora, mettendo tutte le somme insieme si ha $\mathcal{SB} \vdash p = q$ modulo l'assioma di idempotenza (rimuove duplicati) e associatività/commutatività (sistema le somme). ■

Teorema Per ogni processo CCS finite p esiste una forma normale q t.c. $\mathcal{SB} \vdash p = q$.

Dimostrazione Per induzione strutturale su p usando

1. Se p e q sono forme normali, allora $\exists r$ forma normale t.c. $\mathcal{SB} \vdash p|q = r$.
2. Se p è una forma normale, allora $\exists r$ forma normale t.c. $\mathcal{SB} \vdash (\nu a)p = r$.

TODO: guardare dal libro meglio

Teorema $p \sim q \Rightarrow \mathcal{SB} \vdash p = q$

Dimostrazione Per il lemma sopra $\exists s, t$ forme normali t.c. $\mathcal{SB} \vdash p = s \wedge \mathcal{SB} \vdash q = t$. Per il teorema di soundness vale anche $p \sim s \wedge q \sim t$. Dato che $p \sim q$ per transitività si anche $s \sim t$. Per il teorema di completezza delle forme normali si ha $\mathcal{SB} \vdash s = t$. Quindi la tesi $\mathcal{SB} \vdash p = q$ segue per transitività.

8.6. Assiomatizzazione per simulation/trace equivalence

Se a SB si aggiunge $S \stackrel{\text{def}}{=} \mu.(x + y) = \mu.(x + y) + \mu.y$ si ottiene un'assiomatizzazione sound e completa di simulation equivalence.

Se a SB si aggiunge $T \stackrel{\text{def}}{=} \mu.(x + y) = \mu.x + \mu.y$ si ottiene un'assiomatizzazione sound e completa di trace equivalence.

8.7. Assiomatizzazione finita per trace equivalence

L'assiomatizzazione sopra non è finita perché usa lo schema infinitario di assioma EXP. Al suo posto si usano

P1. $0|x = x$

P2. $x|y = y|x$

P3. $(x + y)|z = x|z + y|z$

P4. $\mu.x|\mu'.y = \mu.(x|\mu'.y) + \mu' .(\mu.x|y)$ se $\mu' \neq \bar{\mu}$

P5. $\alpha.x|\bar{a}.y = \alpha.(x|\bar{a}.y) + \bar{a}.(\alpha.x|y) + \tau.(x|y)$

Il P3 non è sound per bisimulatione e neanche per simulazione.

8.8. Assiomatizzazione WB per \approx^c

Prese le seguenti leggi

$W_1.$ $\mu.\tau.x = \mu.x$

$W_2.$ $x + \tau.x = \tau.x$

$W_3.$ $\mu.(x + \tau.y) = \mu.(x + \tau.y) + \mu.y$

Preso $\mathcal{WB} = \mathcal{SB} \cup \{W_1, W_2, W_3\}$ si ha il **Teorema di soundness** in cui per p, q processi CCS finite (closed), $\mathcal{WB} \vdash p = q \Rightarrow p \approx^c q$.

Dimostrazione Per induzione sulla prova di $\mathcal{WB} \vdash p = q$. La soundness degli assiomi W_1, W_2, W_3 è una proprietà algebrica. ■

La **forma normale saturata** è una normale $p = \sum_i \mu_i.p_i$ se ogni volta che $p \xrightarrow{\mu} p'$ allora $\mu.p'$ è un addendo di p (e p' è a sua volta saturato). Ogni forma normale può essere saturata. Ogni processo può essere ridotto in forma saturata. Le forme saturate hanno completezza.

8.8.1. Saturation Lemma

Data una forma normale p , se $p \xrightarrow{\mu} p'$ allora $\mathcal{WB} \vdash p = p + \mu.p'$

Dimostrazione Per induzione sulla lunghezza di $p \xrightarrow{\epsilon} \xrightarrow{\mu} \xrightarrow{\epsilon} p'$. Il caso base è $p \xrightarrow{\mu} p'$, ovvero $\mu.p'$ è una somma di p . Poi per idempotenza la tesi regge.

In modo induttivo si hanno due casi

1. $p \xrightarrow{\mu} q \xrightarrow{\tau} p'$

Qui $\mu.q$ è una somma di p . Sappiamo, per induzione, che $\mathcal{WB} \vdash q = q + \tau.p'$. Si conclude che

- Per l'assioma di idempotenza $\mathcal{WB} \vdash p = p + \mu.q$
- Per induzione e sostituzione $\mathcal{WB} \vdash p = p + \mu.(q + \tau.p')$
- Per l'assioma W_3 $\mathcal{WB} \vdash p = p + \mu.(q + \tau.p') + \mu.p'$
- Per i passi precedenti $\mathcal{WB} \vdash p = p + \mu.p'$

$$2. \quad p \xrightarrow{\tau} q \xrightarrow{\mu} p'$$

Qui $\tau.q$ è una somma di p . Sappiamo, per induzione, che $\mathcal{WB} \vdash q = q + \mu.p'$. Si conclude che

- Per l'assioma di idempotenza $\mathcal{WB} \vdash p = p + \tau.q$
- Per induzione e sostituzione $\mathcal{WB} \vdash p = p + \tau.q + q + \mu.p'$
- Per l'assioma W_2 $\mathcal{WB} \vdash p = p + \tau.q + q$
- Per i passi precedenti $\mathcal{WB} \vdash p = p + \mu.p'$

■

Proposizione (Saturazione di forme normali) Per ogni forma normale p esiste una forma normale saturata q della stessa profondità di $\mathcal{WB} \vdash p = q$.

Dimostrazione Per induzione sulla profondità di p . Se $= 0$, allora $p = 0$ ed è una forma normale saturata. Altrimenti, si assume che per ogni somma $\mu_i.p_i$ di p , $\mathcal{WB} \vdash p_i = q_i$ dove q_i è una forma saturata t.c. $\text{depth}(p_i) = \text{depth}(q_i)$. Per sostituzione si ha $\mathcal{WB} \vdash \mu_i.p_i = \mu_i = q_i$.

Preso $q' = \sum_i \mu_i.q_i$ si va a $\mathcal{WB} \vdash p = q'$. Il processo q' è una forma normale di uguale profondità ma non saturata per qualche i ; μ_i può essere τ .

Preso l'insieme $I = \left\{ (\mu'_k, p'_k) \mid q' \xrightarrow{\mu'_k} p'_k \text{ ma non } q' \xrightarrow{\mu'_k} p'_k \right\}$. per il Saturation Lemma, se $|I| = m$, $\mathcal{WB} \vdash q' = q' + \mu'_1.p'_1 + \dots + \mu'_m.p'_m$ che è una forma saturata. La forma normale saturata ha la stessa profondità di q' perché ogni somma di $\mu'_k.p'_k$ è la profondità più piccola.

Quindi ogni processo può essere ridotto in forma normale e poi in forma normale saturata. ■

Proposizione (Completezza per forme normali saturate) Se p e q sono forme normali saturate t.c. $p \approx^c q$ allora $\mathcal{WB} \vdash p = q$

Dimostrazione Per induzione sulla somma delle profondità di p e q . Se la somma è 0, allora si ha $\mu_p = q = 0$ e la tesi sulla riflessività vale. Altrimenti, supposto $\mu.p'$ come somma di p , allora $p \xrightarrow{\mu} p'$. Per $p \approx^c q$ si ha $q \Rightarrow q'$ con $p' \approx^c q'$. Dato che q è in forma naturale saturata, allora si ha $q \rightarrow q'$. Sommando, per ogni somma $\mu.p'$ di p si ha una somma $\mu.q'$ di q t.c. $p' \approx^c q'$. Per il lemma di Hennessy si ha

$$p' \approx q' \iff (p' \approx^c q' \vee p' \approx^c \tau.q' \vee \tau.p' \approx^c q')$$

Si hanno tre casi:

1. Se $p' \approx^c q'$, allora p' e q' sono forme normali saturate, per induzione la somma delle profondità è strettamente decrescente. $\mathcal{WB} \vdash p' = q'$ quindi $\mathcal{WB} \vdash \mu.p' = \mu.q'$ per sostituzione.
2. Se $p' \approx^c \tau.q'$, bisogna ridurre $\tau.q'$ ad una forma normale saturata. Sappiamo che esiste una forma normale saturata q'' uguale alla profondità t.c. $\mathcal{WB} \vdash \tau.q' = q''$. Per il teorema della soundness abbiamo $\tau.q' \approx^c q''$ e dunque $p' \approx^c q''$ per transitività. Dato che $p' \approx^c q''$ e la somma delle profondità di p' e q' è una in meno di p e q , allora si può applicare per induzione e derivare $\mathcal{WB} \vdash p' = q''$. Quindi $\mathcal{WB} \vdash p' = \tau.q'$ per transitività, $\mathcal{WB} \vdash \mu.p' = \mu.\tau.q'$ per sostituzione e $\mathcal{WB} \vdash \mu.p' = \mu.q'$ per W1.

3. Se $\tau.p' \approx^c q'$ si procede nel medesimo modo visto sopra.

Per tutti i casi, per ogni somma $\mu.p'$ di p , si ha una somma $\mu.q'$ di q t.c. $\mathcal{WB} \vdash \mu.p' = \mu.q'$. Simmetricamente si prova per ogni somma $\mu.q'$ di q t.c. $\mathcal{WB} \vdash \mu.q' = \mu.p'$. Inoltre, $\mathcal{WB} \vdash p = q$ per sostituzione e possibili applicazioni degli assiomi di idempotenza (per rimuovere duplicati) e associatività/commutatività (riarrangiare le somme). ■

Teorema $p \approx^c q \Rightarrow \mathcal{WB} \vdash p = q$

Dimostrazione Per il lemma di riduzione a forma normale saturata, esistono forme normali saturate s e t t.c. $\mathcal{WB} \vdash p = s$ e $\mathcal{WB} \vdash q = t$. Per teorema di soundness si ha $p \approx^c s$ e $q \approx^c t$. Dato che $p \approx^c q$ per transitività anche $s \approx^c t$. Per teorema di completezza delle forme normali saturate si ha $\mathcal{WB} \vdash s = t$. Quindi la tesi $\mathcal{WB} \vdash p = q$ segue per transitività. ■

8.9. Assiomatizzazione finita di \sim via operatori ausiliari di CCS finite

- Left merge

$$(\text{Left}) \frac{p \xrightarrow{\mu} p'}{p|q \xrightarrow{\mu} p'|q}$$

- Sync merge

$$(\text{Merge}) \frac{p \xrightarrow{\alpha} p' \quad q \xrightarrow{\bar{\alpha}} q'}{p\|q \xrightarrow{\tau} p'|q'}$$

\sim è una congruenza per Left e Sync merge.

Al posto dell'assioma EXP si hanno

Par. $x|y = x|y + y|x + x\|y$

L1. $0|y = 0$

L2. $(\mu.x)|y = \mu.(x|y)$

L3. $(x + y)|z = x|z + y|z$

C1. $x\|y = y\|x$

C2. $0\|y = 0$

C3. $(\mu_1.x)\|(\mu_2.y) = \tau.(x|y)$ se $\mu_1 = \bar{\mu_2}$

C4. $(\mu_1.x)\|(\mu_2.y) = 0$ se $\mu_1 \neq \bar{\mu_2}$

C5. $(x + y)\|z = x\|z + y\|z$

ASB è sound. $\mathcal{ASB} \vdash p = q \Rightarrow p \sim q$

Per ogni processo p (che usa anche gli operatori ausiliari), esiste una forma normale q t.c.

$\mathcal{ASB} \vdash p = q$

Inoltre è completa per forma normale.

8.10. Assiomatizzazioni finite per rooted weak bisimilarity

TODO: prendere dal libro

9. Teoria dei punti fissi

Per trovare l'equivalenza di bisimulazione \sim bisogna usare un algoritmo. Essa è vista come massimo punto fisso di una funzione tra relazione, ovvero l'equivalenza più grande.

I punti fissi sono anche usati nelle logiche con formule ricorsive.

Un **poset** (insieme parzialmente ordinato) è una coppia (D, \leq) in cui D è un insieme e $\leq \subseteq D \times D$ è una relazione

- riflessiva $d \leq d \quad \forall d \in D$
- antisimmetrica $d \leq e \wedge e \leq d \Rightarrow d = e \quad \forall d, e \in D$
- transitiva $d \leq e \wedge e \leq f \Rightarrow d \leq f \quad \forall d, e, f \in D$

È totalmente ordinato se $d \leq e \vee e \leq d \quad \forall d, e \in D$.

Esempio

Un esempio totalmente ordinato è l'insieme dei numeri naturali (N, \leq) .

Un esempio non totalmente ordinato è l'insieme delle parti $(2^S, \subseteq)$. L'insieme minimo è \emptyset , quello max è S .

Altri esempi di poset sono (A^*, \leq) ovvero insieme delle parole sull'alfabeto A con prefix ordering $s \leq t \iff \exists u \in A^*.su = t$ e (F, \leq) con $f : S \mapsto D$ con (D, \leq') poset con $f \leq g \iff f(s) \leq' g(s) \quad \forall s \in S$ (grafico di F sta sempre sotto quello di G).

9.1. Estremi

L'estremo superiore = minimo dei maggioranti = sup.

L'estremo inferiore = massimo dei minoranti = inf.

Preso un poset (D, \leq) e $X \subseteq D$ si ha

- $d \in D$ è maggiorante (o minorante) per $X \iff x \leq d$ (o $d \leq x$) $\forall x \in X$.
 - d è sup (o inf) per X ed indicato con $\bigcup X$ (o $\bigcap X$) \iff
1. d è maggiorante (o minorante) di X .
 2. $d \leq d'$ (o $d' \leq d$) $\forall d' \in D$ che è maggiorante (o minorante) per X .

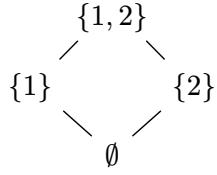
Se il sup (o inf) d di X è $d \in X$, allora d è $\max X$ (o $\min X$).

Esempio

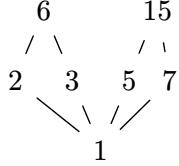
Per (N, \leq) tutti i sotto insiemi finiti X di N hanno un sup; nessuno sotto insieme infinito ha sup. Tutti i sotto insiemi X di N hanno inf. ∞ non è un numero, quindi non vale.

Per $(2^S, \subseteq)$ ha $\bigcup X = \sup$ e $\bigcap X = \inf$.

Preso $2^{\{1,2\}}$ si ha $X = \{\{1\}, \{2\}\}$, $\bigcup X = \{1, 2\}$, $\bigcap X = \emptyset$.

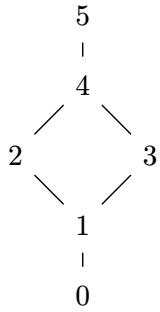


Preso $X = \{3, 5\}$ si ha



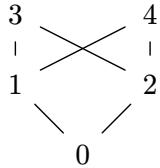
si ha $\bigcup X = 1$ (che poi è il m.c.m.) e $\bigcap X = 15$ (che poi è M.C.D.).

Preso $X = \{2, 3\}$



si ha $\bigcup X = \sup X = 4$ e $\bigcap X = \inf X = 1$.

Presi $X = \{1, 2\}$ e $Y = \{1, 2, 3\}$



Si hanno maggioranti di $X = \{3, 4\}$, $\emptyset \bigcup X$. I minoranti di $X = \{0\}$, $\bigcap X = 0$.

Si hanno maggioranti di $Y = \{3\}$ ma $3 \in Y$ quindi $\bigcup Y = \max Y = 3$. I minoranti di $Y = \{0\}$, $\bigcap Y = 0$. Inoltre questo è un esempio di reticolo.

