Lezione 13 MSC
Finitary CCS è Turing completo

Roberto Gorrieri

Turing-completezza

- Un formalismo è Turing-completo quando i programmi in quel formalismo possono calcolare tutte le funzioni calcolabili con macchine di Turing.
- Oltre alle macchine di Turing, altri formalismi Turing-completi sono il lambda-calcolo, le Counter Machines (CM), ed altri ancora (linguaggi di programmazione, purchè la memoria disponibile sia infinita).
- Mostreremo come una qualunque CM possa essere modellata fedelmente da un processo di Finitary CCS, cioè pure Finitary CCS è Turing-completo.

Indecidibilità delle equivalenze

- Conseguenza della Turing-completezza è che le equivalenze comportamentali sono indecidibili per Finitary CCS.
- Se, per assurdo, fossero decidibili, allora saremmo in grado di risolvere il problema della fermata (Halting Problem), un problema che è indecidibile per formalismi Turing-completi.

Halting Problem

- Supponiamo di avere una enumerazione dei programmi nel formalismo P_1, P_2, P_3, …
- halt(x,y) = 1 se $P_x(y)$ termina, 0 altrimenti
- Se halt fosse calcolabile, allora anche K(x) = halt(x, x) sarebbe calcolabile.
- Ma allora anche G(x) = 1 se K(x) = 0, G(x) indefinito se K(x) = 1, dovrebbe essere calcolabile. Supponiamo che P_j calcoli G.
- Ora se G(j)=1, allora vuol dire che K(j)=0, ma allora halt(j,j)=0, ovvero $P_j(j)$ diverge: contraddizione!
- Similmente, se G(j) è indefinito, allora vuol dire che K(j) = 1, ovvero che halt(j,j) = 1, ovvero che $P_j(j)$ termina: contraddizione!
- Dato che l'unica assunzione che abbiamo fatto è che halt sia calcolabile, dobbiamo concludere che halt non è calcolabile.

Counter Machines

- Usa registri/contatori dove memorizzare valori interi non negativi (cioè naturali): $r_1, \ldots, r_n$
- Inizialmente vengono messi valori $v_1, \ldots, v_n$ nei registri (se non specificato, il registro è inizializzato a 0)
- Il programma è un insieme indicizzato di istruzioni
  $$\{(1,I_1), \ldots (m,I_m)\}$$
- Si inizia dall'istruzione di indice 1, si procede alla successiva (a meno di istruzioni di salto) e si termina solo quando si arriva ad un indice non presente, tipicamente più grande di m.
- Se il programma termina, il risultato si trova nei registri specificati come output.

Counter Machines (2)

- Una classe speciale di CM (anch'essa Turing-completa) usa due soli tipi di operazioni, dove assumiamo che $1 \leq i \leq m$ e $1 \leq j \leq n$:
  - (i, Inc(r_j)) : incrementa il registro r_j e poi passa all'istruzione i+1
  - (i, DecJump(r_j,s)) : se il valore in r_j non è zero, allora decrementa r_j e poi passa all'istruzione i+1, altrimenti salta all'istruzione di indice s (dove s può essere maggiore di m).

Esempio: somma di due numeri

- Semplice programma che calcola la somma dei valori contenuti nei registri r_1 e r_2 mettendo il risultato in r_1
- Usa 3 registri, dove il registro r_3 è a zero
- Usa 3 istruzioni in sequenza:

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

- Si comincia dalla prima istruzione, decrementando r_2, poi incrementando r_1, quindi saltando incondizionatamente alla prima istruzione, terminando quando r_2 è zero (l'istruzione 4 non esiste).

CM - definizione

- Lo stato interno della CM M = (I, n), dove I = {(1,I_1), …(m,I_m)} e n è il numero dei registri, è dato dalla n+1-tupla (i, v_1, --, v_n). Lo stato iniziale ha i = 1 e i vari v_i sono quelli forniti in input.
- La transizione (i, v_1, --, v_n) $\rightarrow_M$ (i', v'_1, --, v'_n) avviene se:

- $I_i = Inc(r_j)$ and $i' = i+1$, $v'_j = v_j + 1$, $v'_p = v_p$ for any $p \neq j$; or
- $I_i = DecJump(r_j, s)$, $v_j > 0$ and $i' = i+1$, $v'_j = v_j - 1$, $v'_p = v_p$ for any $p \neq j$; or
- $I_i = DecJump(r_j, s)$, $v_j = 0$ and $i' = s$, $v'_p = v_p$ for any $p = 1, \ldots, n$.

