Linguaggi di programmazione con laboratorio

$$(\text{nome})\ \frac{\text{premesse}}{\text{conclusioni}}\ \text{condizione}$$ ← REGOLE DI INFERENZA

SEMANTICA OPERAZIONALE

SMALL STEPS

· ogni regola muove in genere un solo costrutto in base a una sola premessa.

Esempio.

$$(\text{sumL})\ \frac{E_0 \longrightarrow E_0'}{E_0 \oplus E_1 \rightarrow E_0' \oplus E_1}$$

...

$\underline{a \longrightarrow^* b}$

"alla fine con tot small steps raggiunge b"

BIG STEPS

· più costrutti vengono mossi in uno step, in genere con più premesse.

Esempio

$$(\text{sum})\ \frac{E_0 \rightarrow E_0' \quad E_1 \rightarrow E_1'}{E_0 \oplus E_1 \rightarrow E_0' \oplus E_1'}$$

...

$a \longrightarrow b$

" " con big steps
" "

In genere l'algoritmo e' bottom up: si sceglie il goal, si indovina la regola da cui e' dedotto e si risolvono le premesse come sotto goal.

Esempio. $(1 \oplus 2) \otimes (3 \oplus 4) \to 21$

$$\frac{\overbrace{1 \oplus 2 \to m_1}^{\text{SUBGOA}} \qquad \overbrace{3 \oplus 4 \to m_2}^{\text{SUBGOAL}}}{(1 \oplus 2) \otimes (3 \oplus 4) \to m} \quad (m_1 \cdot m_2 = m)$$

$$\Downarrow$$

$$(1+2=3)\ \frac{\dfrac{}{1 \oplus 2 \to 3} \qquad \dfrac{}{3 \oplus 4 \to 7}\ (3+4=7)}{(1 \oplus 2) \otimes (3 \oplus 4) \to 21}\ (3 \cdot 7 = 21)$$

Segnature e Termini

(Σ, ar), $\Sigma_i = ar^{-1}(i)$

Simboli

$ar : \Sigma \to \mathbb{N}$, l'arietà

(descritto anche come $\{ f/ar(f), \dots \}$)

(o $T_{\Sigma, X}$)

$\underbrace{T_\Sigma(X)}$:

Termini in Σ con le variabili di X.

$$\boxed{\frac{c \in \Sigma_0}{c \in T_\Sigma(X)} \qquad \frac{x \in X}{x \in T_\Sigma(X)} \qquad \frac{f \in \Sigma_i \quad e_1, \dots, e_i \in T_\Sigma(X)}{f(e_1, \dots, e_i) \in T_\Sigma(X)}\ (i > 0)}$$

$T_\Sigma \triangleq T_\Sigma(\emptyset) \to$ Termini chiusi

$vars : T_\Sigma(X) \to \wp(X) \to vars(t) = \{ \text{var. di } X \text{ in } t \}$

$(T_\Sigma = vars^{-1}(\emptyset), \forall X)$.

SOSTITUZIONE (finita): $\rho : X \to T_\Sigma(X)$ con $\rho \neq id$ per finiti $x \in X$.

ρ si estende naturalmente su $T_\Sigma(X)$ ed è t.c.

$$\begin{cases} \rho(f(e_1, \dots, e_t)) = f(\rho(e_1), \dots, \rho(e_t)) \\ \rho(c) = c \end{cases}$$

(si scrive anche $t\rho$ per dire $\rho(T)$)

$\rightarrow \rho = [x_1 = t_1, \dots, x_m = t_m]$

$\rightarrow$ **RELAZIONE MGT** (more general than):

t_1 mgt $t_2 \overset{def}{\iff} \exists \rho$ sost. t.c. $t_2 = \rho(t_1)$.

Si estende anche alle sostituzioni:

ρ_1 mgt $\rho_2 \overset{def}{\iff} \exists \rho$ sost. t.c. $\rho_2 = \rho \circ \rho_1$

Problema del unificazione

Dato un sistema $G = \{ l_1 \overset{?}{=} r_1, \dots, l_m \overset{?}{=} r_m \}$ Trovare ρ (più generale) t.c. $\rho(l_i) = \rho(r_i)$

uniche a meno di renaming

Algoritmo

$G \cup \{t = t\}$ [DELETE]
$\rightarrow G$

$G \cup \{ f(u_1, \dots, u_m) = f(u_1', \dots, u_m') \}$
$\rightarrow G \cup \{ u_1 = u_1', \dots, u_m = u_m' \}$ [DECOMPOSE]

$G \cup \{x = t\}$ [ELIMINATE VAR]
$\rightarrow G[x = t] \cup \{x = t\}$

$G \cup \{ f(u_1, \dots, u_m) = g(u_1, \dots, u_m) \}$
$\rightarrow$ imp., se $f \neq g$ o $m \neq m$. [CONFLICT]

$G \cup \{ x = f(u_1, \dots, u_m) \}$
$\rightarrow$ imp., se $x \in vars(u_i)$ per un i [OCCUR-CHECK]

$G \cup \{ f(u_1, \dots, u_m) = x \} \rightarrow$
$\rightarrow G \cup \{ x = f(u_1, \dots) \}$ [SWAP]

Sistemi logici

$$\frac{}{y} \rightsquigarrow \text{assiomi}$$

Un insieme di regole → SISTEMA LOGICO

$d \Vdash_R y$ → y si deriva come teorema in R con l'albero di derivazione d

$$\Vdash_R y \triangleq \exists d \text{ t.c. } d \Vdash y.$$

$$I_R = \{ y \mid \Vdash_R y \} \rightarrow \text{TEOREMI}$$

Notazione inline

$(1 \oplus 2) \otimes (3 \oplus 4) \to 21$

$\nwarrow$ $1 \oplus 2 \to 3$, $3 \oplus 4 \to 7$

$\nwarrow$ $1 \to 1$, $2 \to 2$, $(3 \oplus 4) \to 7$

$\nwarrow_*$ $3 \oplus 4 \to 7$

$\nwarrow_*$

"skip"

Prolog e SLD

X - variabili Σ - segnatura di funzioni

Π - segnatura predicati

$h :- r_1, \dots, r_m$ ⇝ CLAUSOLA DI HORN

$$\wr\wr$$

$$\frac{r_1 \quad r_2 \quad \dots \quad r_m}{h}$$

?- goal (sintassi Prolog)

$\downarrow$

cerca tramite l'alg. di unificazione una premessa valida su sostituzione fino a trovare un risultato. } SLD

Esempio

sum(0, y, y). $[0 + y = y]$

sum(s(x), y, s(z)) :- sum(x, y, z). $[x + y = z \Rightarrow \Rightarrow (x+1) + y = = z+1]$

?- sum(s(s(0)), s(s(0)), m) $[2+2=?]$

sum(s(s(0)), s(s(0)), m)

$\uparrow$ [m = s(z)] sum(s(0), s(s(0)), z)

$\uparrow$ [z = s(w)] sum(0, s(s(0)), w)

$\uparrow$ [w = s(s(0))] $\square$ $\longrightarrow$

$\rightsquigarrow$ m = s(s(s(s(0)))) $\longrightarrow$ m = 4 ✓

Induzione

Sia $\prec \subseteq A \times A$. Allora si dice che $\prec$ è ben fondata se <u>non</u> ammette catene discendenti infinite.