9.2. Reticolo

Un poset (D, \leq) è un reticolo $\iff \forall d, e \in D \exists \sup \bigcup \{d, e\} \wedge \inf \bigcap \{d, e\}$. Insomma, ha sempre min e max. L'insieme delle parti è un buon esempio di reticolo.

Un poset (D, \leq) è un reticolo completo $\iff \sup \bigcup X \wedge \inf \bigcap X$ esistono per ogni $X \subseteq D$. Vi sono reticoli non completi. In genere hanno struttura a rombo.

Un reticolo completo ha un min $\perp = \bigcap D$ e un max $\top = \bigcup D$. Inoltre, $\bigcup \emptyset = \perp$, $\bigcap \emptyset = \top$.

Esempio

(N, \leq) è un reticolo non completo perché non ha sup per i suoi sotto insiemi infiniti ($\infty \notin N$).

$(N \cup \{\infty\}, \leq')$ e $(2^S, \subseteq)$ sono reticolli completi.

9.3. Monotonia

Preso (D, \leq) , una funzione $f : D \mapsto D$ è mononota $\iff d \leq d' \Rightarrow f(d) \leq f(d') \quad \forall d, d' \in D$.

Un elemento d è detto punto fisso $\iff d = f(d)$.

- Post punto fisso $\iff d \leq f(d)$
- Pre punto fisso $\iff f(d) \leq d$

Esempio

$f : 2^{\mathbb{N}} \mapsto 2^{\mathbb{N}}$ in cui $f(X) = X \cup \{1, 2\}$ si ha f monotona e tutti gli infiniti punti fissi hanno la forma $\{Y \mid \{1, 2\} \subseteq Y\}$ con $\{1, 2\}$ con min punto fisso e \mathbb{N} come max punto fisso.. Infatti,

$$x \leq y \Rightarrow f(x) \leq f(y) \Rightarrow X \cup \{1, 2\} \leq Y$$

Ogni X che ha $\{1, 2\}$ è punto fisso per quella funzione.

$$f(\{1, 2, 3\}) = \{1, 2, 3\} \cup \{1, 2\} = \{1, 2, 3\}$$

Invece, preso $X = \{2\}$ con un $g(X) = \{1, 2, 3\}$ si vede come $g(X) = X \cup \{1, 2\}$, se $X \neq \{2\}$ la g non è monotona perché $\{1, 2, 3\} \notin \{1, 2\}$.

9.4. Potenza di funzione

Preso $f : D \mapsto D$ su un insieme D . Per ogni $n \in \mathbb{N}$, si definisce $f^n(d)$ per ogni $d \in D$ come:

- $f^0(d) = d$
- $f^{n+1}(d) = f(f^n(d))$

9.5. Teorema di Knaster-Tarski

Preso il reticolo completo (D, \leq) e la funzione monotona $f : D \mapsto D$. f ha:

- max punto fisso $Z_{\max} = \bigcup\{x \in D \mid x \leq f(x)\}$ (sup dei post punti fissi).
- min punto fisso $Z_{\min} = \bigcap\{x \in D \mid f(x) \leq x\}$ (inf dei pre punti fissi).

Dimostrazione Si dimostra sia che esistono il max e min punti fissi.

Preso $A = \{x \in D \mid x \leq f(x)\}$ l'insieme dei post punti fissi con $Z_{\max} = \bigcup A$.

1. Z_{\max} è punto fisso $Z_{\max} = f(Z_{\max})$ e lo si dimostra per antisimmetria:

- $Z_{\max} \leq f(Z_{\max})$

Dalla definizione si sa che $Z_{\max} = \bigcup A$, quindi $\forall x \in A$, vale $x \leq Z_{\max}$. Data la monotonia di f , $x \leq Z_{\max} \Rightarrow f(x) \leq f(Z_{\max})$.

Quindi $\forall x \in A$, $x \leq f(x) \leq f(Z_{\max})$, ovvero, $f(Z_{\max})$ è un maggiorante per A . Sempre da definizione, Z_{\max} è minimo maggiorante di A , dunque $Z_{\max} \leq f(Z_{\max})$.

- $f(Z_{\max}) \leq Z_{\max}$

Per monotonia di f si ha $f(Z_{\max}) \leq f(f(Z_{\max}))$, ovvero $f(Z_{\max})$ è post punto fisso e quindi $f(Z_{\max}) \in A$. Poiché Z_{\max} è un maggiorante di A , deve essere $f(Z_{\max}) \leq Z_{\max}$.

Quindi $Z_{\max} = f(Z_{\max})$.

2. Z_{\max} è il massimo punto fisso, ovvero se $f(d) = d$, allora $d \leq Z_{\max}$. Preso d come un qualunque punto fisso, $d = f(d)$, allora vale anche che $d \leq f(d)$, ovvero $d \in A$ e dunque $d \leq \bigcup A = Z_{\max}$.

Preso $B = \{x \in D \mid f(x) \leq x\}$ l'insieme dei pre punti fissi con $Z_{\min} = \bigcap B$.

1. Z_{\min} è punto fisso $Z_{\min} = f(Z_{\min})$ e lo si dimostra per antisimmetria:

- $f(Z_{\min}) \leq Z_{\min}$

Dalla definizione si sa che $Z_{\min} = \bigcap B$, quindi $\forall x \in B$, vale $x \geq Z_{\min}$. Data la monotonia di f , $x \geq Z_{\min} \Rightarrow f(x) \geq f(Z_{\min})$.

Quindi $\forall x \in B$, $x \geq f(x) \geq f(Z_{\min})$, ovvero, $f(Z_{\min})$ è un minorante per B . Sempre da definizione, Z_{\min} è massimo minorante di B , dunque $Z_{\min} \geq f(Z_{\min})$.

- $Z_{\min} \leq f(Z_{\min})$

Per monotonia di f si ha $f(Z_{\min}) \geq f(f(Z_{\min}))$, ovvero $f(Z_{\min})$ è pre punto fisso e quindi $f(Z_{\min}) \in B$. Poiché Z_{\min} è un minorante di B , deve essere $f(Z_{\min}) \geq Z_{\min}$.

Quindi $Z_{\min} = f(Z_{\min})$.

2. Z_{\min} è il minimo punto fisso, ovvero se $f(d) = d$, allora $d \geq Z_{\min}$. Preso d come un qualunque punto fisso, $d = f(d)$, allora vale anche che $d \geq f(d)$, ovvero $d \in B$ e dunque $d \geq \bigcap B = Z_{\min}$.



Esempio

$(2^S, \subseteq)$ con $f : 2^S \mapsto 2^S$ monotona si ha

$$Z_{\max} = \bigcup \{X \in 2^S \mid X \subseteq f(X)\}$$

$$Z_{\min} = \bigcap \{X \in 2^S \mid X \supseteq f(X)\}$$

Nel caso d'esempio dell'altra pagina in cui si parla di $S = \mathbb{N}$ e $f(X) = X \cup \{1, 2\}$ allora si ha

$Z_{\max} = \bigcup \{X \subseteq \mathbb{N} \mid X \subseteq X \cup \{1, 2\}\} = \mathbb{N}$ e questo fa l'unione di tutti i inumeri, tanto è indifferente se contengono $\{1, 2\}$ perché aggiunte da f .

$Z_{\min} = \bigcap \{X \subseteq \mathbb{N} \mid X \cup \{1, 2\} \subseteq X\} = \{1, 2\}$ intersezione con quelli che già contengono $\{1, 2\}$.

9.6. Calcolo dei punti fissi

Sia (D, \leq) un reticolo completo finito e $f : D \mapsto D$ una funzione monotona. Allora si ha minimo punto fisso

- $Z_{\min} = f^m(\perp)$ per qualche $m \in \mathbb{N}$. Partendo dal sotto va a salire finché non trova il min punto fisso.
- $Z_{\max} = f^M(\top)$ per qualche $M \in \mathbb{N}$. Partendo dal sopra va a scendere finché non trova il max punto fisso.

Dimostrazione Per dimostrare che $Z_{\min} = f^m(\perp)$ si vede che, data la monotonia di f e \perp che è il minimo, allora abbiamo che $\perp \leq f(\perp) \leq f^2(\perp) \leq \dots$ ma, dato che D è finito, la catena deve avere una costante da un certo punto in poi, ovvero $\exists m. \forall k \geq m. f^k(\perp) = f^m(\perp)$.

Quindi $f(f^m(\perp)) = f^{m+1}(\perp) = f^m(\perp)$ e dunque $f^m(\perp)$ è punto fisso. Preso un punto fisso $d = f(d)$. Dato che $f^m(\perp)$ è il minimo punto fisso, vale che $\perp \leq d$ e per monotonia, anche $f(\perp) \leq f(d) = d$ ma in generale $f^k(\perp) \leq f^k(d) = d \quad \forall k \in \mathbb{N}$. In conclusione, $f^m(\perp) \leq d$.

Per dimostare che $Z_{\max} = f^M(\top)$ si vede che, data la monotonia di f e \top che è il massimo, allora abbiamo che $\top \geq f(\top) \geq f^2(\top) \geq \dots$ ma, dato che D è finito, la catena deve avere una costante da un certo punto in poi, ovvero $\exists M. \forall k \geq M. f^k(\top) = f^M(\top)$.

Quindi $f(f^M(\top)) = f^{M+1}(\top) = f^M(\top)$ e dunque $f^M(\top)$ è punto fisso. Preso un punto fisso $d = f(d)$. Dato che $f^M(\top)$ è massimo punto fisso, vale che $d \leq \top$ e per monotonia, anche $d = f(d) \leq f(\top)$ ma in generale $d = f^k(d) \leq f^k(\top) \quad \forall k \in \mathbb{N}$. In conclusione $f^M(\top) \geq d$. ■

Esempio

$g : 2^{\{0,1,2\}} \mapsto 2^{\{0,1,2\}}$ con $g(X) = (X \cap \{1\}) \cup \{2\}$ si ha g monotona e $2^{\{0,1,2\}}$ reticolo completo (sono 8 elementi). Si ha

$$g(\emptyset) = \{2\} \quad g^2(\emptyset) = g(\{2\}) = \{2\} \quad Z_{\min} = \{2\}$$

$$g(\{0, 1, 2\}) = \{1, 2\} \quad g^2(\{0, 1, 2\}) = g(\{1, 2\}) = \{1, 2\} \quad Z_{\max} = \{1, 2\}$$

9.7. Convergenza e divergenza

Dato un LTS $TS = (Q, A, \rightarrow)$.

Uno stato q è detto convergente se può raggiungere uno stato di deadlock. La convergenza è calcolata come minimo punto fisso.

Uno stato q è detto divergente se può eseguire una computazione infinita. La divergenza è calcolata come massimo punto fisso.

Uno stato deve essere almeno uno fra i due.

Preso il reticolo completo con Q finito, 2^Q .

Per vedere la **convergenza** si prende la funzione $C : 2^Q \mapsto 2^Q$ monotona.

$$C(X) = \left\{ q \mid \exists q'. q \xrightarrow{\mu} q', q' \in X \right\} \cup \{q \mid q \text{ è deadlock}\}$$

$C^1(\emptyset) = \{q \mid q \text{ è deadlock}\}$ ovvero stati che con al più zero transizioni raggiungono un deadlock.

$C^2(\emptyset) = C(\{q \mid q \text{ è deadlock}\})$ ovvero stati che possono fare una transizione e poi andare in deadlock oppure che lo sono già.

$C^k(\emptyset)$ stati che eseguono al più $k - 1$ transizioni e vanno in deadlock.

Per vedere la **divergenza** si prende la funzione $D : 2^D \mapsto 2^D$ monotona.

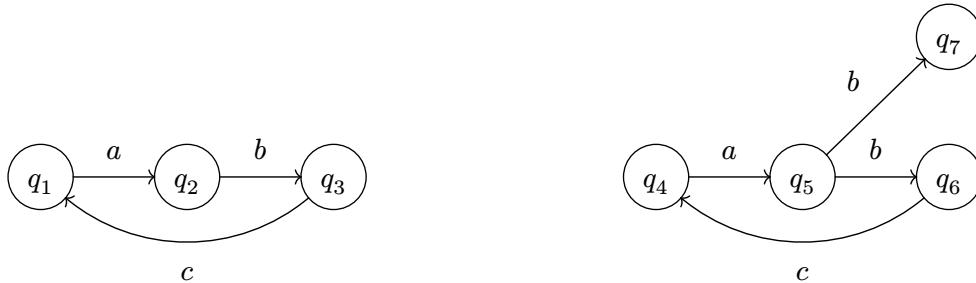
$$D(X) = \left\{ q \mid \exists q'. q \xrightarrow{\mu} q', q' \in X \right\}$$

$D^1(Q) = \left\{ q \mid \exists q'. q \xrightarrow{\mu} q', q' \in Q \right\}$ ovvero stati che possono fare almeno una transizione.

$D^2(Q) = D(D^1(Q))$ ovvero stati che possono fare una transizione ed arrivare su $D^1(Q)$ e quindi fare 2 transizioni in totale.

$D^k(Q)$ ovvero stati che possono fare almeno k transizioni.

Esempio



$$C^1(\emptyset) = \{q_7\}$$

$$C^2(\emptyset) = \{q_7, q_5\}$$

$$C^3(\emptyset) = \{q_7, q_5, q_4\}$$

$$C^4(\emptyset) = \{q_7, q_5, q_4, q_6\}$$

$$C^4(\emptyset) = C^5(\emptyset)$$

$$D^1(Q) = Q \setminus \{q_7\} = \{q_1, q_2, q_3, q_4, q_5, q_6\}$$

$$D^2(Q) = D^1(Q)$$

9.7.1. Sempre convergenza e sempre divergenza

Uno stato è sempre convergente $SC(X)$ se tutte le computazioni terminano. È l'insieme Z_{\min} di SC il complemento dell'insieme Z_{\max} di D .

Uno stato è sempre divergente $SD(X)$ se tutte le computazioni non raggiungono mai un deadlock. È l'insieme Z_{\max} di SD il complemento dell'insieme Z_{\min} di C .

$$\overline{Z_{\max}(D)} = Z_{\min}(SC) \quad \overline{Z_{\min}(SC)} = Z_{\max}(D)$$

Dimostrazione Presa una funzione monotona $f : 2^S \mapsto 2^S$ si ha $Z_{\max} = \bigcup\{X \subseteq S \mid X \subseteq f(X)\}$ e

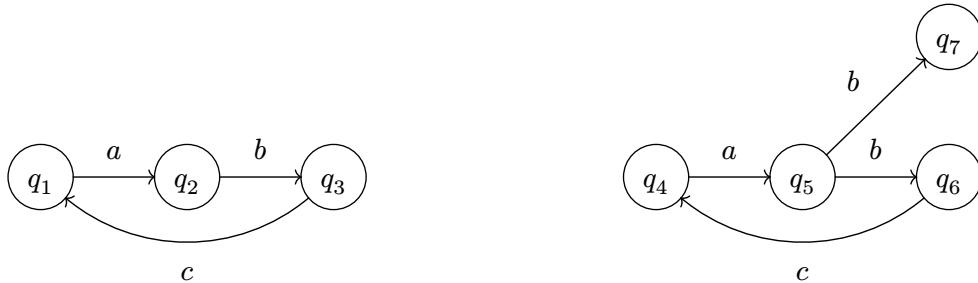
$$\begin{aligned} \overline{Z_{\max}} &= \overline{\bigcup\{X \mid X \subseteq f(X)\}} = \\ &= \bigcap\{\overline{X} \mid X \subseteq f(X)\} = \bigcap\{Y \mid \overline{Y} \subseteq f(\overline{Y})\} = \bigcap\{Y \mid \overline{f(\overline{Y})} \subseteq Y\} = \\ &= \bigcap\{Y \mid f_d(Y) \subseteq Y\} \end{aligned}$$

qui $f_d(Y) = \overline{f(\overline{Y})}$ che è monotona

$$X \subseteq Y \Rightarrow \overline{Y} \subseteq \overline{X} \Rightarrow f(\overline{Y}) \subseteq f(\overline{X}) \Rightarrow \overline{f(\overline{X})} \subseteq \overline{f(\overline{Y})} \Rightarrow f_d(X) \subseteq f_d(Y)$$

■

Esempio



Ricordando che

$$C^4(\emptyset) = \{q_7, q_5, q_4, q_6\}$$

$$C^4(\emptyset) = C^5(\emptyset)$$

$$D^1(Q) = Q \setminus \{q_7\} = \{q_1, q_2, q_3, q_4, q_5, q_6\}$$

$$D^2(Q) = D^1(Q)$$

si può trovare

$$SC^1(\emptyset) = \{q_7\}$$

$$SC^2(\emptyset) = SC^1(\emptyset)$$

$$SD^1(Q) = \{q_1, q_2, q_3, q_4, q_5, q_6\}$$

$$SD^2(Q) = \{q_1, q_2, q_3, q_4, q_6\}$$

$$SD^3(Q) = \{q_1, q_2, q_3, q_6\}$$

$$SD^4(Q) = \{q_1, q_2, q_3\}$$

$$SD^5(Q) = SD^4(Q)$$

Esempio

Si ha

$$C(X) = \left\{ q \mid \exists q'. q \xrightarrow{\mu} q', q' \in X \right\} \cup \{q \mid q \text{ è deadlock}\}$$

si può avere

$$\begin{aligned} \overline{C(\overline{X})} &= \left\{ q \mid \nexists q'. q \xrightarrow{\mu} q', q' \in X \right\} \cap \{q \mid q \text{ non è deadlock}\} = \\ &= \left\{ q \mid \exists q'. q \xrightarrow{\mu} q' \wedge \forall q'. q \xrightarrow{\mu} q', q' \in X \right\} = SD(X) \end{aligned}$$

9.8. Bisimulazione come punto fisso

Si può riformulare \sim come massimo punto fisso di una funzione F che trasforma relazioni binarie R su stati. L'insieme di tutte le relazioni binarie su Q , definito come $2^{Q \times Q}$ e finito se Q finito, è un reticolo completo con $\top = Q \times Q$.