- N.B: la macchina è deterministica.

CM – definizione (2)

- Dati gli input v_1, --, v_n, per la CM M = (I, n), lo stato interno (i, v'_1, --, v'_n) è detto terminale se (1, v_1, --, v_n) $\rightarrow^*_M$ (i, v'_1, --, v'_n) con i > m. I valori v'_1, --, v'_n sono detti gli output.
- Se da (1, v_1, --, v_n) non si raggiunge nessuna configurazione terminale, allora diciamo che la macchina, per quegli input, diverge.
- La funzione calcolata da M è

  $f_M(v_1, \ldots, v_n) = (v'_1, \ldots, v'_n)$ se la computazione termina in (i, v'_1, --, v'_n) con i > m, indefinita altrimenti.

Exercise

- Consider the CM *M* defined as

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

- Compute the finite set of the configurations reachable from the initial one (1,3,2,0), i.e., when register r_1 holds value 3, r_2 holds value 2 and r_3 is 0.
- What is the partial function $f_M(v1,v2,v3)$ computed by *M*?

Classe universale minimale di CM

- 3CM = counter machines che usano tre contatori, classe universale (ovvero Turing-completa)
- 2CM = counter machines che usano solo 2 contatori, pure universale, ma gli argomenti in input (così come gli output) vanno opportunamente codificati via godelizzazione.

Allora useremo, per semplicità, le 3CM come nostro formalismo universale.

CM in CCS

- Data una CM M = (I,3), con input v_1, v_2, v_3, definiamo un processo CCS $CM_{M(v1,\, v2,\, v3)}$ che modella fedelmente il comportamento della CM M.

$$CM_{M(v_1,v_2,v_3)} \stackrel{def}{=} (\nu L)(P_1 \mid \ldots \mid P_m \mid R_1 \mid R_2 \mid R_3 \mid B_{(v_1,v_2,v_3)})$$

dove P_i sono le costanti che definiscono le istruzioni I_i, R_j sono le costanti che definiscono i registri (contatori) r_j, B è il programmino di bootstrapping che inizializza i registri e attiva la prima istruzione e L è l'insieme delle azioni inc_j, $zero_j$, dec_j $1 \leq j \leq 3$ e p_i per ogni i che occorre in qualche istruzione (tipicamente $1 \leq i \leq m+1$)

CM in CCS (2)

- L'istruzione (i, Inc(r_j)) è definita con una costante ricorsiva:

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

  dove p_i è l'azione che accetta l'abilitazione dell'istruzione, inc_j corrisponde all'incremento operato sul registro r_j, e 'p_{i+1} abilita la successiva istruzione di indice i+1.
- L'istruzione (i, DecJump(r_j,s)) viene modellata come

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

  dove la scelta tra $zero_j$ e dec_j è guidata dallo stato del registro: se il registro r_j è a 0 allora solo la sincronizzazione su zero è possibile (e l'istruzione attivata è quella di indice s), mentre se il registro r_j non è a 0, allora solo la sincronizzazione su dec_j è possibile (con decremento di r_j e attivazione dell'istruzione i+1).

CM in CCS (3)

- Ogni registro r_j viene modellato da un contatore R_j:

$$R_j \stackrel{def}{=} zero_j.R_j + inc_j.((\nu a)(R_{j_1} \mid a.R_j))$$
$$R_{j_1} \stackrel{def}{=} dec_j.\bar{a}.\mathbf{0} + inc_j.((\nu b)(R_{j_2} \mid b.R_{j_1}))$$
$$R_{j_2} \stackrel{def}{=} dec_j.\bar{b}.\mathbf{0} + inc_j.((\nu a)(R_{j_1} \mid a.R_{j_2}))$$