$\prec^+$ = chiusura transitiva di $\prec$

$\prec^*$ = chiusura riflessiva di $\prec$ e transitiva

$(\prec \subseteq \prec^+ \subseteq \prec^*)$

$(\prec$ b.f. $\Longleftrightarrow$ $\prec^+$ b.f.$)$

$$\begin{cases} \prec \text{ ha cicli sse } \exists a \text{ t.c. } a \prec^+ a. \\ \prec \text{ b.f. } \Rightarrow \prec \text{ aciclica.} \\ \prec \text{ aciclica e } A \text{ finito } \longrightarrow \prec \text{ b.f.} \end{cases}$$

Teorema (induzione b.f.)
Se $\prec \subseteq A \times A$ è b.f., allora:

$$(\forall a,\ P(a)) \iff (\forall a,\ \underbrace{(\forall b \prec a,\ P(b)) \Rightarrow P(a)}_{})$$

include i casi base (i minimali), per cui la premessa è vera.

$\leadsto$ su $T_{\Sigma,X}$ si può costruire la relazione $\prec$ di sottotermine immediato:

$$\frac{f \in \Sigma_i \quad e_1, \dots, e_i \in T_{\Sigma,X}}{(e_i, f(e_1, \dots, e_i)) \in \prec}$$

Siccome depth: $T_{\Sigma,X} \longrightarrow \mathbb{N}$ traduce catene discend. in catene discendenti, $\prec$ su $T_{\Sigma,X}$ è b.f.
L'induzione su $T_{\Sigma,X}$ si dice INDUZIONE STRUTTURALE.

Segnature con Tipo

Stavolta si richiede più struttura:

$$\begin{cases} S \leftarrow \text{insieme di Tipi} \\ \Sigma = \{ \Sigma_{s_1 s_2 \dots s_m, s} \}_{s_1, s_2, \dots, s_m, s \in S} \text{ dove} \\ \Sigma_{s_1 s_2 \dots s_m, s} \text{ contiene simboli di funzione} \\ f: s_1 \times \dots \times s_m \to s. \end{cases}$$

(si usa ε in caso di dominio vuoto)

Si definisce in modo naturale $T_{\Sigma, s}$ come l'insieme dei termini chiusi nel tipo s (i.e. codominio s).

Esempio. (IMP)

$$S = \{Aexp,\ Bexp,\ Com\}$$

arithmetical expr. — boolean exp. — command

$$\left.\begin{array}{l} \Sigma_{Aexp\,Aexp,\,Aexp} = \{+,\ -,\ \times,\ /\} \\ \Sigma_{\varepsilon,\,Aexp} = \underbrace{Ide}_{\text{identificatori (per le var.)}} \cup \mathbb{Z} \end{array}\right\} Aexp$$

$$\left.\begin{array}{l} \Sigma_{Aexp\,Aexp,\,Bexp} = \{=,\ \leq\} \\ \Sigma_{Bexp\,Bexp,\,Bexp} = \{\vee,\ \wedge\} \\ \Sigma_{Bexp,\,Bexp} = \{\neg\} \\ \Sigma_{\varepsilon,\,Bexp} = \{true,\ false\} \end{array}\right\} Bexp$$

$$\left.\begin{array}{l} \Sigma_{Com\,Com,\,Com} = \{;\} \\ \Sigma_{Bexp\,Com\,Com,\,Com} = \{if\ *\ then\ *\ else\ *\} \\ \Sigma_{Bexp\,Com,\,Com} = \{while\ *\ do\ *\} \\ \Sigma_{Aexp,\,Com} = \{x := *\ \mid x \in Ide\} \\ \Sigma_{\varepsilon,\,Com} = \{skip\} \end{array}\right\} Com$$

Definiamo $\mathbb{M} = \{ f \in \mathbb{Z}^{\mathrm{Ide}} \mid f(x) \neq 0$ per finiti $x \}$

$\leadsto$ $(m_1/x_1, \ldots, m_k/x_k)$ corrisponde a σ t.c. $\sigma(x_i) = m_i$ e $\sigma(x) = 0$ per ogni altra x.

$\leadsto$ $\sigma[m/x]$ sostituzione in σ t.c. $x = m$:

$$\sigma[m/x](y) = \begin{cases} \sigma(y) & \text{se } y \neq x \\ m & \text{se } y = x \end{cases}$$

(si estende naturalm. a $\sigma[m_1/x_1, \ldots, m_k/x_k]$ per x_i distinti)

$\leadsto$ per mostrare alcuni fatti sui termini talvolta l'induzione strutturale <u>non</u> è adeguata (es. nei casi di semantica ricorsiva, nella premessa di una regola c'è la stessa complessità logica della conclusione.

Si definisce:

$D_R = \{ d \mid d \Vdash_R y \}$ come l'insieme di derivazioni in R.

Su D_R si pone $d \prec d'$ secondo la sottoderivazione immediata:

$$\prec = \{ (d_i, \frac{d_1 \ldots d_m}{y}) \}$$

$\prec$ è b.f. (height: $D_R \rightarrow \mathbb{N}$ che associa le deriv. alle loro altezze nell'albero fa le veci di depth su $\prec$ per $T_{\Sigma,X}$).

L'induzione su D_R è detta INDUZIONE SULLE REGOLE:

$$\frac{\forall \frac{x_1 \dots x_m}{y} \in R \ (\{x_1, \dots, x_m\} \subseteq I_R \wedge P(x_1) \wedge \dots) \Longrightarrow P(y)}{\forall x \in I_R, \quad P(x)}$$

(I casi base sono gli assiomi e le regole con premesse non derivabili)

Equivalenza tra programmi

→ $c \sim_o c' \overset{def}{\Longleftrightarrow} \forall \sigma, \sigma' (\langle c, \sigma \rangle \to \sigma' \Longleftrightarrow \langle c', \sigma \rangle \to \sigma')$.

→ tutti i comandi divergenti sono equivalenti

Per la semantica denotazionale, $c \sim_D c' \overset{def}{\Longleftrightarrow} C[\![c]\!] = C[\![c']\!]$. Mentre abbiamo dovuto dimostrare il determinismo dei comandi nella semantica operazionale, nella semantica denotazionale il determinismo è dovuto al fatto che $C[\![\cdot]\!]$ è una funzione — ben definita grazie al teorema di ricorsione:

Teorema (di ricorsione) Sia A b.f. Allora $\exists! f : A \to B$ dove $\forall a \in A, \ f(a) = g(a, f|_{\lfloor a \rfloor})$ $(\lfloor a \rfloor \triangleq \{b \in A \mid b \prec a\})$.

→ applicato a $(Aexp, \prec)$, $(Bexp, \prec)$ e $(Com, \prec)$, il teorema mostra che le semantiche denotazionali sono ben definite.

→ Aexp e Bexp sono CONSISTENTI: arrivano allo stesso risultato sia in semantica operazionale che in quella denotazionale.

Ordini parziali e funzioni monotone o continue

Un poset $(A, \preceq)$ e' tale che:

- $a \preceq a$ (riflessiva) $\forall a \in A$
- $a \preceq b \wedge b \preceq a \implies a = b$ (antisimmetrica) $\forall a, b \in A$
- $a \preceq b \wedge b \preceq c \implies a \preceq c$ (transitiva) $\forall a, b, c \in A$

Si dice che $\preceq$ é totale se ogni coppia di elementi é confrontabile.