La funzione F è un funzionale definito come $F : \mathcal{P}(Q \times Q) \mapsto \mathcal{P}(Q \times Q)$. Se $R \subseteq Q \times Q$ allora $(q_1, q_2) \in F(R) \iff \forall \mu \in A$

- $\forall q'_1. q_1 \xrightarrow{\mu} q'_1, \exists q'_2. q_2 \xrightarrow{\mu} q'_2 \wedge (q'_1, q'_2) \in R$
- $\forall q'_2. q_2 \xrightarrow{\mu} q'_2, \exists q'_1. q_1 \xrightarrow{\mu} q'_1 \wedge (q'_1, q'_2) \in R$

La differenza della definizione con quella della bisimulazione è che c'è un *iff* e quella coppia, inoltre, sta nell'insieme da cui si è partiti.

Per ogni $TS = (Q, A, \rightarrow)$ si ha

1. F è monotono. (eg. $R_1 \subseteq R_2 \Rightarrow F(R_1) \subseteq F(R_2)$).
2. $R \subseteq Q \times Q$ è bisimulazione $\iff R \subseteq F(R)$.

Dimostrazione Il fatto che F sia monotono, vuol dire che $(q_1, q_2) \in F(R_1) \Rightarrow \forall \mu \in A \dots$ quanto scritto sopra. Il fatto che $R_1 \subseteq R_2 \Rightarrow \forall \mu \in A \dots$ quanto scritto per quello prima. Dunque $(q_1, q_2) \in F(R_2)$.

Il punto 2. dice che se R è bisimulazione, allora $(q_1, q_2) \in R \Rightarrow \forall \mu \in A$

- $\forall q'_1. q_1 \xrightarrow{\mu} q'_1, \exists q'_2. q_2 \xrightarrow{\mu} q'_2 \wedge (q'_1, q'_2) \in R$

- $\forall q'_2.q_2 \xrightarrow{\mu} q'_2, \exists q'_1.q_1 \xrightarrow{\mu} q'_1 \wedge (q'_1, q'_2) \in R$

significa che $(q_1, q_2) \in F(R)$ e se $R \subseteq F(R)$ allora $F(R)$ vale $\forall r \in R$ quindi R è bisimulazione.

■

Per Knaster-Tarski abbiamo che $Z_{\max} = \bigcup\{R \mid R \subseteq F(R)\}$. Poiché R è bisimulazione $\iff R \subseteq F(R)$ allora $Z_{\max} = \bigcup\{R \mid R \text{ è una bisimulazione}\}$ ovvero $Z_{\max} = \sim$.

Un algoritmo per calcolare Z_{\max} di F con Q finito è

$$x := Q \times Q$$

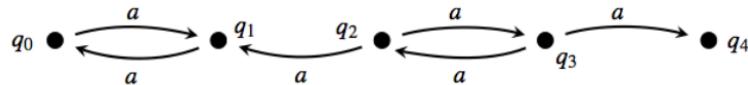
$$y := F(x)$$

while $x \neq y$ do $\{x := y; y := F(y)\}$

return x

L'algoritmo termina sempre perché Q è finito.

Esempio



$$Q = \{q_0, q_1, q_2, q_3, q_4\}$$

$$F^0(Q \times Q) = Q \times Q$$

$$F^1(Q \times Q) = \{(q_0, q_1), (q_0, q_2), (q_0, q_3), (q_1, q_2), (q_1, q_3), (q_2, q_3)\} + \text{simmetriche} + \text{riflessive}$$

$$F^2(Q \times Q) = \{(q_0, q_1), (q_0, q_2), (q_1, q_2)\} + \text{simmetriche} + \text{riflessive}$$

$$F^3(Q \times Q) = \{(q_0, q_1)\} + \text{simmetriche} + \text{riflessive}$$

$$F^4(Q \times Q) = F^3(Q \times Q)$$

9.8.1. Minimizzazione

Preso $TS = (Q, A, \rightarrow)$ e calcolata \sim con algoritmo iterativo, si costruisce un LTS minimo $TS' = (Q', A, \rightarrow')$ dove

$$Q = \{[q]_\sim \mid q \in Q\}$$

$$[q]_\sim = \{q' \in Q \mid q' \sim q\}$$

$$\rightarrow' = \{([q]_\sim, a, [q']_\sim) \mid (q, a, q') \in \rightarrow\}$$

L'LTS minimo rispetto a \sim va a fondere gli stati equivalenti.

Ogni stato $[q]_\sim$ è una classe d'equivalenza per gli stati TS . $\forall q, q' \in Q, q \sim q' \iff [q]_\sim = [q']_\sim$.

$([q]_\sim, \mu, [q']_\sim)$ è una transizione in $TS_\sim \Rightarrow \forall q_1 \in Q. q \sim q_1, \exists q_2 \in Q. q_1 \xrightarrow{\mu} q_2 \wedge q' \sim q_2 \Rightarrow ([q_1]_\sim, \mu, [q_2]_\sim) = ([q]_\sim, \mu, [q']_\sim)$. Quindi TS_\sim è indipendente dallo stato q per la classe d'equivalenza $[q]_\sim$.

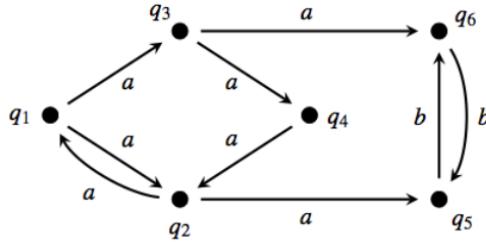
Proposizione Presi $TS = (Q, A, \rightarrow)$ e suo LTS minimo associato $TS_\sim = (Q_\sim, A, \rightarrow_\sim)$ allora valgono

- $q \sim [q]_\sim \forall q \in Q \wedge [q]_\sim \in Q_\sim$.
- $\forall [q]_\sim, [q']_\sim \in Q_\sim, \text{ se } [q]_\sim \sim [q']_\sim \Rightarrow [q]_\sim = [q']_\sim$.

Dimostrazione Per dimostrare il primo punto si prende in considerazione $R \subseteq Q \times Q_\sim$ definita come $R = \{(q, [q]_\sim) \mid q \in Q\}$. R è bisimulazione.

Per la seconda parte, si vede che $q \sim [q]_\sim$ come $q' \sim [q']_\sim$ (vale per la dimostrazione del punto 1). Se $[q]_\sim \sim [q']_\sim$ allora, per transitività, si ha $q \sim q'$ e così, per costruzione di TS_\sim , anche $[q]_\sim = [q']_\sim$. ■

Esempio

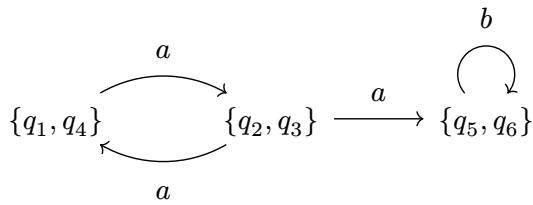


$$F^0(Q \times Q) = Q \times Q$$

$$F^1(Q \times Q) = \{(q_1, q_2), (q_1, q_3), (q_1, q_4), (q_2, q_3), (q_2, q_4), (q_3, q_4), (q_5, q_6)\} + \text{simmetriche} + \text{identità}$$

$$F^2(Q \times Q) = \{(q_1, q_4), (q_2, q_3), (q_5, q_6)\} + \text{simmetriche} + \text{identità}$$

$$F^3(Q \times Q) = F^2(Q \times Q)$$



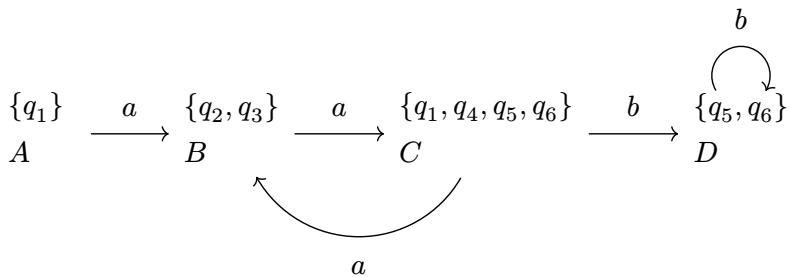
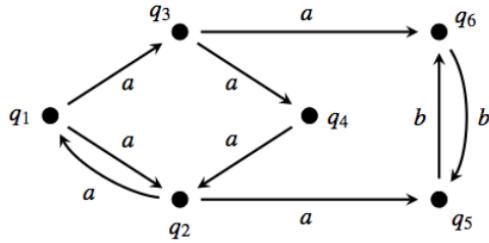
Il **minimo deterministico** di un LTS lo si può ottenere facendo:

1. Trasformare TS in deterministico dTS con la costruzione di sottoinsiemi.
2. Calcolare \sim sopra dTS (\sim e trace-equivalence coincidono su LTS deterministici).
3. Minimizzare dTS rispetto a \sim per ottenere il minimo LTS deterministico per TS .

Potrebbero essere LTS non deterministici equivalenti più piccoli.

Esempio

Trovare il minimo LTS deterministico di



$$F^0(Q \times Q) = Q \times Q$$

$$F^1(Q \times Q) = \{(A, B)\} + \text{simmetriche} + \text{identità}$$

$$F^2(Q \times Q) = \mathbf{Id}$$

dTS già minimo.

Preso un LTS $TS = (Q, A, \rightarrow)$, $\forall i \in \mathbb{N}$ si definisce la relazione \sim_i :

- $\sim_0 = Q \times Q$
 - $q_1 \sim_{i+1} q_2 \iff \forall \mu \in A$
1. $\forall q'_1.q_1 \xrightarrow{\mu} q'_1, \exists q'_2.q_2 \xrightarrow{\mu} q'_2 \wedge q'_1 \sim_i q'_2$
 2. $\forall q'_2.q_2 \xrightarrow{\mu} q'_2, \exists q'_1.q_1 \xrightarrow{\mu} q'_1 \wedge q'_1 \sim_i q'_2$

Se riescono a fare il gioco di bisimulazione lunghe i , allora fino a i sono bisimili.

Si denota \sim_ω la relazione $\bigcap_{i \in \mathbb{N}} \sim_i$ se per ogni i .

\sim_ω è una relazione d'equivalenza. $\forall i \in \mathbb{N}$ si ha

1. \sim_i relazione d'equivalenza;
2. $\sim_{i+1} \subseteq \sim_i$
3. $\sim_i = F^i(Q \times Q)$

L'ultimo è equivalente all' i -esimo passo dell'algoritmo usato.

La catena, possibilmente infinita, con \sim_ω ha come limite

$$\sim_0 = F^0(Q \times Q) \supseteq \sim_1 = F^1(Q \times Q) \supseteq \dots \supseteq \sim_i = F^i(Q \times Q) \supseteq \dots \sim_\omega$$

Teorema Se $TS = (Q, A, \rightarrow)$ è image finite, allora $\sim_\omega = \sim$

Dimostrazione Prima si vede $\sim \subseteq \sim_i \forall i$ per induzione su i . Per monotonia di F e il fatto che \sim è punto fisso per F , abbiamo $\sim = F(\sim) \subseteq F(\sim_i) = \sim_{i+1}$, e quindi $\sim \subseteq \sim_\omega$.

Per dimostrare $\sim_\omega \subseteq \sim$ si vede che la relazione $R = \{(q_1, q_2) \mid q_1 \xrightarrow{\mu} q_2\}$ è bisimulazione. Assumendo $(q_1 q_2) \in R$ quindi $q_1 \sim_i q_2 \quad \forall i \in \mathbb{N}$. Se $q_1 \xrightarrow{\mu} q'_1$ allora $\forall i. \exists q_{2_i}. q_2 \xrightarrow{\mu} q_{2_i}$ con $q'_1 \sim_i q_{2_i}$. Visto che l'LTS è image-finite, l'insieme $K = \{q_{2_k} \mid q_2 \xrightarrow{\mu} q_{2_k} \wedge q'_1 \sim_k q_{2_k} \wedge k \in \mathbb{N}\}$ è finito. Quindi c'è almeno un $q_{2_n} \in K$ t.c. $q'_1 \sim_i q_{2_n}$. Visto che $q \sim_i q'$ allora $q \sim_j q'$ per un $j < i$, allora si può concludere che $q'_1 \sim_i q_{2_n} \forall i$. Quindi $q'_1 \sim_\omega q_{2_n} \wedge (q'_1, q_{2_n}) \in R$. Il caso simmetrico si ha quando q_2 muove per primo, ma è uguale. Quindi $R = \sim_\omega$ è bisimulazione causa $\sim_\omega \subseteq \sim$. ■

Visto che \sim_ω controlla solo le computazione finite, serve che l'LTS sia image-finite.

10. Model checking

L'equivalence checking non è flessibile per descrivere proprietà puntuali o parziali. Non si può rispondere a cose del tipo “il sistema può fare subito a ma non b ”, oppure “come faccia a , poi può fare subito b ”, oppure “il sistema non può mai andare in deadlock”. *L'ultimo esempio è esprimibile con logica temporale, non modale.*

Questo perché l'equivalence checking viene espresso come $\text{Spec} \approx \text{Impl}$ in cui Spec è una specifica scritta in CCS descrivente il comportamento astratto del sistema che deve soddisfare, Impl è l'implementazione che deve essere corretta e \approx è un'equivalenza.

Preso sempre Impl si può definire un set di proprietà rappresentanti la specifica, la relazione di soddisfacimento $\text{Impl} \models F$ con F proprietà del set della specifica.