- Se assumiamo che gli input siano v_1, --, v_n allora il processo B di bootstrapping è definito come:

$$B_{(v_1,\ldots,v_n)} \stackrel{def}{=} \underbrace{\overline{inc}_1.\cdots.\overline{inc}_1}_{v_1 \text{ times}}.\ldots.\underbrace{\overline{inc}_n.\cdots.\overline{inc}_n}_{v_n \text{ times}}.\overline{p}_1.\mathbf{0}$$

Example 3.22. Consider the CM M of Example 3.21 and Exercise 3.68. The process $CM_{M(3,2,0)}$ is:

$$CM_{M(3,2,0)} \stackrel{def}{=} (\nu L)(P_1 \,|\, P_2 \,|\, P_3 \,|\, R_1 \,|\, R_2 \,|\, R_3 \,|\, B_{(3,2,0)})$$

where:

- $P_1 \stackrel{def}{=} p_1.P_1' \qquad P_1' \stackrel{def}{=} \overline{zero}_2.\overline{p}_4.P_1 + \overline{dec}_2.\overline{p}_2.P_1$
- $P_2 \stackrel{def}{=} p_2.P_2' \qquad P_2' \stackrel{def}{=} \overline{inc}_1.\overline{p}_3.P_2$
- $P_3 \stackrel{def}{=} p_3.P_3' \qquad P_3' \stackrel{def}{=} \overline{zero}_3.\overline{p}_1.P_3 + \overline{dec}_3.\overline{p}_4.P_3$
- $B_{(3,2,0)} \stackrel{def}{=} \overline{inc}_1.\overline{inc}_1.\overline{inc}_1.\overline{inc}_2.\overline{inc}_2.\overline{p}_1.\mathbf{0}$
- $L = \{inc_j, zero_j, dec_j \mid 1 \leq j \leq 3\} \cup \{p_i \mid 1 \leq i \leq 4\}$.

By performing the bootstrapping, $CM_{M(3,2,0)}$ reaches the state

$$(\nu L)(P_1' \,|\, P_2 \,|\, P_3 \,|\, R_1' \,|\, R_2' \,|\, R_3 \,|\, \mathbf{0}),$$

which represents the CCS process for the CM M ready to execute the first instruction. R_1' stands for $(\nu a)((\nu b)((\nu a)(R_{1_1} \,|\, a.R_{1_2}) \,|\, b.R_{1_1}) \,|\, a.R_1)$, while R_2' stands for $(\nu a)((\nu b)(R_{2_2} \,|\, b.R_{2_1}) \,|\, a.R_2)$. □

CM in CCS (4)

For $i = 1, \ldots, m$, let $\langle CM_{(i,v_1,v_2,v_3)} \rangle$ be the set of all the terms of the form

$$(\nu L)(P_1 \,|\, \ldots \,|\, P_{i-1} \,|\, P'_i \,|\, P_{i+1} \,|\, \ldots \,|\, P_m \,|\, R'_1 \,|\, R'_2 \,|\, R'_3 \,|\, \mathbf{0})$$

where for $j = 1,2,3$, $R'_j \approx Counter_{v_j}$ and R'_j cannot perform τ initially, i.e., $R'_j \not\xrightarrow{\tau}$. It is not difficult to see that if $Q, Q' \in \langle CM_{(i,v_1,v_2,v_3)} \rangle$ then $Q \sim Q'$.

- Lo stato iniziale $(1, v_1, \ldots, v_n)$ della CM M corrisponde ad un processo CCS in <$CM_{(1,v1,\ldots,vn)}$>
- Ogni cambio di stato della CM M, e.g.,

  $$(i, v_1, \text{--}, v_n) \rightarrow_M (i', v'_1, \text{--}, v'_n)$$

  determina una sequenza di sincronizzazioni che portano da un processo in <$CM_{(i,v1,\ldots,vn)}$)> ad un processo in <$CM_{(i',v'1,\ldots,v'n)}$ >

CM in CCS (5)

- L'encoding della CM M in CCS con CM_M è corretto:

Proposition 3.8. *Given a CM M with inputs* v_1, v_2, v_3, *let* $CM_{M(v_1,v_2,v_3)}$ *be the CCS process defined above, such that* $CM_{M(v_1,v_2,v_3)} \longrightarrow^* Q \in \langle CM_{(1,v_1,v_2,v_3)} \rangle$. *Then the following hold:*

- $(1, v_1, v_2, v_3) \rightsquigarrow_M^* (i, v_1', v_2', v_3')$ *if and only if for all* $Q \in \langle CM_{(1,v_1,v_2,v_3)} \rangle$ *there exists some* $Q' \in \langle CM_{(i',v_1',v_2',v_3')} \rangle$ *such that* $Q \longrightarrow^* Q'$;
- *if* $Q \in \langle CM_{(i,v_1',v_2',v_3')} \rangle$ *and* $Q \longrightarrow^* Q'$, *then there exists* $Q'' \in \langle CM_{(i',v_1'',v_2'',v_3'')} \rangle$ *such that* $Q' \longrightarrow^* Q''$, *for suitable* i', v_1'', v_2'', v_3'';
- $(1, v_1, v_2, v_3) \Uparrow$ *if and only if* $CM_{M(v_1,v_2,v_3)} \Uparrow$. $\square$

- Corollario: **Finitary CCS è Turing-completo**.

Rendere osservabile la terminazione

- Tutte le transizioni sono etichettate tau! Non “vedo” nulla: un processo che termina e uno che diverge sono weak bisimili
- Si assume esista una nuova istruzione P_{m+1}:
  con √ etichetta osservabile!

$$P_{m+1} \stackrel{def}{=} p_{m+1}.\surd.\mathbf{0}$$

- Il processo $TCM_{M(v1, v2, v3)}$ nel suo complesso è ora

$$TCM_{M(v_1,v_2,v_3)} \stackrel{def}{=} (\nu L')(P_1 \mid \ldots \mid P_m \mid P_{m+1} \mid R_1 \mid R_2 \mid R_3 \mid B_{(v_1,v_2,v_3)})$$

- L’istruzione (i, DecJump(r_j,s)) viene modellata come

$$P_i \stackrel{def}{=} p_i.P_i' \qquad P_i' \stackrel{def}{=} \begin{cases} \overline{zero}_j.\overline{p}_s.P_i + \overline{dec}_j.\overline{p}_{i+1}.P_i & \text{if } s \leq m, \\ \overline{zero}_j.\overline{p}_{m+1}.P_i + \overline{dec}_j.\overline{p}_{i+1}.P_i) & \text{otherwise} \end{cases}$$

- Con questa nuova definizione, la CM M termina sse il processo CCS TCM_M è weakly bisimile a √.0.

Weak bisimilarity è indecidibile

1. Data una enumerazione delle CM, M_1, M_2, …, abbiamo anche una enumerazione di processi Finitary CCS TCM_{M1} , TCM_{M2} …
2. Halting problem può essere riformulato: se y è l'encoding di v_1, v_2, v_3, allora

   $$h(x, y) = \begin{cases} 1 & \text{se } TCM_{Mx(v1,\, v2,\, v3)} \approx \surd.0 \\ 0 & \text{altrimenti} \end{cases}$$

3. Se ≈ fosse decidibile, allora avrei trovato un algoritmo per calcolare la funzione h, che sappiamo non può esistere per formalismi Turing-completi.

Strong bisimilarity è indecidibile

- Consideriamo il processo Div = tau.Div
- Allora la CM M diverge sse $TCM_M \sim$ Div.
- Halting problem può essere riformulato: se y è l'encoding di v_1, v_2, v_3,

$$h'(x, y) = 0 \quad \text{se } TCM_{Mx(v1,\, v2,\, v3)} \sim \text{Div}$$
$$= 1 \quad \text{altrimenti}$$

Quindi anche strong bisimilarity è indecidbile tra processi di Finitary CCS.