$f : (A, \preceq) \longrightarrow (B, \sqsubseteq)$ si dice MONOTONA se
$a \preceq b \implies f(a) \sqsubseteq f(b)$.

Un PO (partial order) si dice COMPLETO (CPO) se $\forall$ catena $a_1 \preceq a_2 \preceq \dots$ $\exists$ un LUB (least upper bound), $\triangleq \bigsqcup_{i \in I} a_i$ (su catene finite esiste sempre).

Se $\preceq$ e $\sqsubseteq$ sono CPO, f si dice CONTINUA se preserva i LUB, i.e. $\underbrace{f\left(\bigsqcup_{i \in I} A_i\right)}_{\text{sempre un bound sugli } f(a_i)} = \bigsqcup_{i \in I} f(A_i)$

CONTINUITÁ DI SCOTT

$\preceq$ $\qquad\qquad$ $<$

$a \preceq b \wedge a \neq b \longrightarrow a < b$

$a \preceq b \longleftarrow a < b \vee a = b$

Se esiste, il minimo si indica con $\perp$ (bottom).

$\rightsquigarrow$ ogni funzione continua é monotona ($a_1 \preceq a_2 \implies$
$\implies f\left(\bigsqcup_{i \in I} a_i\right) = \bigsqcup_{i \in I} f(a_i)$
$f\left(\bigsqcup_{i \in I} a_i\right) = f(a_2)$
$\implies f(a_1) \sqsubseteq f(a_2)$)

$\rightsquigarrow$ ogni funzione monotona su A con catene solamente finite è continua.

Composizione di monotone/continue è monotona/continua

Lemma se $(A, \leq)$ è t.c. $\leq \in PO_\perp$, allora $f: A \to A$ è t.c. $\{f^n(\perp)\}_{n \in \mathbb{N}}$ è una catena.

$\perp \leq f(\perp)$ dacché $\perp$ è bottom. Quindi

$\perp \leq f(\perp) \leq f^2(\perp) \leq \dots$ per monotonicità ■

Teorema (di Kleene) Se $(A, \leq)$ è $CPO_\perp$ e f è continua, $\bigsqcup_{i \in \mathbb{N}} f^i(\perp) \triangleq fix\ f$ è il minimo punto fisso di f.

- $f\left(\bigsqcup_{i \in \mathbb{N}} f^i(\perp)\right) = \bigsqcup_{i \in \mathbb{N}} f^{i+1}(\perp) = \bigsqcup_{i \in \mathbb{N}} f^i(\perp)$ ✓

- $f(x) = x$. $\quad \perp \leq x \Longrightarrow f(\perp) \leq \underbrace{f(x)}_{=x}$

$\Longrightarrow f^i(\perp) \leq x \quad \forall i \in \mathbb{N} \Longrightarrow$

$\Longrightarrow \bigsqcup_{i \in \mathbb{N}} f^i(\perp) \leq x$. ✓ ■

$\rightarrow$ identifichiamo $Pf(A, B)$ con $Fun(A, B \sqcup \{\perp\})$.

$f(a)$ non def. $\longleftrightarrow$ $\bar{f}(a) = \perp$

Operatore delle conseguenze immediate

Se R è un sistema logico e S è un insieme di formule, $\hat{R} : \mathcal{P}(I_R) \to \mathcal{P}(I_R)$ — detto OPERATORE DELLE CONS. IMM. — è t.c.

$$\hat{R}(S) = \left\{ y \;\middle|\; \exists \frac{x_1 \dots x_m}{y} \text{ ist. in } R \text{ con } \{x_1, \dots, x_m\} \subseteq S \right\}$$

$\mathcal{P}(I_R)$ è un $CPO_\perp$ con $\subseteq$.

$\hat{R}$ è continua, se le regole di R hanno premesse finite.

- $\hat{R}\left(\bigcup_{i \in \mathbb{N}} S_i\right) \supseteq \bigcup_{i \in \mathbb{N}} \hat{R}(S_i)$:

  ogni $\hat{R}(S_i)$ è t.c. $\hat{R}\left(\bigcup_{i \in \mathbb{N}} S_i\right) \supseteq \hat{R}(S_i)$ (le cons. di S_i sono tra quelle di $\bigcup_{i \in \mathbb{N}} S_i$)

- $\hat{R}\left(\bigcup_{i \in \mathbb{N}} S_i\right) \subseteq \bigcup_{i \in \mathbb{N}} \hat{R}(S_i)$:

  se $y \in \hat{R}\left(\bigcup_{i \in \mathbb{N}} S_i\right)$, $\exists\, x_1, \dots, x_m \in S_{i_1}, \dots, S_{i_m}$ t.c. $\frac{x_1 \dots x_m}{y}$ ist. di R. Poiché $\{S_i\}_{i \in \mathbb{N}}$ è una catena, $\exists\, k = \max\{i_j\}$ è t.c. $x_i \in S_k \;\forall i \Longrightarrow y \in \hat{R}(S_k) \subseteq \bigcup_{i \in \mathbb{N}} \hat{R}(S_i)$.

Per il teorema di Kleene, $\overbrace{\bigcup_{i \in \mathbb{N}} \hat{R}(\underbrace{\emptyset}_{=\perp})}^{=I_R}$ è il minimo punto fisso di $\hat{R}$.

$\rightsquigarrow \lambda x.\, e \longleftrightarrow [x \mapsto e]$ (lambda notation) per le funzioni anonime

$\rightsquigarrow A \to B, C \longleftrightarrow$ if A: B else: C

Semantica denotazionale di IMP

$C[\![\text{while } b \text{ do } c]\!]\sigma = B[\![b]\!]\sigma \to C[\![\text{while } b \text{ do } c]\!]^*(C[\![c]\!]\sigma), \sigma,$ (1)

dove se $f: \underbrace{\Sigma}_{M} \to \underbrace{\Sigma_\perp}_{M_\perp}$, $f^*: \Sigma_\perp \to \Sigma_\perp$ è t.c. $f^*|_\Sigma = f$ e $f(\perp) = \perp$.

LIFTING

(1) $\Rightarrow C[\![\text{while } b \text{ do } c]\!]$ è un p.to fisso di

$$\underbrace{\Gamma_{b,c} = \lambda\varphi.\, \underbrace{\lambda\sigma.\, \underbrace{B[\![b]\!]\sigma \to \varphi^*(C[\![c]\!]\sigma), \sigma}_{\Sigma_\perp}}_{\Sigma \to \Sigma_\perp}}_{\text{siccome } C[\![\text{while } b \text{ do } c]\!] \text{ è } \Sigma \to \Sigma_\perp, \text{ il totale è}}$$

$$\underbrace{(\Sigma \to \Sigma_\perp)}_{\text{ha } \perp} \longrightarrow (\Sigma \to \Sigma_\perp)$$

Scelto $R_{b,c} = \left\{ \frac{(\sigma'', \sigma')}{(\sigma, \sigma')} (B[\![b]\!]\sigma \wedge C[\![c]\!]\sigma = \sigma''), \frac{}{(\sigma, \sigma)} (\neg B[\![b]\!]) \right\}$, $R_{b,c}$ è un sistema logico t.c. $\hat{R}_{b,c} \equiv \Gamma_{b,c}$ sulle funz. parz. $\Sigma \to \Sigma_\perp$.