10.1. Logica Hennessy-Milner

È una logica modale, ovvero esprime “cosa può accedere ora”. Ha due operatori:

- Possibilità $\langle - \rangle$
- Necessità $[-]$

Esempi sono:

- Posso ricevere un messaggio all'inizio (computazione lunga 1) $P \models \langle \text{acc} \rangle tt$
- Non posso inoltrare inizialmente, nemmeno facendo un po' di τ prima e/o dopo (su computazioni “weak”) $P \models [[\overline{\text{del}}]] ff$
- Se ricevo un messaggio, posso inoltrarlo eventualmente facendo τ prima e/o dopo

$$P \models [\text{acc}] \langle \overline{\text{del}} \rangle tt$$

- Non posso ricevere due messaggi di seguito, nemmeno con τ fra le due azioni $P \models [\text{acc}][[\text{acc}]] ff$

Preso un LTS image-finite, due stati P e Q sono bisimili \iff soddisfano le stesse formule della logica HM.

10.2. Sintassi

L'insieme \mathcal{M} di formule su un set di azioni Act è definito come:

$$F, G ::= X \mid tt \mid ff \mid F \wedge G \mid F \vee G \mid \langle a \rangle F \mid [a]F$$

dove $a \in \text{Act}$, $tt = \text{true}$, $ff = \text{false}$, X è una singola variabile (vista in un HML con ricorsione).

$$\langle \{a_1, \dots, a_n\} \rangle F = \langle a_1 \rangle F \vee \langle a_2 \rangle F \vee \dots \vee \langle a_n \rangle F$$

$$[\{a_1, \dots, a_n\}] F = \langle a_1 \rangle F \wedge \langle a_2 \rangle F \wedge \dots \wedge \langle a_n \rangle F$$

10.3. Semantica

Si segna con $\llbracket F \rrbracket$ per denotare l'insieme dei processi in Q che soddisfano F .

- $\llbracket tt \rrbracket = Q$
- $\llbracket ff \rrbracket = \emptyset$
- $\llbracket F \wedge G \rrbracket = \llbracket F \rrbracket \cap \llbracket G \rrbracket$
- $\llbracket F \vee G \rrbracket = \llbracket F \rrbracket \cup \llbracket G \rrbracket$
- $\llbracket \langle a \rangle F \rrbracket = \langle \cdot a \cdot \rangle \llbracket F \rrbracket$
- $\llbracket [a]F \rrbracket = [\cdot a \cdot] \llbracket F \rrbracket$

Diamond può esser visto come \exists ; box come \forall .

$$\langle \cdot a \cdot \rangle S = \left\{ p \in Q \mid \exists p' \in S. p \xrightarrow{a} p' \right\}$$

$$[\cdot a \cdot]S = \left\{ p \in Q \mid \forall p'. \left(p \xrightarrow{a} p' \Rightarrow p' \in S \right) \right\}$$

$$p \in \llbracket F \rrbracket \iff p \models F$$

$$F \cong G \iff \forall \text{ LTS } \llbracket F \rrbracket = \llbracket G \rrbracket$$

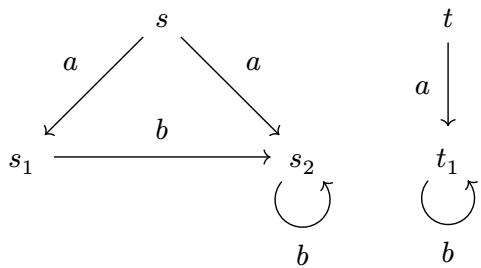
$P \models \langle a \rangle tt$ significa “ P può fare a adesso”.

$P \models [a]ff$ significa “ P non può fare a adesso”.

$$\langle a \rangle ff \equiv ff$$

$$[a]tt \equiv tt$$

Esempio



Stato che facendo a arriva a $s_1 \vee t_1$: $\langle \cdot a \cdot \rangle \{s_1, t_1\} = \{s, t\}$

Tutti gli stati che, se fanno a , arrivano a $s_1 \vee t_1$: $[\cdot a \cdot] \{s_1, t_1\} = \{s_1, s_2, t, t_1\}$ sono compresi gli stati che non possono fare a .

$$\langle \cdot b \cdot \rangle \{s_1, t_1\} = \{t_1\}$$

$$[\cdot b \cdot] \{s_1, t_1\} = \{s, t, t_1\}$$

Esempio

$[\text{coffee}] \langle \text{biscuit} \rangle tt$ = qualunque azioni di coffee in uscita, poi raggiungo uno stato in cui posso biscuit. Partendo da un box coffee allora ho anche gli altri che non fanno coffee all'inizio.

$\langle \text{coffee} \rangle tt \vee \langle \text{tea} \rangle tt$ = posso avere entrambi.

$\langle \text{coffee} \rangle tt \wedge [\text{tea}]ff$ = subito posso avere il caffè ma non il tea.

$[\text{coffee}][\text{coffee}] \langle \text{tea} \rangle tt$ = se faccio uno o due coffee, allora raggiungo uno stato in cui posso fare tea.

$[\text{coffee}][\text{tea}]ff$ = se prendo un caffè non potrò avere un tea.

Esempio

Presi $a.b + a.c$ e $a.(b + c)$ ho

- $\langle a \rangle [b]ff$ è soddisfatto solo dal primo.
- $[a](\langle b \rangle tt \wedge \langle c \rangle tt)$ è soddisfatto solo dal secondo.

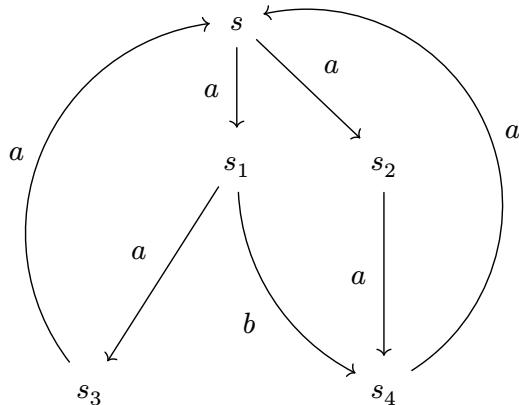
Presi $a.(b.c + b.d)$ e $a.b.c + a.b.d$ ho

- $[a]\langle b \rangle \langle c \rangle tt$ è soddisfatto solo dal primo.
- $\langle a \rangle [b] \langle c \rangle tt$ è soddisfatto solo dal secondo.

Presi $a.(b.c + b.d) + a.b.d$ e $a.(b.c + b.d)$ ho

- $\langle a \rangle [b] \langle d \rangle tt$ è soddisfatto solo dal primo.
- $[a]\langle b \rangle \langle c \rangle tt$ è soddisfatto solo dal secondo.

Esempio



- $s \not\models [a]\langle b \rangle tt$
- $s \models \langle a \rangle \langle b \rangle tt$
- $s \models [a]\langle a \rangle [a][b]ff$ perché finiscono tutti su s che non possono fare b
- $s \models \langle a \rangle (\langle a \rangle tt \wedge \langle b \rangle tt)$ perché c'è $s \rightarrow s_1$
- $s \models [a](\langle a \rangle tt \vee \langle b \rangle tt)$ perché sia s_1 che s_2 possono fare a o b
- $s \not\models \langle a \rangle ([b][a]ff \vee \langle b \rangle tt)$ perché s_4 può fare a ; il $\langle b \rangle tt$ mi obbliga a fare $s \rightarrow s_1$
- $s \not\models \langle a \rangle ([a](\langle a \rangle tt \vee [b]ff) \vee \langle b \rangle ff)$ perché sarebbe un $\langle a \rangle ff$ che è sempre ff
- $\llbracket [a][b]ff \rrbracket = [\cdot a \cdot]([\cdot b \cdot] \emptyset) = [\cdot a \cdot] \{s, s_2, s_3, s_4\} = \{s_1, s_2, s_3, s_4\}$. Non prendo s perché può finire in b .
- $\llbracket \langle a \rangle (\langle a \rangle tt \wedge \langle b \rangle tt) \rrbracket = \langle \cdot a \cdot \rangle (\langle \cdot a \cdot \rangle Q \cap \langle \cdot b \cdot \rangle Q) = \langle \cdot a \cdot \rangle (Q \cap \{s_1\}) = \langle \cdot a \cdot \rangle \{s_1\} = \{s\}$

10.4. Negazione

Non c'è un costrutto per fare negazione ma si può creare una formula di complemento corrispondente ad una negazione vera e propria

$F \rightarrow F^c$ dove $\llbracket F^c \rrbracket = Q \setminus \llbracket F \rrbracket = \llbracket \neg F \rrbracket$

con la sintassi definita come:

- $tt^c = ff$
- $ff^c = tt$
- $(F \wedge G)^c = F^c \vee G^c$
- $(F \vee G)^c = F^c \wedge G^c$
- $(\langle a \rangle F)^c = [a]F^c$
- $([a]F)^c = \langle a \rangle F^c$
- $(\langle a \rangle tt)^c = [a]ff$
- $([a]ff)^c = \langle a \rangle tt$

10.5. Bisimilarità e soddisfacibilità

Presi $A = a.A + a.\mathbf{0}$ e $B = a.a.B + a.\mathbf{0}$ essi non sono bisimili perché $A \sim a.B$ che fanno entrambi a ma $\mathbf{0} \sim B$.

Preso invece $F = \langle a \rangle \langle a \rangle [a]ff$ esso $\models A$ ma $\not\models B$. $B \models \langle a \rangle [a] \langle a \rangle tt$ ma $\not\models A$ perché A c'ha quel $\mathbf{0}$.

Teorema Preso un LTS $TS = (\text{Proc}, A, \rightarrow)$ image-finite. $\forall P, Q \in \text{Proc}$ si ha

$$P \sim Q \iff (P \models F \iff Q \models \forall F)$$

ovvero, P e P' sono bisimili \iff soddisfano lo stesso insieme di formule.

Dimostrazione

(\Rightarrow) non serve che si assuma LTS image-finite. Preso $P \sim Q \wedge P \models F$ per simmetria bisogna trovare che $Q \models F$ per induzione strutturale su F .

- $F = tt$, per definizione di $\models, Q \models tt$.
- $F = ff$, impossibile perché $P \not\models ff$.
- $F = F_1 \wedge F_2$, con ipotesi induttiva $R \sim S \wedge R \models F_i \Rightarrow S \models F_i, i = 1, 2$, si ha che $Q \models F_i, i = 1, 2$. Allora per definizione di \models si ha $Q \models F_1 \wedge F_2$.
- $F = F_1 \vee F_2$ è analogo a sopra.
- $F = \langle a \rangle G$, con ipotesi induttiva $R \sim S \wedge R \models G \Rightarrow S \models G$. Sapendo che $P \sim Q \wedge P \models \langle a \rangle G$, per definizione di $\models, \exists P'. P \xrightarrow{a} P' \wedge P' \models G$. Per definizione di $\sim, \exists Q'. Q \xrightarrow{a} Q' \wedge P' \sim Q'$. Allora, per ipotesi induttiva, $Q' \models G$ e per definizione di $\models, Q \models \langle a \rangle G$.
- $F = [a]G$, con ipotesi induttiva $R \sim S \wedge R \models G \Rightarrow S \models G$. Sapendo che $P \sim Q \wedge P \models [a]G$. Per definizione di $\models, \forall P'. P \xrightarrow{a} P', P' \models G$. Per definizione di $\sim, \forall Q \xrightarrow{a} Q', \exists P'. P \xrightarrow{a} P', P' \sim Q'$. Allora, per induzione, $Q' \models G \quad \forall Q'$ che derivano con a . Allora, per definizione di $\models, Q \models [a]G$.

(\Leftarrow) P e Q soddisfano la stessa formula che implica $P \sim Q$. Adesso si prende l'ipotesi di image-finite. Si prova che $R = \{(P, Q) \mid P$ e Q soddisfano la stessa formula $\} \in \text{R}$ è una bisimulazione (R è simmetrica).

Assumendo $(P, Q) \in R \wedge P \xrightarrow{a} P'$ si prova che $\exists Q'. Q \xrightarrow{a} Q' \wedge (P', Q') \in R$. Dato che R è simmetrica, basta provare che R è bisimulazione.

Assumendo per assurdo che $\nexists Q'. Q \xrightarrow{a} Q' \wedge (P', Q') \in R$. Dato che l'LTS è image-finite, l'insieme $\{Q' \mid Q \xrightarrow{a} Q'\}$ è finito denotato con $\{Q_1, Q_2, \dots, Q_n\}$ con $n \geq 0$. Da la cosa assunta prima $(P', Q') \notin R$ per $Q' \in \{Q_1, Q_2, \dots, Q_n\}$ e dunque $\nexists Q_i. Q_i \models F_i$. Quindi, $F = \langle a \rangle (F_1 \wedge F_2 \dots \wedge F_n)$

è t.c. $P \models F$ ma $Q \not\models F$. Quindi si ha una contraddizione perché $(P, Q) \notin R$. Quindi l'ipotesi che $\nexists Q'. Q \xrightarrow{a} Q' \wedge (P', Q') \in R$ è sbagliata e quindi R è bisimulazione. ■

10.6. HML con ricorsione

La versione con proprietà temporali è eseguita mediante ricorsione e rispetta due proprietà:

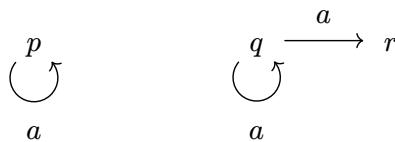
1. Safety: qualcosa di brutto non può accadere.
2. Liveness: qualcosa di buono potrebbe accadere.

Necessita di minimi e massimi punti fissi e la si può vedere come logica temporale.

Esempio $P \models X \quad X \stackrel{\max}{=} \langle \text{Act} \rangle tt \wedge [\text{Act}]X$ in cui X è massimo punto fisso. Lo si può leggere come “posso fare un'azione” \wedge “qualunque azione fatta X vale”. Non è raggiungibile uno stato di deadlock.

Esempio $P \models Y \quad Y \stackrel{\max}{=} F \wedge [\text{acc}](\langle \overline{\text{del}} \rangle Y \wedge [[\text{acc}]]ff) \quad F = \langle \text{acc} \rangle tt \wedge [[\overline{\text{del}}]]ff$. Lo si può leggere come “se faccio acc, raggiungo uno stato in cui posso fare $\overline{\text{del}}$ ma non posso fare acc”, oppure anche come “alternanza di acc e $\overline{\text{del}}$ ”.

Non si può, ad esempio, presi i due LTS sotto



$$p \models [a]\langle a \rangle tt \wedge q \not\models [a]\langle a \rangle tt$$

Definire un modo per far valere la formula per due stati qualsiasi con n , cioè un modo di dire “per tutti gli stati $[a]$ deve essere valida l'azione dopo.”

$$p \models [a]^{n+1}\langle a \rangle tt \wedge q \not\models [a]^{n+1}\langle a \rangle tt$$

Definiamo dunque

- **Invariabilmente**

$$\text{Inv}(\langle a \rangle tt) = \langle a \rangle tt \wedge [a]\langle a \rangle tt \wedge [a][a]\langle a \rangle tt \wedge \dots = \bigwedge_{i \geq 0} [a]^i \langle a \rangle tt$$

Ovvero

$$X \equiv \langle a \rangle tt \wedge [a]X$$

Per trovare soluzione a $S = \langle \cdot a \cdot \rangle Q \cap [\cdot a \cdot]S$ si cerca Z_{\max} .

Un processo potrebbe non avere la proprietà $\text{Inv}(\langle a \rangle tt)$ perché raggiunge uno stato in cui non è possibile fare a .

La proprietà **safety** è data

$$X \stackrel{\max}{=} F \wedge [\text{Act}]X$$

in cui F vale per tutti gli stati raggiungibili.

Per come è definito, non si può raggiungere un deadlock.

- **Possibilmente**

$$\text{Pos}([a]tt) = [a]ff \vee \langle a \rangle [a]ff \vee \langle a \rangle \langle a \rangle [a]ff \vee \dots = \bigvee_{i \geq 0} \langle a \rangle^i [a]ff$$

Ovvero

$$Y \equiv [a]ff \vee \langle a \rangle Y$$

Per trovare soluzione a $S = [\cdot a \cdot] \emptyset \cup \langle \cdot a \cdot \rangle S$ si cerca Z_{\min} .