Osservazioni (1)

Remark 3.15. **(Set $sort(p)$ is not effectively decidable)** Given an enumeration of CCS processes $p_1, p_2, p_3, \ldots$, as well as an enumeration of actions $\mu_1, \mu_2, \mu_3, \ldots$, function

$$Srt(x,y) = \begin{cases} 1 & \text{if action } \mu_y \text{ belongs to } sort(p_x) \\ 0 & \text{otherwise} \end{cases}$$

cannot be computable. If *Srt* were computable, then we would solve the halting problem. In fact, in the construction above, action $\surd$ belongs to $sort(TCM_M)$ if and only if the CM M terminates. This observation has the consequence that, in general, for a finitary CCS process p the set $sort(p)$ is *not effectively decidable*: even if set $sort(p)$ is finite (hence decidable) by Corollary 4.1, it is not possible to give explicitly an algorithm that checks when a given action μ belongs to $sort(p)$, even if we know that such an algorithm must exist. As a matter of fact, if $sort(p_x)$ were effectively decidable for all x, then function *Srt* would be easily effectively computable.[22] □

Osservazioni (2)

Exercise 3.73. (Reachability is undecidable) With the same intuition as above, one can conclude that the *reachability problem* is undecidable for finitary CCS. This can be formalized by means of the following function $Reach : \mathscr{P} \times \mathscr{P} \to \{0,1\}$:

$$Reach(p,q) = \begin{cases} 1 & \text{if } p \longrightarrow^* q \\ 0 & \text{otherwise} \end{cases}$$

Argue that if *Reach* were computable, then we would solve the halting problem for CMs. (*Hint:* CM M with inputs (v_1, v_2, v_3) terminates if and only if $TCM_{M(v_1,v_2,v_3)}$ reaches a state where instruction of index $m+1$ has been activated, i.e., a state/term which contains $\surd.\mathbf{0}$) □

- Il problema della reachability non può essere decidibile per formalismi Turing-completi.

Cosa succede per finite-net CCS?

- Decidibilità di reachability: posso decidere se un certo stato (ad esempio quello che fa √) è raggiungibile dallo stato iniziale. Allora …
- Halting problem risolvibile. Allora…
- Non Turing-completezza.
- Tuttavia, bisimulation equivalence è indecidibile per finite-net CCS, perché così è per reti di Petri finite. Vedi sezione 3.5.4.

Modellazione migliore … di 2-CM

- $CM_{M(v1,v2)} = (vL)(R_1 | R_2 | B(v_1,v_2))$
- $B(v_1,v_2)$ inizializza i due contatori (con valori godelizzati) e, anziché terminare con il processo finito $`p_1.0$, termina con con la costante P_1.
- L'istruzione (i, Inc(r_j)) è definita da

$$P_i = `inc_j.P_{i+1}$$

- L'istruzione (i, DecJump(r_j,s)) è modellata

$$P_i = `zero_j.P_s + `dec_j.P_{i+1}$$

Modellazione migliore … di 2-CM (2)

- I 2 contatori/registri sono definiti come al solito, per j = 1, 2,

$$R_j \stackrel{def}{=} zero_j.R_j + inc_j.((\nu a)(R_{j_1} \,|\, a.R_j))$$
$$R_{j_1} \stackrel{def}{=} dec_j.\bar{a}.\mathbf{0} + inc_j.((\nu b)(R_{j_2} \,|\, b.R_{j_1}))$$
$$R_{j_2} \stackrel{def}{=} dec_j.\bar{b}.\mathbf{0} + inc_j.((\nu a)(R_{j_1} \,|\, a.R_{j_2}))$$

- Usano in tutto 6 costanti e 6 azioni (ovvero Inc_j/dec_j/$zero_j$ per j = 1, 2), più le due azioni bound a e b, cioè 8 azioni in tutto.

Modellazione migliore …di 2-CM (3)

- Quindi bastano 8 azioni per modellare qualunque 2-CM.
- Bastano (m+1) + 6 costanti per modellare una Turing-machine universale (UTM) con una 2-CM, dove m è il numero di istruzioni della 2-CM che simula la UTM e 6 sono le costanti per i 2 contatori.
- Allora CCS(h,k) è Turing-completo, dove h=m+7 e k = 8, tuttavia m è un valore grande.
- Risultato noto con minimo hxk è: CCS(25,12) è Turing-completo, attraverso una simulazione diretta (non per mezzo di una 2-CM) di una Macchina di Turing Universale (UTM) con 15 stati e 2 simboli.