Le approssimazioni $\hat{R}_{b,c}(\emptyset) = \perp$, $\hat{R}_{b,c}^2(\emptyset) = \perp$, ... aggiungono risp. i cicli con 0, 1, ... iterazioni:

$$C[\![\text{while } b \text{ do } c]\!] = \bigcup_{i \in \mathbb{N}} \hat{R}_{b,c}^i(\emptyset) =$$

$$= \text{fix } \Gamma_{b,c}$$

$\rightsquigarrow$ questo metodo si generalizza ad altri comandi ricorsivi.

Composizionalità, consistenza e completezza

Principio di composizionalità: il significato di un'espr. è univoc. determ. dal significato dei suoi costituenti, nella semantica denotazionale

I comandi sono CONSISTENTI (CORRETTI e COMPLETI)

CORRETTI	COMPLETI
$\underbrace{\langle c, \sigma \rangle \to \sigma' \implies C[\![c]\!]\sigma = \sigma'}$	$\underbrace{C[\![c]\!]\sigma = \sigma' \implies \langle c, \sigma \rangle \to \sigma'}$
induz. sulle regole	induz. strutturale

Inoltre la semantica operaz. costruita è CONGRUENTE:

$$a_1 \sim_{op} a_2 \implies \mathbb{A}[a_1] \sim_{op} \mathbb{A}[a_2], \quad \text{con}$$

$\mathbb{A}[\cdot]$ contesto

HOFL - Higher Order Functional Language

$\rightsquigarrow [\![rec\ x.t]\!]\rho = [\![t]\!]\rho\left[{}^{[\![rec\ x.t]\!]\rho}/_{t}\right] \Longrightarrow$

$\Longrightarrow [\![rec\ x.t]\!]\rho = fix\ \lambda d.\ [\![t]\!]\rho\left[{}^{d}/_{t}\right].$

$t ::= x \in Ide \mid n \in \mathbb{Z} \mid t_0\ \underbrace{op}_{+,-,\cdot,/}\ t_1 \mid if\ t\ then\ t_0\ else\ t_1$

$t = 0 \to true$

$t \neq 0 \to false$

$\mid (t_0, t_1) \mid fst(t) \mid snd(t) \mid \lambda x.t \mid t_0\ t_1$
$\mid rec\ x.t$

PRE-TERMINI

(non sono ancora tipati!)

$\tau ::= int \mid \overbrace{\tau_0 * \tau_1}^{\text{coppi}} \mid \tau_0 \to \tau_1 \}$ TIPI

Assumiamo variabili tipate $\hat{\cdot} : Ide \to \tau$ restituisce il tipo.

$\underbrace{t : \tau}_{t \text{ ha tipo } \tau}$ $\rightsquigarrow$ si costruiscono con queste le regole d'inferenza

Un pre-termine t si dice ben fondato se $\exists \tau \in T \mid t : \tau$.

CHURCH TYPE THEORY

data un'etichetta di tipo si deduce il tipo dei termini sottostanti (bottom-up)

CURRY TYPE THEORY

si inferiscono i tipi tramite l'algoritmo di unificazione.

Esempio. $t \overset{def}{=} \text{rec } p . \lambda x . (\underbrace{x}_{int}, \underbrace{\underbrace{p}_{int \to \tau_4} \underbrace{(x+2)}_{int}}_{\tau_4})$

$$\underbrace{\underbrace{(x, p(x+2))}_{int * \tau_4}}_{int \to \tau_4}$$

$\Longrightarrow \tau_4 = int * \tau_4 , \quad \text{↯} \quad$ (occur-check)

Variabili libere

$fv(*) = \{\text{variabili libere in } *\}$

Esempio. $fv(x) = \{x\} \qquad fv(x + (\lambda y . y)(3)) = \{x\}$

$\rightsquigarrow (\lambda y . t)[t'/x] \triangleq \lambda z . (t[z/y][t'/x])$

$\forall z \notin fv(\lambda y . t) \cup fv(t') \cup \{x\}$

$\rightsquigarrow (\text{rec } y . t)[t'/x] = \text{rec } z . t[z/y][t'/x]$

$\forall z \notin fv(\text{rec } y . t) \cup fv(t') \cup \{x\}$

Le sostituzioni rispettano i tipi e li mantengono.

Si assegna una semantica solo ai termini ben tipati e chiusi. In HOFL si implementa una semantica operazionale big step con FORME CANONICHE:

$\underbrace{C_\tau}_{\text{forme can. di tipo } \tau} \subseteq T_\tau$

$$\frac{}{n \in C_{int}} \qquad \frac{t_0 : \tau_0 \quad t_1 : \tau_1 \quad t_0, t_1 \text{ chiusi}}{(t_0, t_1) \in C_{\tau_0 * \tau_1}}$$

$$\frac{\lambda x . t : \tau_0 \to \tau_1 \quad \lambda x . t \text{ chiuso}}{\lambda x . t \in C_{\tau_0 \to \tau_1}}$$

(La semantica è lazy, non valuta i "costrutti" per trovare le forme canoniche)

LAZY

$$\frac{t_1 \to \lambda x.\, t_1' \quad t_1'[{}^{t_0}/_{x}] \to c}{(t_1\ t_0) \to c}$$

EAGER

$$\frac{t_1 \to \lambda x.\, t_1' \quad t_0 \to c \quad t_1'[{}^{c}/_{x}] \to c'}{(t_1\ t_0) \to c'}$$

Esempio $(\lambda y.\, 1)\ (\mathrm{rec}\ x.\, x) \to 1$ (lazy)

$\not\to$ (eager)

La semantica op. di HOFL e' deterministica, non cambia i tipi in fase di esecuzione $(t \to c\ ,\ t : \tau \Rightarrow c : \tau)$, ma $\equiv_{op}$ <u>non</u> e' una congruenza:

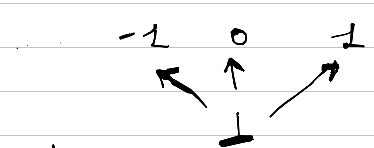
$$1 + 1 \equiv_{op} 2$$

$$\lambda x.\, 1 + 2 \not\equiv_{op} \lambda x.\, 2$$

In HOFL abbiamo infiniti tipi, quindi è necessario approfondire la teoria dei (CPO$_{(\perp)}$).

Ogni tipo τ sara' associato a un dominio semantico D_τ in modo che $[\![t]\!]\rho \in D_\tau$ dove $t : \tau$.

Scegliamo $D_{int} = \mathbb{Z}_\perp$ e costruiamo $D_1 \times D_2$ e $D_1 \to D_2$.



nessun ordine su $\mathbb{Z}$ (e' piatto)

Ogni op. su $\mathbb{Z}$ si estende su $\mathbb{Z}_\perp$:

$$\underbrace{a\ \underline{op}\ b}_{\mathbb{Z}_\perp \times \mathbb{Z}_\perp \to \mathbb{Z}_\perp} = \begin{cases} \perp & a = \perp \text{ o } b = \perp \\ a\ op\ b & \text{altrim.} \end{cases}$$

$\longrightarrow$ estensione strict

- $\underline{op}$ é monotono e quindi continuo ($\mathbb{Z}_\perp$ ha solo cat. di al più 2 el.) (su $\mathbb{Z}_\perp \times \mathbb{Z}_\perp$, con $\leq$ def. come segue ora).