Ad esempio, un processo ha $\text{Pos}(\langle a \rangle tt)$ se può raggiungere uno stato che fa a .

Non vado avanti essendo un Z_{\min} , dunque c'è uno stato in cui F vale.

$$Y \stackrel{\min}{=} F \vee \langle \text{Act} \rangle Y$$

Per come è definito, si può raggiungere un deadlock.

- $\text{Inv}(\langle \text{Act} \rangle tt)$

Significa che non è possibile raggiungere un deadlock.

- $\text{Pos}([\text{Act}]ff)$

Significa che potrebbe raggiungere un deadlock.

- $\text{Inv}(F)^c = \text{Pos}(F^c)$

Che poi ha senso per la questione Z_{\min} e Z_{\max} .

- $\text{Inv}(\langle a \rangle tt)$

è uguale a $X \stackrel{\max}{=} \langle a \rangle tt \wedge [a]X$.

Mi serve Z_{\max} perché non ho uno stato in cui non posso fare a : in questo caso a può essere fatto e in qualsiasi transizione raggiunta, a è comunque eseguita.

- $\text{Pos}([a]ff)$

è uguale a $Y \stackrel{\min}{=} [a]ff \vee \langle a \rangle Y$.

Si dimostra solo trovando uno stato in cui non posso fare a , dunque Z_{\min} . a non può essere fatta oppure c'è una transizione con etichetta a che porta ad uno stato in cui tale proprietà è valida.

- $\text{Safe}(F)$

è una proprietà che vale se c'è una computazione completa (che sia finita o no) che attraversa ogni stato che soddisfa la formula.

È uguale a $X \stackrel{\max}{=} F \wedge ([\text{Act}]ff \vee \langle \text{Act} \rangle X)$ e si calcola dunque con Z_{\max} .

- $\text{Even}(F)$

è una proprietà che vale se ogni computazione completa contiene almeno uno stato in cui soddisfa la formula.

È uguale a $Y \stackrel{\min}{=} F \vee (\langle \text{Act} \rangle tt \wedge [\text{Act}]Y)$ e si calcola dunque con Z_{\min} .

- $\text{Safe}(F)^c = \text{Even}(F^c)$

- **Strong Until**

$$F \mathcal{U}^s G \stackrel{\text{min}}{=} G \vee (F \wedge \langle \text{Act} \rangle tt \wedge [\text{Act}](F \mathcal{U}^s G))$$

Formula che descrive l'evento in cui p raggiungerà prima o poi uno stato in cui la formula G è vera e ogni stato prima F deve essere vera. F potrebbe essere uguale a tt .

È uguale a $\text{Even}(G) \equiv tt \mathcal{U}^s G$

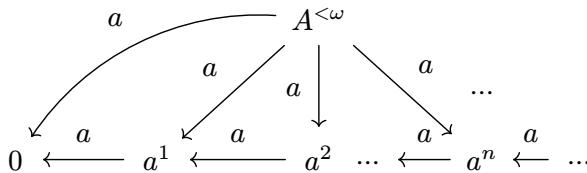
- **Weak Until**

$$F \mathcal{U}^w G \stackrel{\text{max}}{=} G \vee (F \wedge [\text{Act}])(F \mathcal{U}^w G)$$

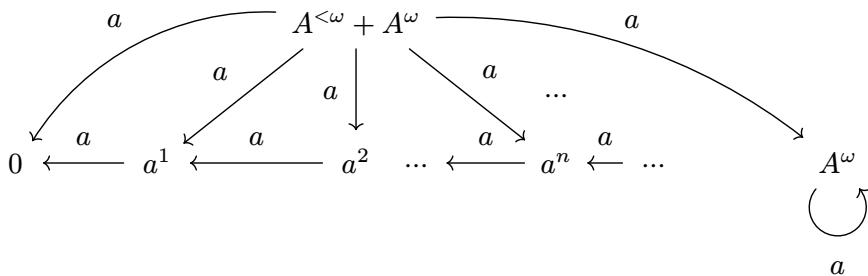
F deve valere in tutti gli stati finché non è G a valere (ma questo potrebbe non accadere mai).

È uguale a $\text{Inv}(F) \equiv F \mathcal{U}^w ff$

Esempio



è finto ma non è bisimile a



perché può fare a infinite volte.

$$A^{<\omega} + A^\omega \models \langle a \rangle \text{Inv}(\langle a \rangle tt)$$

$$A^{<\omega} \models [a] \text{Pos}([a]ff)$$

10.6.1. Semantica per formule aperte

Una formula è detta “aperta” quando vi è una variabile aperta X . In questo caso è interpretata dalla funzione $\mathcal{O}_F : 2^Q \mapsto 2^Q$: ritorna un insieme di processi che soddisfano F dato un insieme di processi in cui si presume soddisfino X .

Preso un $TS = (Q, A, \{ \xrightarrow{a} \mid a \in A \})$, per ogni $S \subseteq Q$ e F si definisce:

$$\mathcal{O}_X(S) = S$$

$$\mathcal{O}_{tt}(S) = Q$$

$$\mathcal{O}_{ff}(S) = \emptyset$$

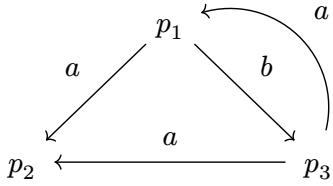
$$\mathcal{O}_{F_1 \wedge F_2}(S) = \mathcal{O}_{F_1}(S) \cap \mathcal{O}_{F_2}(S)$$

$$\mathcal{O}_{F_1 \vee F_2}(S) = \mathcal{O}_{F_1}(S) \cup \mathcal{O}_{F_2}(S)$$

$$\mathcal{O}_{\langle a \rangle F}(S) = \langle \cdot a \cdot \rangle \mathcal{O}_F(S)$$

$$\mathcal{O}_{[a]F}(S) = [\cdot a \cdot] \mathcal{O}_F(S)$$

Esempio



Preso $F = \langle a \rangle X$. Se X è soddisfatta da p_q , allora $\langle a \rangle X$ sarà soddisfatta da p_3 e lo si vede dall'insieme $\mathcal{O}_{\langle a \rangle X}(\{p_1\}) = \{p_3\}$. Se l'insieme che soddisfa X è $\{p_1, p_2\}$ allora $\mathcal{O}_{\langle a \rangle X}(\{p_1, p_2\}) = \{p_1, p_3\}$.

Invce, $\mathcal{O}_{[b]X}(\{p_2\}) = \{p_2, p_3\}$.

10.6.2. Semantica per formule aperte

Un'equazione può essere interpretata come insieme per l'equazione

$$[\![X]\!] = \mathcal{O}_{F_X}([\![X]\!])$$

Vista la monotonia di \mathcal{O}_{F_X} allora esso ha dei punti fissi, tra cui:

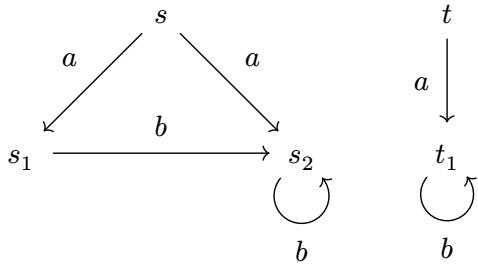
$$\text{FIX } \mathcal{O}_{F_X} = \bigcup \{ S \subseteq Q \mid S \subseteq \mathcal{O}_{F_X}(S) \}$$

$$\text{fix } \mathcal{O}_{F_X} = \bigcap \{ S \subseteq Q \mid S \supseteq \mathcal{O}_{F_X}(S) \}$$

Se Q è finito, allora $\text{FIX } \mathcal{O}_{F_X} = (\mathcal{O}_{F_X})^M(Q)$ per qualche M e $\text{fix } \mathcal{O}_{F_X} = (\mathcal{O}_{F_X})^M(\emptyset)$ per qualche m .

Esempio

$$X \stackrel{\max}{=} F_X \text{ con } F_X = \langle b \rangle tt \wedge [b]X$$



quindi si cerca la più grande soluzione per

$$[\![X]\!] = (\langle \cdot b \cdot \rangle \{s, s_1, s_2, t, t_1\}) \cap [\cdot b \cdot] [\![X]\!].$$

Da qui il più grande punto fisso per

$$\mathcal{O}_{F_X}(S) = (\langle \cdot b \cdot \rangle \{s, s_1, s_2, t, t_1\}) \cap [\cdot b \cdot] S \text{ trovato come:}$$

$$\mathcal{O}_{F_X}(s, s_1, s_2, t, t_1) = (\langle \cdot b \cdot \rangle \{s, s_1, s_2, t, t_1\}) \cap [\cdot b \cdot](s, s_1, s_2, t, t_1) =$$

$$\begin{aligned} &= \{s_1, s_2, t_1\} \cap \{s, s_1, s_2, t, t_1\} \\ &= \{s_1, s_2, t_1\} \end{aligned}$$

$$\mathcal{O}_{F_X}(s_1, s_2, t_1) = (\langle \cdot b \cdot \rangle \{s, s_1, s_2, t, t_1\}) \cap [\cdot b \cdot](s_1, s_2, t_1) =$$

$$\begin{aligned} &= \{s_1, s_2, t_1\} \cap \{s, s_1, s_2, t, t_1\} \\ &= \{s_1, s_2, t_1\} \end{aligned}$$

quindi $\{s_1, s_2, t_1\}$ è il più grande punto fisso.

Preso $\text{Inv}(F)$, ovvero i casi in cui F vale per ogni stato in ogni sequenza di transizione, possiamo definire $\mathcal{J} : 2^Q \mapsto 2^Q$ come $\mathcal{J}(S) = [\![F]\!] \cap [\cdot \text{Act} \cdot] S$ e si trova l'unica soluzione $\text{FIX } \mathcal{J} = \bigcup \{S \mid S \subseteq \mathcal{J}(S)\}$.

Per Z_{\max} si vede $\text{Inv} = \{p \mid p \xrightarrow{\sigma} p' \Rightarrow p' \in [\![F]\!] \forall \sigma \in A^* \wedge p' \in Q\}$.

Dunque, per ogni $TS = (Q, A, \left\{ \xrightarrow{a} \mid a \in \text{Act} \right\})$, $\text{Inv} = \text{FIX } \mathcal{J}$ vale.

Dimostrazione

- $\text{Inv} \subseteq \text{FIX } \mathcal{J}$

Basta dimostrare $\text{Inv} \subseteq \mathcal{J}(\text{Inv})$ e se questo è vero, allora è un punto fisso, ma $\text{FIX } \mathcal{J}$ è il sup dei punti fisi.

Preso $p \in \mathcal{J}(\text{Inv})$ che poi è equivalente a $p \in [\![F]\!] \wedge p \in [\cdot \text{Act} \cdot] \text{ Inv}$. Con $\sigma = \varepsilon$ si ha che $p \xrightarrow{\varepsilon} p$ regge.

Per provare $p \in [\cdot \text{Act} \cdot] \text{ Inv}$ dobbiamo vedere che per ogni p' e a

$$p \xrightarrow{a} p' \Rightarrow p' \in \text{Inv}$$

che è equivalente a provare che per ogni sequenza di azioni σ' e p'' si ha

$$p \xrightarrow{a} p' \wedge p' \xrightarrow{\sigma'} p'' \Rightarrow p'' \in [\![F]\!]$$

e segue $\sigma = a\sigma'$.

- $\text{FIX } \mathcal{I} \subseteq \text{Inv}$

Per dimostrare che $\text{FIX } \mathcal{I} (= \llbracket F \rrbracket \cap [\cdot \text{ Act } \cdot] \text{ FIX } \mathcal{I}) \subseteq \text{Inv}$ si assume che $p \in \text{FIX } \mathcal{I} \wedge p \xrightarrow{\sigma} p'$ e si dimostra che $p' \in \llbracket F \rrbracket$ su induzione per $|\sigma|$.

– $\sigma = \varepsilon$. Si ha $p = p'$ e per assuzione $p \in \text{FIX } \mathcal{I}$ vale che $p' \in \llbracket F \rrbracket$.

– $\sigma = a\sigma'$. Si ha $p \xrightarrow{a} p'' \xrightarrow{\sigma'} p'$ per qualche p'' . Per assunzione $p \in \text{FIX } \mathcal{I}$ e segue che $p'' \in \text{FIX } \mathcal{I}$. Dato $|\sigma'| < |\sigma| \wedge p'' \in \text{FIX } \mathcal{I}$, per ipotesi induttiva si conclude che $p' \in \llbracket F \rrbracket$ come richiesto.

■

10.6.3. Mutua ricorsione

Preso l'insieme delle variabili $\mathcal{X} = \{X_1, \dots, X_n\}$ e le formule F_{X_i} le quali possono contenere ogni variabile in \mathcal{X} .

Un esempio è $X = [a]Y, Y = \langle a \rangle X$.

Semanticamente si ha

$$S_1 = \mathcal{O}_{F_{X_1}}(S_1, \dots, S_n),$$

...

$$S_n = \mathcal{O}_{F_{X_n}}(S_1, \dots, S_n),$$

con un $\text{dom} = 2^{Q^n}$ per n è il numero di volte di prodotto di 2^Q con se stesso. $(S_1, \dots, S_n) \sqsubseteq (S'_1, \dots, S'_{n'})$ if $S_1 \subseteq S'_1 \wedge S_2 \subseteq S'_{2'} \wedge \dots \wedge S_n \subseteq S'_{n'}$.

La coppia $(\mathcal{D}, \sqsubseteq)$ è definita con min e sup upper bound, su computazioni tra insiemi di componenti.

$$\bigsqcup \{(A_1^i, \dots, A_n^i) \mid i \in I\} = (\bigcup \{A_1^i \mid i \in I\}, \dots, \bigcup \{A_n^i \mid i \in I\})$$

$$\bigsqcap \{(A_1^i, \dots, A_n^i) \mid i \in I\} = (\bigcap \{A_1^i \mid i \in I\}, \dots, \bigcap \{A_n^i \mid i \in I\})$$

con I indice. Presa D dichiarazione su insieme di variabili $X = \{X_1, \dots, X_n\}$ che associa una formula F_{X_i} su ogni variabile $X_i, 1 \leq i \leq n$. Si cerca la più grande o la minima soluzione per l'equazione

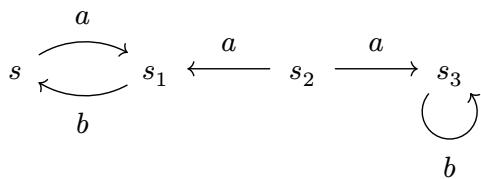
$$\llbracket D \rrbracket(S_1, \dots, S_n) = \left(\mathcal{O}_{F_{X_1}}(S_1, \dots, S_n), \dots, \mathcal{O}_{F_{X_n}}(S_1, \dots, S_n) \right) \text{ dove}$$

$$\mathcal{O}_{X_i}(S_1, \dots, S_n) = S_i \text{ con } 1 \leq i \leq n$$

Esempio

$$X \stackrel{\max}{=} \langle a \rangle Y \wedge [a]Y \wedge [b]ff$$

$$Y \stackrel{\max}{=} \langle b \rangle X \wedge [b]X \wedge [a]ff$$



bisogna trovare il più grande punto fisso che mappa (S_1, S_2) a $(\langle \cdot a \cdot \rangle S_2 \cap [\cdot a \cdot] S_2 \cap \{s, s_2\}, \langle \cdot b \cdot \rangle S_1 \cap [\cdot b \cdot] S_1 \cap \{s_1, s_3\})$

dove S_1 è l'insieme degli stati che soddisfano X ; S_2 è l'insieme degli stati che soddisfano Y ; $\langle \cdot a \cdot \rangle S_2 \cap [\cdot a \cdot] S_2 \cap \{s, s_2\}$ è l'insieme degli stati che soddisfano la parte destra della definizione di X ; $\langle \cdot b \cdot \rangle S_1 \cap [\cdot b \cdot] S_1 \cap \{s_1, s_3\}$ è l'insieme degli stati che soddisfano la parte destra della definizione di Y .