Se A e B sono $CPO_\perp$, anche $A \times B$ lo è
se:

$$(a,b) \le (a',b') \iff a \le a' \land b \le b'$$

$$\perp_{A\times B} = (\perp_A, \perp_B) \quad e \quad \bigsqcup_{i\in\mathbb{N}} (a_i,b_i) = \left(\bigsqcup_{i\in\mathbb{N}} a_i, \bigsqcup_{i\in\mathbb{N}} b_i\right)$$

$\rightarrow$ π_1 e π_2 sono monotone e continue.

Switching lemma

Se $e_{n,m} \le e_{n',m'} \Longleftarrow n \le n' \land m \le m'$,
allora:

$$\begin{array}{ccccccccc}
\bigsqcup_{m\in\mathbb{N}} e_{0,m} & \le & \bigsqcup_{m\in\mathbb{N}} e_{1,m} & \le & \cdots & & \bigsqcup_{m\in\mathbb{N}} e_{m,m} & & \\
\rotatebox{90}{$\le$} & & \rotatebox{90}{$\le$} & & & \cdots & \rotatebox{90}{$\le$} & & \\
\vdots & & \vdots & & e_{2,2} & & \vdots & & \\
\rotatebox{90}{$\le$} & & \rotatebox{90}{$\le$} & \le & \rotatebox{90}{$\le$} & & \rotatebox{90}{$\le$} & & \\
e_{0,1} & \le & e_{1,1} & \le & e_{2,1} & \le \cdots \le & \bigsqcup_{m\in\mathbb{N}} e_{m,1} & & \\
\rotatebox{90}{$\le$} & \le & \rotatebox{90}{$\le$} & & \rotatebox{90}{$\le$} & & \rotatebox{90}{$\le$} & & \\
e_{0,0} & \le & e_{1,0} & \le & e_{2,0} & \le \cdots \le & \bigsqcup_{m\in\mathbb{N}} e_{m,0} & &
\end{array}$$

SWITCHING LEMMA

$$\boxed{\bigsqcup_{m\in\mathbb{N}} \bigsqcup_{m\in\mathbb{N}} e_{n,m} = \bigsqcup_{m\in\mathbb{N}} \bigsqcup_{m\in\mathbb{N}} e_{m,m} = \bigsqcup_{m\in\mathbb{N}} e_{m,m}}$$

Domini funzionali

(lo stesso ordine vale su $A \to B$)

Diamo un ordine su $[A \to B] \overset{\Delta}{=} \{ f : A \to B \mid f \text{ continua} \}$:

$$f \le g \iff \forall a \in A,\ f(a) \le g(a)$$

$\rightsquigarrow$ su $[\mathbb{Z}_\perp \to \mathbb{Z}_\perp]$, $f \le g \iff g$ $\underbrace{\text{più definita di}}$ f

$B_\perp^A \cong Pf(A,B)$ naturalmente

$A \to B$ è $CPO_\perp$:

- $\perp_{[A,B]} = \lambda x. \perp_B$ (BOTTOM)

- $f_1 \leq f_2 \leq \dots \rightsquigarrow \bigsqcup_{i \in \mathbb{N}} f_i = \lambda x. \underbrace{\bigsqcup_{i \in \mathbb{N}} f_i(x)}_{h(x)}$.

(LIMITE)

Infatti h è un upper bound degli f_i e se g è un upper bound:

$\{ g(a)$ upper bound degli $f_i(a) \Longrightarrow \rightsquigarrow$

$\Longrightarrow g(a) \geq \bigsqcup_{i \in \mathbb{N}} f_i(a)$

$\rightsquigarrow g \geq h$ ✓

Lemma Se le f_i sono continue, anche h lo è.

$h\left(\bigsqcup_{m \in \mathbb{N}} d_i\right) = \bigsqcup_{i \in \mathbb{N}} f_i\left(\bigsqcup_{m \in \mathbb{N}} d_i\right) \overset{f_i \text{ cont.}}{=}$

$= \bigsqcup_{i \in \mathbb{N}} \bigsqcup_{m \in \mathbb{N}} f_i(d_i) \overset{\text{SWITCHING LEMMA}}{=} \bigsqcup_{m \in \mathbb{N}} \underbrace{\bigsqcup_{i \in \mathbb{N}} f_i(d_i)}_{h(d_i)}$

$= \bigsqcup_{m \in \mathbb{N}} h(d_i)$. ■

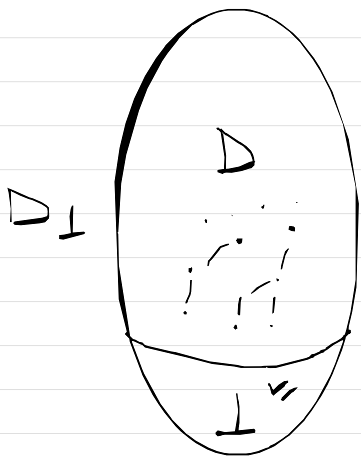
$\rightsquigarrow$ quindi anche $[A \to B]$ è $CPO_\perp$, se A e B lo sono.

Domini arricchito con bottom

Se A è CPO, si definisce $A_\bot$ come:

$$A_\bot = \{\bot\} \sqcup A,$$

e si pone $\bot \leq \lfloor a \rfloor$ $\forall a$, dove $\lfloor \cdot \rfloor$ è l'imm. di A in $A_\bot$ (lifting) — $\lfloor\ \rfloor^{-1} : \lfloor A \rfloor \to A$ è detto delifting.



Se $f : A \to B$ con B $CPO_\bot$, allora:

$$\underbrace{f^*(x)}_{A_\bot \to B} = \begin{cases} f(a) & \exists a, x = \lfloor a \rfloor \\ \bot_B & x = \bot \end{cases}$$

LIFTING di funzioni

- f continua $\Longrightarrow$ f^* continua
- $\cdot^*$ continuo e monotono $(\cdot^* : [A \to B] \to [A_\bot \to B])$

$\rightsquigarrow$ $\text{let } x \Leftarrow t . e \triangleq (\lambda x . e)^*(t)$
(estensione di $\text{let } x = t . e$).

- $f : A \to B \times C$ continua sse le sue proiez. lo sono
- $f : A \times B \to C$ continua sse f_a, f_b cont. $\forall a, b$

$[b \to f(a,b)]$ $\qquad$ $[a \to f(a,b)]$

- la valutazione eval/apply $[D \times E] \times D \to E$ è continua e anche $\underbrace{\text{fix}}_{[D \to D] \to D}$ lo è.

Definiamo allora:

$$\begin{cases} D_{\tau_1 \times \tau_2} \triangleq (D_{\tau_1} \times D_{\tau_2})_\perp & e \\ D_{\tau_1 \to \tau_2} \triangleq [D_{\tau_1} \to D_{\tau_2}]_\perp \end{cases}$$

→ $Cond_\tau : \mathbb{Z}_\perp \times D_\tau \times D_\tau \to D_\tau$ è Tc.

$$Cond_\tau(v, d_1, d_2) \triangleq \begin{cases} \perp_{D_\tau} & se \quad v = \perp_{\mathbb{Z}_\perp} \\ d_1 & se \quad v = \lfloor 0 \rfloor \\ d_2 & altrimenti \end{cases}$$

diverge con $[\![t]\!]\rho = \perp$

→ $[\![t\ t_0]\!]\rho \triangleq let\ \varphi \Leftarrow [\![t]\!]\rho\ .\ \varphi([\![t_0]\!]\rho)$ (LAZY)

→ $[\![t\ t_0]\!]\rho \triangleq let\ \varphi \Leftarrow [\![t]\!]\rho\ .\ let\ d \Leftarrow [\![t_0]\!]\rho\ .\ \varphi(\lfloor d \rfloor)$ (EAGER)

diverge anche con $[\![t_0]\!]\rho = \perp$

Esempio $f \triangleq \lambda x.1$ $\quad x : int$
$\quad y : int$

(lazy) $[\![f\ rec\ y.y]\!]\rho =$

$= let\ \varphi \Leftarrow [\![\lambda x.1]\!]\rho\ .\ \varphi(\underbrace{[\![rec\ y.y]\!]\rho}_{=\perp_{\mathbb{Z}_\perp}}) = \Big([\![\lambda x.1]\!]\rho =$
$= \lfloor \lambda d . [\![1]\!]\rho[^d/_x] \rfloor =$
$= \lfloor \lambda d . \lfloor 1 \rfloor \rfloor \Big)$

$= \lfloor \lambda d . \lfloor 1 \rfloor \rfloor (\perp_{\mathbb{Z}_\perp}) = 1$

(eager) $[\![f\ rec\ y.y]\!]\rho = let\ \varphi \Leftarrow [\![f]\!]\rho\ .\ let\ d \Leftarrow [\![rec\ y.y]\!]\rho\ .$
$\varphi(\lfloor d \rfloor) =$
$= let\ d \Leftarrow \underbrace{[\![rec\ y.y]\!]\rho}_{=\perp_{\mathbb{Z}_\perp}}\ .\ \lfloor \lambda d . \lfloor 1 \rfloor \rfloor (\lfloor d \rfloor) =$

$= \perp_{\mathbb{Z}_\perp}$

→ Cond, lamda, ... sono continue e monotone!

Lemma (di sostituzione) $[\![t[^{t_1}/_x]]\!] \rho = [\![t]\!] \rho \left[^{[\![t_1]\!] \rho}/_x \right]$

$\rightsquigarrow$ garantisce che $\forall x \in fv(t),\ \rho(x) = \rho'(x) \Longrightarrow$
$\Longrightarrow [\![t]\!] \rho = [\![t]\!] \rho'$.

(Quindi t chiusi $\Longrightarrow [\![t]\!] \rho = [\![t]\!] \rho' \quad \forall \rho, \rho'$).

$\rightsquigarrow$ i termini canonici <u>non</u> sono bottom.

In HOFL, vale la correttezza: (su int è consistente)

$$t \longrightarrow c \Longrightarrow [\![t]\!] \rho = [\![c]\!] \rho$$

$\rightsquigarrow$ $t \downarrow \overset{def}{\Longleftrightarrow} t \longrightarrow c$ per qlc. c $\quad (t \uparrow \triangleq \neg t \downarrow)$

$\rightsquigarrow$ $t \Downarrow \overset{def}{\Longleftrightarrow} [\![t]\!] \rho \neq \bot \quad \forall \rho$ $\quad (t \Uparrow \triangleq \neg t \Downarrow)$

$$t \downarrow \Longleftrightarrow t \Downarrow$$

(<u>non</u> valida se non si aggiungono i bottom nella semantica)

In particolare $\equiv_{op} \subsetneq \equiv_{den}$.

CCS (Calculus of communicating systems)

→ $\overline{\alpha}$, uscita con label α; α, entrata "

$p, q ::=$ nil   processo inattivo $p\backslash\{\alpha_1, \dots, \alpha_m\} \triangleq p\backslash\alpha_1\backslash \dots \backslash\alpha_m$

| x variabile

| $\mu.p$ ($\mu.p \xrightarrow{\mu} p$)

| $p\backslash\alpha$ (p ristretto senza α e $\overline{\alpha}$)

| $p[\phi]$ (rietichettatura)

| $p+q$ (scelta <u>non</u> deterministica)

| $p|q$ (parallelo)

| rec $x.p$ (ricorsione) → unico binder

Esempio $t \triangleq \text{rec } x.\ \text{coin}(\overline{\text{coffee}}.x + \overline{\text{tea}}.\text{nil})$

$$\begin{tikzcd}
t \arrow[d, "\text{coin}"] \\
\overline{\text{coffee}}.t + \overline{\text{tea}}.\text{nil} \arrow[u, bend left=60, "\overline{\text{coffee}}"] \arrow[d, "\overline{\text{tea}}"] \\
\text{nil}
\end{tikzcd}$$

LTS di t (labelled transition system)

I processi paralleli con un input si muovono da una sola parte; possono <u>anche</u> sincronizzarsi:

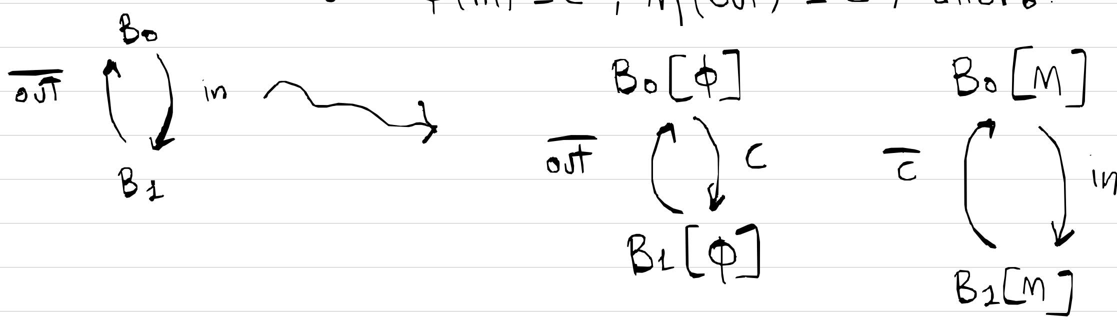
$$\frac{p_1 \xrightarrow{\lambda} q_1 \qquad p_2 \xrightarrow{\overline{\lambda}} q_2}{p_1|p_2 \xrightarrow{\tau} q_1|q_2}$$

SINCRONIZZAZIONE

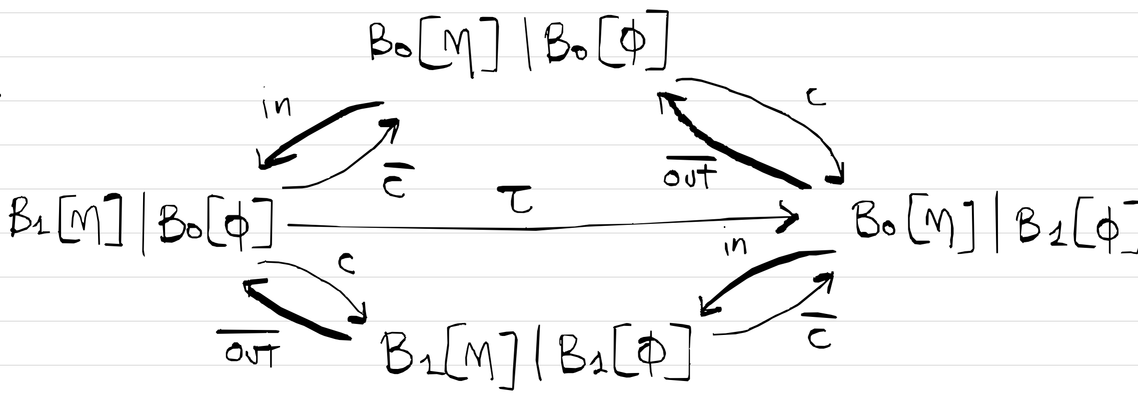
Se $\phi(\tau) = \tau$ e $\phi(\overline{\lambda}) = \overline{\phi(\lambda)}$, con $p[\phi]$ si indica il processo con cui si sostituiscono i label secondo ϕ.