Si inizia con $(S_1, S_2) = (\{s, s_1, s_2, s_3\}, \{s, s_1, s_2, s_3\})$. Applicando la funzione si ha $(\{s, s_2\}, \{s_1, s_3\})$.

Con $(\{s, s_2\}, \{s_1, s_3\})$, applicando la funzione, si ha $(\{s, s_2\}, \{s_1\})$.

Con $(\{s, s_2\}, \{s_1\})$, applicando la funzione, si ha $(\{s\}, \{s_1\})$.

Con $(\{s\}, \{s_1\})$, applicando la funzione, si ha $(\{s\}, \{s_1\})$. Dunque è il Z_{\max} .

Vi sono casi in cui il sistema raggiunge uno stato in cui può divergere. Ad esempio, $\text{Pos}(F) \stackrel{\min}{=} F \vee \langle \text{Act} \rangle \text{Pos}(F), F \stackrel{\max}{=} \langle \tau \rangle F$ non sono mutualmente ricorsive, ma è come sono stati costruiti i blocchi uno sopra l'altro. Prima si computa la semantica per F e poi per Pos .

10.6.3.1. Mutua ricorsione annidata

Una n -annidata equazione mutualmente ricorsiva per un sistema E è una n -upla definita come

$$\langle (D_1, \mathcal{X}_1, m_1), (D_2, \mathcal{X}_2, m_2), \dots, (D_n, \mathcal{X}_n, m_n) \rangle$$

con \mathcal{X}_i una coppia per insieme finito di variabili per $1 \leq i \leq n$.

D_i è una dichiarazione che mappa le variabili dell'insieme \mathcal{X}_i per formule HML con ricorsione che possono usare variabili in $\bigcup_{1 \leq j \leq i} \mathcal{X}_j$.

$$m_i = \max \vee m_i = \min$$

$m_i \neq m_{i+1}$ perché l'insieme di equazioni ha unica soluzione che viene ottenuta risolvendo un blocco usando le soluzioni di quello precedente.

11. Mutua esclusione

11.1. Algoritmo Peterson

Visto anche nel corso di Sistemi Operativi, viene usato per sincronizzare due processi per l'uso di risorse condivise usando delle sezioni chiamate *critiche*.

```

while true do
begin
    'non critical section'
     $b_i := \text{true}$ 
     $k := j$ 
    while  $b_j$  and  $k = j$  do skip
    'critical section'
     $b_i := \text{false}$ 
end
```

In modo intuitivo è corretto e assicura mutua esclusione ma l'ultimo che assegna k resta bloccato.

Per modellare l'algoritmo usando CCS si definisce una variabile b_1 che può avere stato input (w) od output (r) e salva valore f (false) o t (true).

$$B_{1f} \stackrel{\text{def}}{=} \overline{b1rf}.B_{1f} + b1wf.B_{1f} + b1wt.B_{1t}$$

- $\overline{b1rf}$ sta a dire che il valore di lettura di b_1 è false.
- $b1wt$ voglio scrivere il valore true, e quindi ritorna di B_{1t} .

$$B_{1t} \stackrel{\text{def}}{=} \overline{b1rt}.B_{1t} + b1wf.B_{1f} + b1wt.B_{1t}$$

$$B_{2f} \stackrel{\text{def}}{=} \overline{b2rf}.B_{2f} + b2wf.B_{2f} + b2wt.B_{2t}$$

$$B_{2t} \stackrel{\text{def}}{=} \overline{b2rt}.B_{2t} + b2wf.B_{2f} + b2wt.B_{2t}$$

$$K_1 \stackrel{\text{def}}{=} \overline{kr1}.K_1 + kw1.K_1 + kw2.K_2$$

$$K_2 \stackrel{\text{def}}{=} \overline{kr2}.K_2 + kw1.K_1 + kw2.K_2$$

Si ignora la sezione non critica: i processi non influiscono su come funziona la mutua esclusione; si ignora anche la sezione critica perché finisce in tempo finito. Si ha focus solo sul meccanismo di mutua esclusione. L'esecuzione della sezione critica è sempre eseguita in tempo finito (no deadlock o divergenza).

$$P_1 \stackrel{\text{def}}{=} \overline{b1wt}.\overline{kw2}.P_{11}$$

$$P_{11} \stackrel{\text{def}}{=} b2rf.P_{12} + b2rt.(kr2.P_{11} + kr1.P_{12})$$

$$P_{12} \stackrel{\text{def}}{=} \text{enter}_1.\text{exit}_1.\overline{b1wf}.P_1$$

$$P_2 \stackrel{\text{def}}{=} \overline{b2wt}.\overline{kw1}.P_{21}$$

$$P_{21} \stackrel{\text{def}}{=} b1rf.P_{22} + b1rt.(kr1.P_{21} + kr2.P_{22})$$

$$P_{22} \stackrel{\text{def}}{=} \text{enter}_2.\text{exit}_2.\overline{b2wf}.P_2$$

$$\text{Peterson} = (\nu L)(P_1 \mid P_2 \mid B_{1f} \mid B_{2f} \mid K_1)$$

dove $L = A \setminus \{\text{enter}_1, \text{enter}_2, \text{exit}_1, \text{exit}_2\}$ (che poi sono le uniche azioni osservabili).

Con HML possiamo definire safety con $F \stackrel{\text{def}}{=} [\text{exit}_1]ff \vee [\text{exit}_2]ff$ che è vera nello stato se non è possibile eseguire entrambi. La formula completa è calcolata col punto fisso $X \stackrel{\text{max}}{=} F \wedge [A]X$. Il calcolo del più grande punto fisso si calcola come

$S \mapsto \llbracket F \rrbracket \cap [\cdot, A \cdot]S$ o con CWB usando >checkprop.

Esempio

$\text{Inv}(G)$ con $G = ([\text{enter}_1][\text{enter}_2]ff) \wedge ([\text{enter}_2][\text{enter}_1]ff)$ è una buona specifica di mutua esclusione? Sì.

$\text{Inv}(H)$ con $H = (\langle \text{enter}_1 \rangle [\text{enter}_2]ff) \wedge (\langle \text{enter}_2 \rangle [\text{enter}_1]ff)$ è una buona specifica di mutua esclusione? No, perché ci potrebbe essere un enter_1 che porta in enter_2 e viceversa.

11.2. Algoritmo Hyman

Viene costruito sulle stesse ipotesi di Peterson. Non è corretto perché i due processori potrebbero essere di velocità diversa e quindi ci sono casi in cui più processi entrano entrambi in sezione critica.

```

while true do
begin
  'non critical section'
   $b_i := \text{true}$ 
  while  $k \neq i$  do begin
    while  $b_j$  do skip
     $k := i$ 
  end
  'critical section'
   $b_i := \text{false}$ 
end
```

11.3. Equivalenza

Una possibile specifica è

$$\text{MutexSpec} \stackrel{\text{def}}{=} \text{enter}_1.\text{exit}_1. \text{MutexSpec} + \text{enter}_2.\text{exit}_2. \text{MutexSpec}$$

per fare equivalence-checking bisogna vedere che non si può scegliere la strong bisimilarity perché Peterson esegue τ ma MutexSpec no; non si può scegliere la weak bisimilarity perché Peterson non è weak bisimilar a MutexSpec.

Ad esempio, Peterson $\xrightarrow{\tau} P \xrightarrow{\text{enter}_1}$ dove P non può eseguire enter_2 nemmeno in modo weak. MutexSpec può rispondere a τ raggiungendo uno stato che può fare sia enter_1 che enter_2 : non sono weak bisimilar.

La mutua esclusione è una proprietà safety: non c'è bisogno dell'uso di equivalenza per bisimulazione. Si usano l'equivalenza a tracce perché ogni traccia weak di Peterson lo è anche di MutexSpec.

$$WTr(\text{Peterson}) \subseteq Tr(\text{MutexSpec})$$

ovvero, ogni esecuzione di Peterson è conforme ad una di mutua esclusione (anche **0** soddisfa la proprietà).

$$Tr(\text{MutexSpec}) \subseteq WTr(\text{Peterson})$$

ovvero, ogni comportamento di mutua esclusione è possibile con Peterson.

$$WTr(\text{MutexSpec}) = WTr(\text{Peterson})$$

12. π calcolo

$$P ::= \mathbf{0} \mid P|P \mid (\nu a)P \mid a(x).P \mid \bar{a}b.P \mid !P$$

Ovvero, processo nullo, parallelo, restrizione, input, output e replicazione.

Alcune regole semantiche sono

$$[\text{inp}] \frac{}{a(b).P \xrightarrow{a(b)} P}$$

$$[\text{out}] \frac{}{\bar{a}b.P \xrightarrow{\bar{a}b} P}$$

$$[\text{parL}] \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}$$

con $bn(\alpha) \cap fn(Q) = \emptyset$

$$[\text{comL}] \frac{P \xrightarrow{\alpha(x)} P' \quad Q \xrightarrow{\bar{a}b} Q'}{P|Q \xrightarrow{\tau} P'\{b/x\}|Q'}$$

$$[\text{closeL}] \frac{P \xrightarrow{\alpha(x)} P' \quad Q \xrightarrow{\bar{a}b} Q'}{P|Q \xrightarrow{\tau} \nu b(P'\{b/x\}|Q')}$$

con $b \notin fn(P)$

$$[\text{res}] \frac{P \xrightarrow{\alpha} P'}{\nu a P \xrightarrow{\alpha} \nu a P'}$$

con $a \notin n(\alpha)$

$$[\text{open}] \frac{P \xrightarrow{\bar{a}b} P'}{\nu b P \xrightarrow{\bar{a}b} P'}$$

con $a \neq b$

$$[\text{rep}] \frac{P|!P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}$$

$$[\text{alpha}] \frac{P =_\alpha Q \quad Q \xrightarrow{\alpha} Q'}{P \xrightarrow{\alpha} Q'}$$

La congruenza strutturale vede

1. $P \equiv Q$ se $P =_\alpha Q$
2. $P|\mathbf{0} \equiv P$
3. $P|Q \equiv Q|P$
4. $P|(Q|R) \equiv (P|Q)|R$
5. $\nu x \mathbf{0} \equiv \mathbf{0}$
6. $\nu x \nu y P \equiv \nu y \nu x P$
7. $\nu x (P|Q) \equiv (\nu x P)|Q$ se $x \notin fn(Q)$

8. $\mathbf{!}P \equiv P \mathbf{|} !P$

La relazione di riduzione (\rightarrow) ha come teorema: $P \rightarrow P' \iff P \xrightarrow{\tau} P'' \equiv P'$. Ad esempio, $x(y).P \mid R \mid \bar{x}z.Q \equiv \bar{x}z.Q \mid x(y).P \mid R \rightarrow Q \mid P\{z/x\} \mid R \equiv P\{z/x\} \mid R \mid Q$

Vi sono le semantiche operation early e late alternative (ma qui non le scrivo, tanto vale leggere le slide o pdf).

12.1. Bisimulazione weak barbed

P ha una forte barba su $x \in \mathbb{N}$, e si segna come $P \downarrow_x \iff$ esiste P' con $P \xrightarrow{x(y)} P'$ per qualche $y \in \mathbb{N}$.

P ha una forte barba su $\bar{x} \in \mathbb{N}$, e si segna come $P \downarrow_{\bar{x}} \iff$ esiste P' con $P \xrightarrow{\bar{x}(z)} P'$ per qualche $z \in \mathbb{N}$.

P ha una debole barba su $a \in \{x, \bar{x} | x \in N\}$, e si segna come $P \Downarrow_a \iff$ esiste P' con $P \xrightarrow{\tau}^* P'$ e $P' \downarrow_a$.

Una relazione \mathcal{R} su \mathcal{P}_π è una bisimulazione weak barbed $\iff P \mathcal{R} Q \Rightarrow$

1. Se $P \downarrow_a$ con $a \in \{x, \bar{x} | x \in N\}$ allora $Q \Downarrow_a$.
2. Se $P \xrightarrow{\tau} P'$ allora esiste $Q \xrightarrow{\tau}^* Q' \wedge P' \mathcal{R} Q'$.

La più grande bisimulazione weak barbed è denotata come \approx^\bullet oppure \approx_{WBB} .

La (weak) barbed congruence è la congruenza ottenuta chiudendo per contesti la (weak) barbed bisimulation equivalence.

Coincide con l'equivalenza (weak) early bisimulation congruence su LTS del π -calcolo.

Esempio

$$P = a(b).\mathbf{0}$$

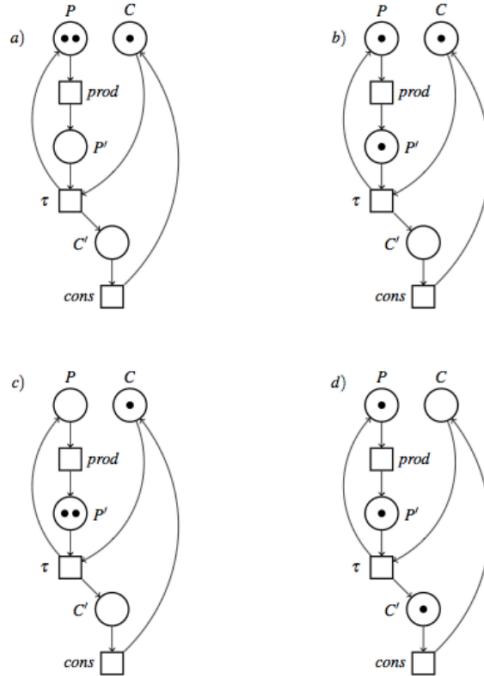
$$Q = a(x).(\nu c).\bar{c}b$$

sono equivalenti perché fanno un input e poi terminano ma il primo $P \xrightarrow{a(b)} \mathbf{0}$ mentre il secondo no nemmeno con alpha-conversione perché $b \in fn(Q)$.

13. Reti di petri

È un modello distribuito non-interleaving. Si ha:

- uno stato globale, che è un multinsieme di stati locali;
- transizioni che coinvolgono solo qualche stato locale;
- token game;
- parallelismo esplicito, che è diverso da LTS, sia per modellazione che equivalenza comportamentale.



Si usa la struttura del **multiset**. Definito come M su insieme A è una lista di elementi possibilmente infinita e non ordinata di A dove nessun elemento può ricorrere un numero infinito di volte.

Ad esempio, $M = \{a, \tau, a, \tau, \tau\}$ è un multiset sull'insieme $A = \{a, \tau\}$.

Preso un insieme contabile S , un multiset finito su S è una funzione $m : S \rightarrow \mathbb{N}$ t.c. $\text{dom}(m) = \{s \in S \mid m(s) \neq 0\}$ è finito. La multiplicità di s in m è data dal numero $m(s)$.

L'insieme di tutti i multiset finiti su S , denotato come $M_{\text{fin}}(S)$ è iterato su m e indicizzato.

Un multiset m con $\text{dom}(m) = \emptyset$ è detto vuoto.

Si scrive $m \subseteq m'$ se $m(s) \leq m'(s)$ per tutti $s \in S$.

Si scrive $m \subset m'$ se $m(s) < m'(s)$ per qualche $s \in S$.

L'unione di multiset è $(m \oplus m')(s) = m(s) + m'(s)$ ed è un operatore commutativo, associativo e con elemento neutrale.

Se $m_2 \subseteq m_1$ allora la differenza è denotata come $(m_1 \ominus m_2)(s) = m_1(s) - m_2(s)$.

Il prodotto scalare di un numero j è definito come $(j \cdot m)(s) = j \cdot (m(s))$.