$\rightsquigarrow$ $\phi(\mu_1) = \nu_1, \dots, \phi(\mu_e) = \nu_e$ sottintende che sugli altri simboli $\neq \overline{\mu_i}$ ϕ agisce come l'identità.

Esempio Se $B_0 \triangleq in.B_1$ e $B_1 \triangleq \overline{out}.B_0$ (buffer a cap. 1) e $\phi(in) = c$, $M(out) = c$, allora:

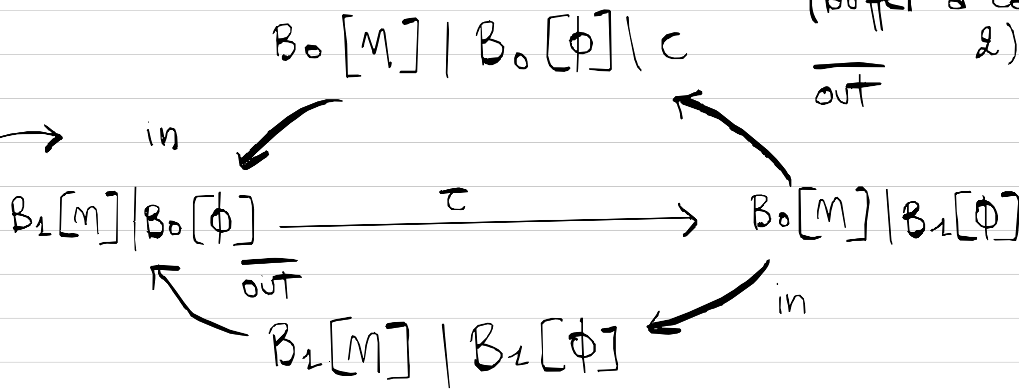


$\Longrightarrow$



(buffer a capacità 2)

$\rightsquigarrow$



Processi sorvegliati

Per limitarci a un numero finito di modi a cui si può giungere, imponiamo che i processi validi siano solo quelli sorvegliati ("guarded"), ossia quelli in cui le variabili sono prefisse da un'azione.

Esempio $p \triangleq rec\ x.\ p|x$ <u>non</u> è valido $(p \xrightarrow{N} q)$

$$p \xrightarrow{N} q|p \qquad p \xrightarrow{N} p|q|p \xrightarrow{N} p|p|q|p \; \cdots$$

(x <u>non</u> ha prefissi)

$$\begin{cases} G(nil, X) \triangleq true \\ G(x, X) \triangleq x \notin X \\ G(p.q, X) \triangleq G(q, \emptyset) \\ G(rec\ x.\ p, X) \triangleq G(p, X \cup \{x\}) \end{cases}$$

$$G(p+q, X) \triangleq G(p|q, X) \triangleq G(p, X) \wedge G(q, X)$$

$$G(p \backslash \alpha, X) \triangleq G(p, X)$$

$$\underbrace{P}_{\text{chiuso}} \text{ sorvegliato } \Longleftrightarrow G(P, \emptyset) \text{ vero.}$$

- $G(p, X \cup \{x\}) \Longrightarrow G(p, X)$
- $G(p, X) \wedge \bigwedge G(p_i, X) \Longrightarrow G(p[{}^{p_1}/_{x_1}, \dots, {}^{p_n}/_{x_n}], X)$
- $p \xrightarrow{N} q \wedge G(p, X) \Longrightarrow G(q, \emptyset)$
- i processi sorvegliati con sost. sorvegliate hanno finite transizioni.

Ci sono due nozioni intermedie di equivalenza:

ISOMORFISMO DI GRAFI	EQUIVALENZA PER TRACCIA
• Troppo concreta (e.g. i due buffer a cap. 2 non sono equiv.)	• Troppo astratta (e.g. i due distributori coin.(coffee + tea) e coin.coffee + coin.tea sono e quiv.)
$\varphi : \begin{array}{c} p \\ \downarrow N \\ q \end{array} \longmapsto \begin{array}{c} \varphi(p) \\ \downarrow \varphi(N) \\ \varphi(q) \end{array}$	$T(X) = \{ N_1 \dots N_m \mid \exists q \text{ t.c. } p \xrightarrow[N_1 \dots N_m]{} q \}$
$A \equiv_{iso} B \iff \exists \varphi$ iso. tra gli LTS	$A \equiv_{tr} B \iff T(A) = T(B)$

- $\equiv_{iso} \subsetneq \equiv_{tr}$
- serve una nozione media tra $\equiv_{iso}$ e $\equiv_{tr}$

Bisimilarità e bisimulazione (forte)

(Sia $\mathbb{P}$ l'insieme di processi generati col contesto corrente) → sorvegliati!

$R \subseteq \mathbb{P} \times \mathbb{P}$ si dice BISIMULAZIONE FORTE (b.f.) se

$$(p,q) \in R \implies \begin{cases} p \xrightarrow{N} p' \implies \exists q' \mid q \xrightarrow{N} q' \wedge (p',q') \in R \\ q \xrightarrow{N} q' \implies \exists p' \mid p \xrightarrow{N} p' \wedge (p',q') \in R \end{cases}$$

In altre parole p e q hanno le stesse transizioni e ricorsivamente anche i loro figli con i label concordi.

- id è una bisimulazione forte,
- $\{(p, f(p))\}$ è " se f è un isomorfismo tra grafi,

($R';R$ o $R'R$)

- $R \cup R'$ e $R \circ R'$ sono b.f. $\Longleftarrow$ R, R' b.f.
- R^{-1} b.f. $\Longleftarrow$ R b.f.

$$p \simeq q \quad \overset{def}{\Longleftrightarrow} \quad \exists R \text{ b.f. t.c. } (p,q) \in R$$

$$\Updownarrow$$

$$\underbrace{(p,q) \in \bigcup_{R\ b.f.} R}_{=\ \simeq}$$

→ la bisimilarità forte è la più grande bisim. forte

BISIMILARITÀ FORTE (rel. di equiv.) → è una **CONGRUENZA**

La bisimilarità è il <u>massimo</u> dei post-punti fissi di: (con $\subseteq$)

$$\Phi : 2^{\mathcal{P}\times\mathcal{P}} \longrightarrow 2^{\mathcal{P}\times\mathcal{P}}$$

$$R \longmapsto \left\{ (p,q) \in \mathcal{P}\times\mathcal{P} \;\middle|\; \begin{array}{l} p \xrightarrow{N} p' \Rightarrow \exists q' \mid q \xrightarrow{N} q' \\ \qquad \wedge\ (p',q') \in R \\ q \xrightarrow{N} q' \Rightarrow \exists p' \mid p \xrightarrow{N} p' \\ \qquad \wedge\ (p',q') \in R \end{array} \right\}$$

Poiché è il massimo, ordinando $2^{\mathcal{P}\times\mathcal{P}}$ con $\supseteq$ (<u>non</u> $\subseteq$!), diventa il minimo $\Longrightarrow$ Teorema di Kleene.
Infatti Φ è sempre monotona, ed è continua su process. finitamente ramificati ($\Leftarrow$ sorvegliati).

Lemma Φ è continua se i processi sono finitamente ramificati.

(l'altra direz. è data dalla monotonia)

Dobbiamo mostrare che $\Phi\left(\bigcap_{i\in\mathbb{N}} R_i\right) \supseteq \bigcap_{i\in\mathbb{N}} \Phi(R_i)$.

Sia $(p,q) \in \Phi(R_i)$ $\forall i \in \mathbb{N}$. Se $p \xrightarrow{N} p'$, allora $\forall i$ $\exists q_i \mid q \xrightarrow{N} q_i$ e $(p', q_i) \in \Phi(R_i)$. In particolare, poiché i processi sono finitam. ramificati, $\{q' \mid q \xrightarrow{N} q'\}$ è finito, quindi per il principio della piccionaia $\exists m \in \mathbb{N} \mid \{ n \mid q_n = q_m \}$ è infinito.
Quindi $(p', q_m) \in \bigcap_{i\in\mathbb{N}} R_i$ (per i $\exists j \geq i$ ($R_i \supseteq R_j$) con cui $(p', q_m) \in R_j \subseteq R_i$). ∎
$\Rightarrow (p,q) \in \Phi\left(\bigcap_{i\in\mathbb{N}} R_i\right)$

Quindi in tal caso...

$$\simeq \; = \bigcap_{i \in \mathbb{N}} \Phi^i(\mathbb{P} \times \mathbb{P})$$

$\rightsquigarrow$ per processi inf. ramificati si ha a disp. in modo teorico il teorema di Knaster–Tarski.

Logica di Hennessy-Milner (HML)

$p \models F \overset{def}{\iff} p$ rispetta F

$$
\begin{aligned}
F, G ::= \; & tt && \text{true} \\
| \; & ff && \text{false} \\
| \; & \bigwedge_i F_i && \text{and} \\
| \; & \bigvee_i F_i && \text{or} \\
| \; & \Box_N F && p \models \Box_N F \overset{def}{\iff} \forall p' \text{ t.c. } p \xrightarrow{N} p',\; p' \models F \\
| \; & \Diamond_N F && p \models \Diamond_N F \overset{def}{\iff} \exists p' \text{ t.c. } p \xrightarrow{N} p',\; p' \models F
\end{aligned}
$$

$\forall p,\; p \models tt$ e $p \not\models ff$.

Non è costruita esplicitamente "$\neg F$", lo si fa induttivamente

$(tt)^c = ff$, $\quad (\bigwedge_i F_i)^c = \bigvee_i F_i^c$, $\quad (\Box_N F)^c = \Diamond_N F^c$

$(ff)^c = tt$, $\quad (\bigvee_i F_i)^c = \bigwedge_i F_i^c$, $\quad (\Diamond_N F)^c = \Box_N F^c$.

$\rightsquigarrow \; p \models \Diamond_\alpha tt \iff \exists p'$ con $p \xrightarrow{\alpha} p'$.

$\rightarrow$ si estende la sintassi con $\Diamond_{\{\alpha_1, \dots, \alpha_m\}} \triangleq \bigvee_i \Diamond_{\alpha_i}$

e $\Box_{\{\alpha_1, \dots, \alpha_m\}} \triangleq \bigwedge_i \Box_{\alpha_i}$, così:

$$p \models \Diamond_{\{\alpha_1, \dots, \alpha_m\}} tt \iff \exists \alpha_i, p' \text{ t.c. } p \xrightarrow{\alpha_i} p'.$$

$$(\Diamond_\emptyset F \triangleq ff, \quad \Box_\emptyset F \triangleq tt)$$

$$p \equiv_{HM} q \iff \forall F\, (p \models F \iff q \models F)$$

Teorema Per proc. finit. ramificati, $\equiv_{HM}$ e $\simeq$ coincidono

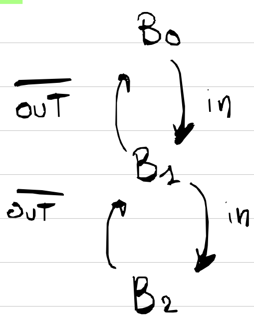
Quindi:

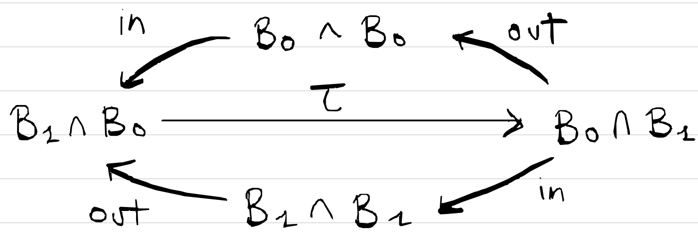
- per dim. $p \simeq q$, si trova una bisimulazione forte
- per dim. $p \not\simeq q$, si trova F t.c. $p \models F$ e $q \not\models F$ o viceversa

Bisimilarità debole

La b. forte **non** tiene conto delle sincronizzazione con τ!

Esempio





A

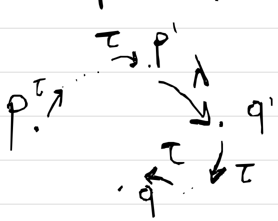
B

A e B **non** sono fortemente bisimili, ma "svolgono la stessa funzione di buffer da capacità 2 $\longrightarrow$ "c'è una τ di diff."

le τ sono silenziose e rappresentano "procedure interne del sistema"

$\rightsquigarrow \quad p \overset{\tau}{\Longrightarrow} q \quad \overset{def}{\Longleftrightarrow} \quad p = q \lor p \xrightarrow{\tau} \cdots \xrightarrow{\tau} q$ con $n > 0$ τ

$\rightsquigarrow \quad p \overset{\lambda}{\Longrightarrow} q \quad \overset{def}{\Longleftrightarrow} \quad \exists p', q' \mid p \overset{\tau}{\Longrightarrow} p' \xrightarrow{\lambda} q' \overset{\tau}{\Longrightarrow} q$



Una **BISIMULAZIONE DEBOLE** R è t.c.

$$\forall (p,q) \in R, \begin{cases} p \xrightarrow{N} p' \Rightarrow \exists q' \mid q \overset{N}{\Longrightarrow} q' \\ q \xrightarrow{N} q' \Rightarrow \exists p' \mid p \overset{N}{\Longrightarrow} p' \end{cases}$$

Il concetto si estende naturalmente a quello di BISIMILARITÀ DEBOLE. E' ancora rel. di equiv. e p.to fisso dell'analogo di Φ per b. forte. Si indica con $\approx$.

→ i processi sorvegliati possono essere infinitamente ramificati sulle $\Longrightarrow$!

→ la b. debole non è una congruenza!

(Non distingue stalli e divergenze silenziose)

$$nil \approx rec\ x.\ \tau.x$$

Presa $\underset{=}{\approx}$ t.c.

$$p \underset{=}{\approx} q \overset{def}{\Longleftrightarrow} p \approx q \wedge \underbrace{\forall r,\ p+r \approx q+r}_{\text{ipotesi minimale}}$$

allora $\underset{=}{\approx}$ è una congruenza! ($\underset{=}{\approx} \subseteq \approx$)

ed è la più grande congruenza in $\approx$