13.1. Reti di Petri P/T

Una rete di Petri Place/Transition è una tupla $N = (S, A, T)$ dove

- S è insieme contabile di posti iterati su s (possibilmente indicizzati).
- $A \subseteq \text{Lab}$ è insieme contabile di etichette iterate su l (possibilmente indicizzate).
- $T \subseteq (\mathcal{M}_{\text{fin}}(S) \setminus \{\emptyset\}) \times A \times \mathcal{M}_{\text{fin}}(S)$ è insieme contabile di transizioni iterate su t (possibilmente indicizzate) t.c. per ogni $l \in A$ esiste una transizione $t \in T$ della forma (m, l, m') . Di esso possiamo denotare:
 - t^* come pre-set m di token da consumare (non può essere un multiset vuoto).
 - $l(t)$ per la sua label l .
 - t^* come post-set m' di token da produrre.

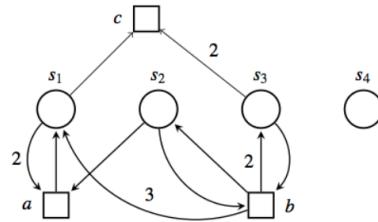
Una transizione t può essere anche rappresentata come $t^* \xrightarrow{l(t)} t^*$.

Un pre-set può essere definito come $t^* = \{t \in T \mid s \in t^*\}$.

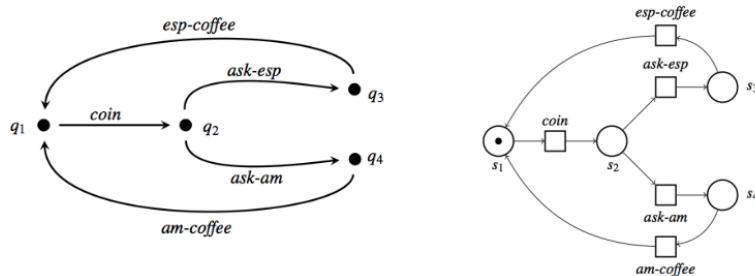
Un post-set può essere definito come $t^* = \{t \in T \mid s \in t^*\}$.

Esempio

$N = (S, A, T)$ con $S = \{s_1, s_2, s_3, s_4\}$, $A = \{a, b, c\}$ e $T = \{(2 \cdot s_1 \oplus s_2, a, s_1), (s_2 \oplus s_3, b, 3 \cdot s_1 \oplus s_2 \oplus 2 \cdot s_3), (s_1 \oplus 2 \cdot s_3, c, \emptyset)\}$.



Gli LTS sono una sottoclasse di rete P/T. Infatti si può rappresentare l'LTS della macchina del caffè in modo semplice attraverso P/T.



13.2. Sistemi di reti P/T, marcatori e token game

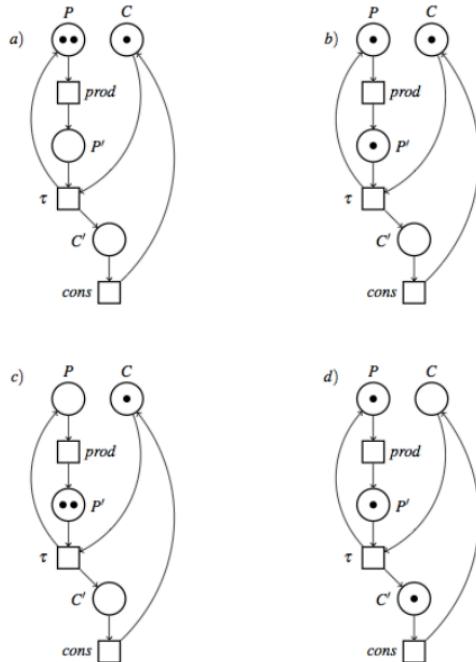
Dato un marcitore m e un posto s , il posto s contiene $m(s)$ token (ed è graficamente rappresentato da $m(s)$ punti dentro il posto s). Un sistema rete P/T $N(m_0)$ è una tupla (S, A, T, m_0) in cui m_0 è una marcatura su S , chiamato “marcatore iniziale”. $N(m_0)$ è altresì chiamata rete marcata.

Da una rete P/T $N = (S, A, T)$ una transizione t è abilitata dal marcitore m e denotata come $m[t]$ se $\bullet t \subseteq m$. L'esecuzione di t abilitata da m a produrre il marcitore $m' = (m \ominus \bullet t \oplus t^\bullet)$ e scritto come $m[t]m'$. Nessun token aggiuntivo può disabilitare una transizione.

Le reti di petri P/T sono permissive perché se t è abilitata da m , allora t è abilitata anche da ogni altro marcitore m' che copre m , tipo un m' t.c. $m \subseteq m'$.

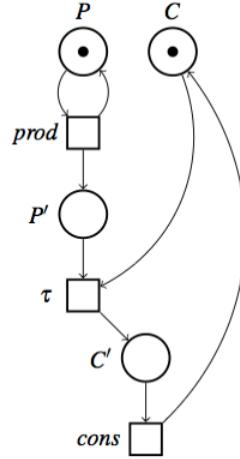
Esempio

2 produttori 1 consumatore



Esempio

Non specificato numero di produttori e consumatori



L'insieme dei marcatori raggiungibili da m in un sistema di reti P/T $N(m_0)$, denotato come $[m]$ è definito dall'insieme t.c.:

- $m \in [m]$
- se $m_1 \in [m]$ e per una transizione $t \in T$, $m_1[t]m_2$ allora $m_2 \in [m]$.

m è *raggiungibile* se m è raggiungibile dal marcatore iniziale m_0 .

Una sequenza firing che parte da m è definita come:

- m è una sequenza firing.
- se $m_1[t_1]m_2...[t_{n-1}]m_n$ con $m = m_1 \wedge n \geq 1$ è una sequenza firing e $m_n[t_n]m_{n+1}$ allora $m = m_1[t_1]m_2...[t_{n-1}]m_n[t_n]m_{n+1}$ è una sequenza firing.

In genere $m = m_1[t_1]m_2...[t_n]m_{n+1}$ è abbreviata $m[t_1...t_n]m_{n+1}$ e $t_1...t_n$ è chiamata *sequenza di transizione* che parte da m e finisce in m_{n+1} .

13.3. Classi di reti P/T

Una rete P/T $N = (S, A, T)$ può essere

- **staticamente aciclica** se non esiste sequenza $x_1x_2...x_n$ t.c. $n \geq 3$, $x_i \in S \cup T$ per $i = 1, \dots, n$, $x_1 = x_n$, $x_1 \in S \wedge x_i \in {}^\bullet x_{i+1}$ per $i = 1, \dots, n-1$.
- **distinta** se tutte le transizioni hanno etichette distinte: per ogni $t_1, t_2 \in T$ se $l(t_1) = l(t_2)$ allora $t_1 = t_2$.
- **finite** se S e T sono insiemi finiti.
- **macchina a stati finiti (FSM)** se N è finita se per tutti $t \in T$, $|{}^\bullet t| = 1 \wedge |t^\bullet| \leq 1$.
- **rete BPP** se N è finito e ogni transizione ha esattamente un posto input.
- **rete CCS** se per ogni $t \in T$, $1 \leq |{}^\bullet t| \leq 2 \wedge$ se $|{}^\bullet t| = 2$ allora $l(t) = \tau$.

13.4. Classi di sistemi di reti P/T

Un sistema di rete P/T $N(m_0)$ è

- **dinamicamente aciclico** se non esiste $m_1 \in [m_0]$ con una sequenza firing non vuota $m_1[t_1]m_2...[t_{n_1}]m_n$ t.c. $m_1 \subseteq m_n$.
- **FSM sequenziale** se N è una FSM e m_0 è singolo, ovvero $|m_0| = 1$.
- **FSM concorrente** se N è una FSM e m_0 è arbitrario.

- **k-bounded** se ogni posto contiene al massimo k token in ogni marcatore raggiungibile. $\forall s \in S \quad m(s) \leq k$ per ogni $m \in [m_0]$.
- **sicuro** se è 1-bounded.
- **bounded** se $\forall s \in S \exists k \in \mathbb{N}$ t.c. $m(s) \leq k$ per ogni $m \in [m_0]$.

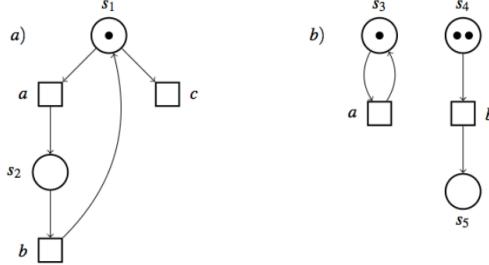


Figura 78: Una FSM sequenziale e una concorrente

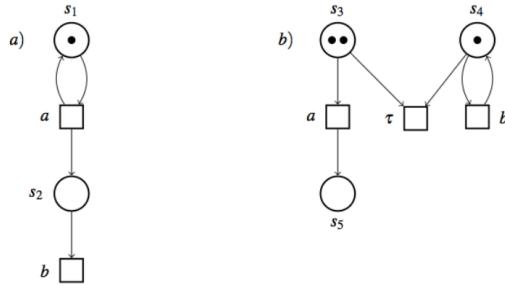


Figura 79: Una rete BPP e una CCS

Dato un sistema di reti P/T $N(m_0)$ allora si ha:

1. Se N è una rete FSM, allora N è rete BPP.
2. Se N è una rete BPP, allora N è rete CCS finita.
3. Se $N(m_0)$ è una FSM sequenziale, allora $N(m_0)$ è FSM concorrente.
4. Se $N(m_0)$ è una FSM sequenziale, allora $N(m_0)$ è sicura.
5. Se $N(m_0)$ è una FSM concorrente, allora $N(m_0)$ è $|m_0|$ -bounded.
6. Se $N(m_0)$ è una finita e bounded, allora $N(m_0)$ è k -bounded per qualche $k \in \mathbb{N}$.

Dimostrazione La boundness implica che per ogni $s \in S$ esiste un upper-bound k_s sul numero dei token che può essere accumulato su s . Se la rete è finita, questo è sufficiente per scegliere il più grande k_s (o k') avente proprietà che per ogni s , $k_s \leq k'$ in modo tale che la rete sia k' -bounded. Ogni rete bounded che non è k -bounded per ogni k è infinita. Ad esempio, $N(m_0) = (S, A, T, m_0)$ dove $S = \{s_i \mid i \in \mathbb{N}\}$, $A = \{a_i \mid i \in \mathbb{N}\}$, $T = \{(s_i, a_i, 2 \cdot S_{i+1}) \mid i \in \mathbb{N}\}$, $m_0 = \{s_0\}$ è una rete infinita t.c. il posto s_i tiene fino a 2^i token; quindi $N(m_0)$ è bounded, ma $\nexists k$ t.c. $2^i \leq k \quad \forall i \in \mathbb{N}$. ■

7. Se $N(m_0)$ è una finita e bounded, allora l'insieme $[m_0]$ dei marcatori raggiungibili da m_0 è finito.
8. Se N è staticamente aciclico, allora $N(m_0)$ è dinamicamente aciclico.
9. Se $N(m_0)$ è finito e dinamicamente aciclico, allora l'insieme delle sequenze firing è finito.

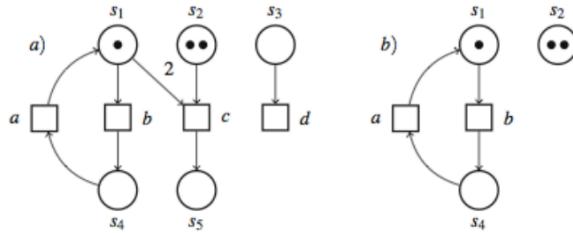
13.5. Sottorete dinamicamente raggiungibile

Dato un sistema di rete P/T $N(m_0) = (S, A, T, m_0)$ la sottorete dinamicamente raggiungibile è $\text{Net}_d(N(m_0)) = (S', A', T', m_0)$ dove

- $S' = \{s \in S \mid \exists m \in [m_0] \text{ t.c. } m(s) \geq 1\}$

- $T' = \{t \in T \mid \exists m \in [m_0] \text{ t.c. } m[t]\}$
- $A' = \{l \mid \exists t \in T' \text{ t.c. } l(t) = l\}$

Esempio



13.5.1. Sottorete dinamicamente ridotta

Un sistema di rete P/T $N(m_0) = (S, A, T, m_0)$ è dinamicamente ridotto se $N(m_0) = \text{Net}_d(N(m_0))$.

13.6. Sottorete staticamente raggiungibile

Dato un sistema di rete P/T $N(m_0) = (S, A, T, m_0)$ diciamo che una transizione t è staticamente abilitata da un insieme di posti $S' \subseteq S$ e si denota da $S' \llbracket t \rrbracket$ se $\text{dom}(\bullet t) \subseteq S'$.

Dati due insiemi di posti $S_1, S_2 \subseteq S$ diciamo che S_2 è staticamente raggiungibile in un passo da S_1 se esiste una transizione $t \in T$ t.c. $S_1 \llbracket t \rrbracket, \text{dom}(t^\bullet) \not\subseteq S_1 \wedge S_2 = S_1 \cup \text{dom}(t^\bullet)$, denotato $S_1 \xrightarrow{t} S_2$. La relazione di raggiungibilità statica $\Rightarrow^* \subseteq \mathcal{P}_{\text{fin}}(S) \times \mathcal{P}_{\text{fin}}(S)$ è la relazione t.c.

- $S_1 \Rightarrow^* S_1$
- Se $S_1 \Rightarrow^* S_2 \wedge S_2 \xrightarrow{\tau} S_3$ allora $S_1 \Rightarrow^* S_3$.

L'insieme dei posti $S_k \subseteq S$ è il più grande insieme staticamente raggiungibile da S_1 se $S_1 \Rightarrow^* S_k \wedge \forall t \in T \text{ t.c. } S_k \llbracket t \rrbracket$ si ha $\text{dom}(t^\bullet) \subseteq S_k$.

Invece, preso un sistema rete P/T $N(m_0)$, il più grande insieme di posti staticamente raggiungibile da $\text{dom}(m_0)$ è $\llbracket \text{dom}(m_0) \rrbracket$.

La sottorete staticamente raggiungibile $\text{Net}_{s(N(m_0))}$ è la rete (S', A', T', m_0) dove

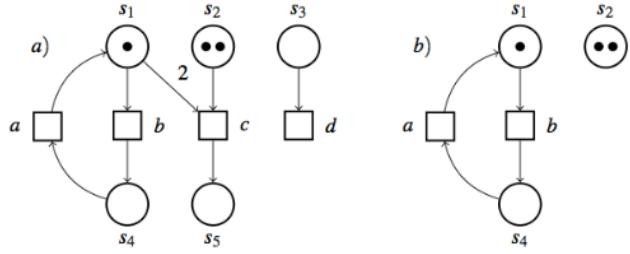
- $S' = \llbracket \text{dom}(m_0) \rrbracket$
- $T' = \{t \in T \mid S' \llbracket t \rrbracket\}$
- $A' = \{l \mid \exists t \in T' \text{ t.c. } l(t) = l\}$

Un sistema rete P/T è staticamente ridotto se $\text{Net}_s(N(m_0)) = N(m_0)$ (il sistema di rete è uguale alla sua sottorete staticamente raggiungibile).

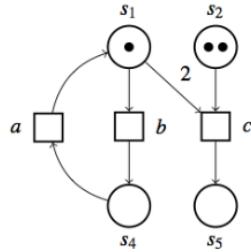
$\text{Net}_s(N(m_0))$ è algoritmamente computabile per ogni sistema di rete P/T in tempo polinomiale.

Esempio

Un sistema di rete P/T con sua sottorete dinamicamente raggiungibile



e la sottorete staticamente raggiungibile di (a).



Se $N(m_0)$ è dinamicamente riducibile, è anche staticamente riducibile.

Se $N(m_0)$ è dinamicamente raggiungibile da sottorete $\text{Net}_d(N(m_0)) = (S', A', T', m_0)$ la quale staticamente raggiungibile da sottorete $\text{Net}_s(N(m_0)) = (S'', A'', T'', m_0)$, allora $S' \subseteq S''$, $T' \subseteq T''$, $A' \subseteq A''$.

Se $N(m_0)$ è rete BPP che riduce staticamente, allora riduce anche dinamicamente.

13.7. Proprietà decidibili

- **Computabilità** di Net_d e Net_s per ogni sistema di rete finito P/T $N(m_0)$.
- **Raggiungibilità** di un dato marcitore con quello iniziale di un sistema finito P/T.
- **Deadlock**, ovvero raggiungere un marcitore che non abilita transizioni.
- **Liveness**: una transazione t è live se per ogni marcitore m raggiungibile da m_0 esiste un marcitore m' raggiungibile da m t.c. t è abilitato a m' . Il sistema finito di rete P/T $N(m_0)$ è live se ogni sua transizione è live.

13.8. Isomorfismo

Date due reti P/T $N_1 = (S_1, A, T_1)$ e $N_2 = (S_2, A, T_2)$ diciamo che esse sono isomorfiche e scriviamo $N_1 \cong N_2$ se esiste una biiezione $f : S_1 \rightarrow S_2$ esterna ai marcatori (dunque su tutti i suoi componenti) t.c. $(m, l, m') \in T_1 \iff (f(m), l, f(m')) \in T_2$.

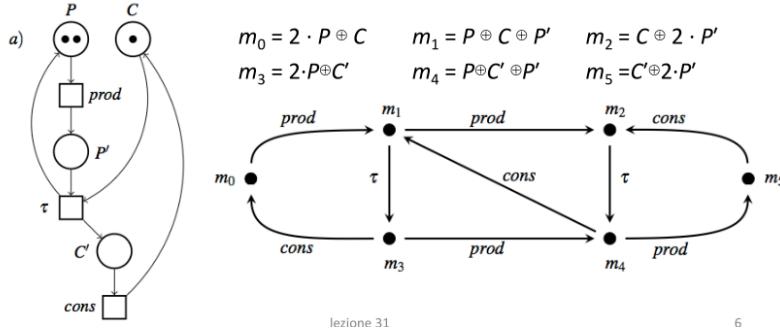
Due sistemi di reti P/T $N_1(m_1)$ e $N_2(m_2)$ sono rooted isomorfiche e scriviamo $N_1(m_1) \cong_r N_2(m_2)$ se l'isomorfismo $f : S_1 \rightarrow S_2$ assicura che $f(m_1) = m_2$.

Una rete isomorfica è spesso troppo concreta come relazione di equivalenza.

Capire se due sistemi di reti finite sono rooted isomorfici è decidibile in classe NP.

13.9. Interleaving marking graph

L'interleaving marking graph di $N(m_0) = (S, A, T, m_0)$ è il rooted LTS $\text{IMG}(N(m_0)) = ([m_0], A, \rightarrow, m_0)$ dove m_0 è lo stato iniziale, l'insieme degli stati è dato da $[m_0]$ e la transizione $\rightarrow \subseteq \mathcal{M}_{\text{fin}}(S) \times A \times \mathcal{M}_{\text{fin}}(S)$ è definito da $m \rightarrow m' \iff \exists t \in T \text{ t.c. } m[t]m' \wedge l(t) = a$.



lezione 31

6

- Se $N(m_0)$ è finito e bounded, allora $\text{IMG}(N(m_0))$ è LTS finite-state.
- Se $N(m_0)$ è finito ma non bounded, allora $\text{IMG}(N(m_0))$ non è finite-state.
- Se $N(m_0)$ e $\text{IMG}(N(m_0))$ non è finite-state, allora N è infinito.
- Se $\text{IMG}(N(m_0))$ è finite-state, allora la sottorete dinamicamente raggiungibile $\text{Net}_d(N(m_0))$ è finita e bounded.
- Se $N(m_0)$ è distinta, allora $\text{IMG}(N(m_0))$ è deterministica.

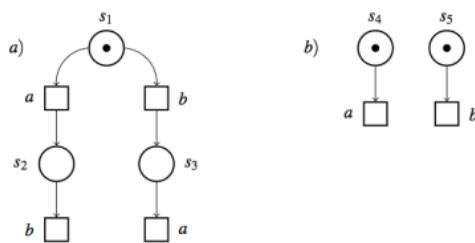
13.9.1. Equivalenza a tracce

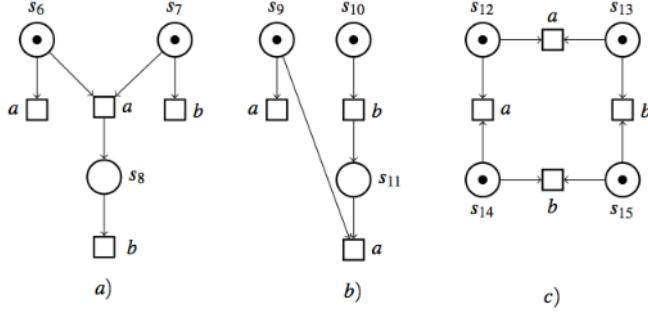
Dato il sistema P/T $N(m_0) = (S, A, T, m_0)$ l'insieme delle tracce $N(m_0)$ è l'insieme $Tr(\text{IMG}(N(m_0))) = \{\sigma \in A^* \mid \exists m \in [m_0]. m_0 \xrightarrow{\sigma}^* m\}$.

Si può abusare la notazione scrivendo semplicemente $Tr(N(m_0))$. L'equivalenza per due sistemi $N_1(m_1)$ e $N_2(m_2)$ è chiaramente verificata se $Tr(N_1(m_1)) = Tr(N_2(m_2))$.

Si può estendere la definizione come $Tr(m) = \{\sigma \in A^* \mid \exists m' \in [m]. \xrightarrow{\sigma}^* m'\}$ così da vedere su due marcatori $m_1, m_2 \in [m_0]$.

Esempio





13.10. Linguaggio rete Petri

Preso un sistema P/T $N(m_0) = (S, A, T, m_0)$ t.c. $B = A \setminus \{\tau\}$ si scrive l'insieme delle equivalenze a tracce deboli come $WCTr(\text{IMG}(N(m_0))) = \left\{ \sigma \in B^* \mid \exists m \in [m_0]. m_0 \xrightarrow{\sigma} m \Rightarrow \forall l \in B \right\}$.

Il linguaggio per la rete Petri $L[N(m_0)]$ è esattamente quello sopra ed include tutti i linguaggi nella classe di quelli dipendenti dal contesto ma non tutti quelli liberi dal contesto (e.g. non contiene ww^R).

13.11. Interleaving bisimilarità

Due sistemi P/T $N_1(m_1)$ e $N_2(m_2)$ sono bisimili interleaving $N_1(m_1) \sim N_2(m_2)$ oppure $m_1 \sim m_2 \iff \exists R \subseteq [m_1] \times [m_2]$ su $\text{IMG}(N_1(m_1))$ e $\text{IMG}(N_2(m_2))$ t.c. $(m_1, m_2) \in R$.

Preso $N(m_0)$ e un LTS rooted $TS(q_0) = (Q, A, \rightarrow, q_0)$ diciamo che N e TS sono bisimili interleaving $N(m_0) \sim TS(q_0)$ oppure $m_0 \sim q_0 \iff \exists R \subseteq [m_0] \times Q$ su $\text{IMG}(N(m_0))$ e $TS(q_0)$ t.c. $(m_0, q_0) \in R$.

Questa bisimilarità coincide con l'equivalenza a tracce su reti P/T distinte perché esse generano IMG deterministici. La bisimilarità e l'equivalenza a tracce per reti P/T finite sono indecidibili. Sono decidibili per reti P/T bounded finite.

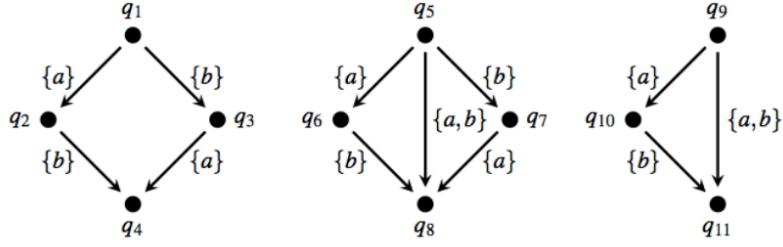
13.12. Step Transition System

Preso insieme Lab di etichette, l'insieme SLab = $\mathcal{M}_{\text{fin}}(\text{Lab}) \setminus \{\emptyset\}$ è l'insieme di etichette step.

Uno step transition system è STS = $(Q, \mathcal{B}, \rightarrow)$ dove

- $Q \neq \emptyset$ è l'insieme contabile di stati iterati su q (possibilmente indicizzato).
- $\mathcal{B} \subseteq \text{SLab}$ è l'insieme contabile di etichette step iterate su M (possibilmente indicizzato).
- $\rightarrow \subseteq Q \times \mathcal{B} \times Q$ è la relazione di transizione.

Esempio



13.12.1. Fully concurrent STS

Un STS = $(Q, \mathcal{B}, \rightarrow)$ è fully concurrent se quando $q \xrightarrow{M_1 \oplus M_2} q'$ con $M_i \neq \emptyset$ per $i = 1, 2$ allora $\exists q_1, q_2$ t.c. $q \rightarrow q_1 \rightarrow q' \wedge q \rightarrow q_2 \rightarrow q'$.

Dunque, un STS è fully concurrent t.c $q \xrightarrow{M} q'$ con $M = \{l_1, l_2, \dots, l_n\}$ e poi per ogni linearizzazione l'_1, l'_2, \dots, l'_n di step M esiste un percorso $q_1 \rightarrow q_2 \rightarrow \dots \rightarrow q_n \rightarrow q_{n+1}$ dove $q_1 = q, q_{n+1} = q'$.
[salto la roba su step marking graph e step bisimilarity]

13.13. Reti di Petri non permissive NP/T

La non permissività disabilita una transizione aggiungendo roba. Una transizione t abilitata da m può essere disabilitata da un marcitore più grande m' perché quest'ultimo può contenere token aggiuntivi.

Un posto testabile è un posto che può essere testato per un preciso numero di token in esso.

Una transizione neg-set è un insieme di posti usati per testare l'assenza di token addizionali, a parte quelli obbligatori presi dal pre-set.

Nella condizione di abilitazione tutti i token di pre-set devono essere disponibili con nessun altro token aggiuntivo presente nei posti neg-set.

Una rete Petri NP/T è una tupla $N = (S, D, A, T)$ dove

- S è insieme contabile di posti, iterati su s (possibilmente indicizzati).
- $D \subseteq S$ è l'insieme dei posti testabili.
- $A \subseteq \text{Lab}$ è l'insieme delle etichette contabile, iterate su l (possibilmente indicizzati).
- $T \subseteq (\mathcal{M}_{\text{fin}}(S) \setminus \{0\}) \times \mathcal{P}_{\text{fin}}(D) \times A \times \mathcal{M}_{\text{fin}}(S)$ è l'insieme delle transizioni, iterate su t (possibilmente indicizzati) t.c. per ogni $l \in A \exists t = (m, I, l, m') \in T$.

Preso $t = (m, I, l, m')$ si usa la notazione $(\bullet t, {}^o t) \xrightarrow{l(t)} t^\bullet$:

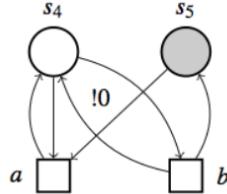
- $\bullet t$ è il pre-set non vuoto m di token da consumare.
- ${}^o t$ è il neg-set non vuoto I di posti testabili in cui non possono essere presenti ulteriori token oltre a quelli richiesti da $\bullet t$.
- $l(t)$ per l'etichetta l .
- t^\bullet è il post-set m' di token da produrre.

Una transizione t è abilitata in m , e scritto come $m[t]$, se $\bullet t(s) \leq m(s) \quad \forall s \in \text{dom}(\bullet t)$ e, in aggiunta, $\bullet t(s) = m(s) \quad \forall s \in {}^o t$. Questo produce un marcitore $m' = (m \ominus \bullet t) \oplus t^\bullet$, e scritto come $m[t]m'$.

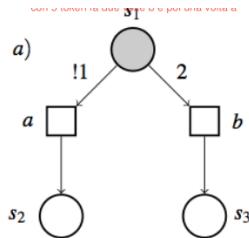
Per convenzione un posto testabile è segnato grigio; un arco tra un posto testabile ad una transizione può avere un numero, anche con prefisso $!$; se $s \in \bullet t$, $s \notin {}^o t$; se $s \in {}^o t \cap \text{dom}(\bullet t)$ allora l'arco tra s e t è etichettato come $!\bullet t(s)$; se $s \in {}^o t \wedge s \notin \text{dom}(\bullet t)$ allora vi è un arco $!0$ da s a t (= nessun token può essere presente in s nel marcatore corrente m per abilitare t).

Esempio

Eseguita una b non si potrà più fare a perché ci sarà un token in s_5 .



Tipo una divisione per 2: con 4 token fa due volte b , con 5 token fa due volte b e poi una volta a .



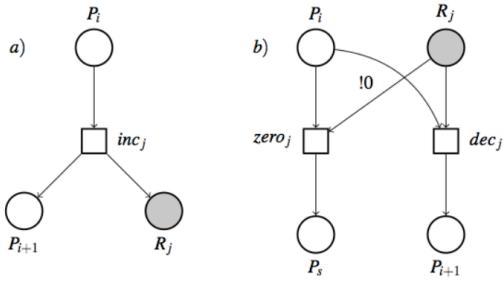
13.14. Turing completezza

Si fa riferimento solo ai NP/T finite. Dunque si può costruire una counter machine $M = (I, n)$ dove $I = \{(1 : I_1), \dots, (m : I_m)\}$ è l'insieme delle istruzioni indicizzabili in M con $|I| = m$, n numero dei registri r_j in M e ogni istruzione I_i come:

- $I_i = \text{Inc}(r_j)$ per $1 \leq j \leq n$ che incrementa r_j e va all'istruzione $i + 1$.
- $I_i = \text{DecJump}(r_j, s)$ per $1 \leq j \leq n$ che se $r_j = 0$ allora salta all'istruzione s , sennò decremente r_j e va a $i + 1$.

Preso dunque $N(m_0) = (S, D, A, T, m_0)$ come

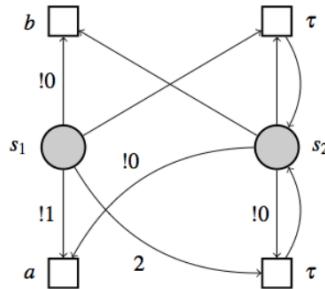
- $S = \{P_1, P_2, \dots, P_m, P_{\text{last}}, R_1, R_2, \dots, R_n\}$ in cui P_i si riferisce al program-counter/istruzione I_i , P_{last} modella il caso in cui la prossima istruzione ha indice $> m$, R_j per ogni registro/contatore r_j .
- $D = \{R_1, R_2, \dots, R_n\}$.
- $A = \{\text{inc}_j, \text{dec}_j, \text{zero}_j \mid 1 \leq j \leq n\}$



13.15. Problema dell'ultimo uomo alzato

Inizialmente nessun token è presente in s_2 . Se nessun token è in s_1 , allora si può eseguire solo a. Se $n > 1$ token sono presenti in s_1 allora solo b potrà essere eseguito: si rimuovono due token da s_1 , si produce un token in s_2 , un token alla volta è consumato da s_1 e quando s_1 è vuoto, allora b potrà essere eseguito da s_2 .

È dinamicamente aciclico perché finisce sempre.



CCS non può risolvere questo problema.