Processi e scambio di messaggi

L'idea di base è che un processo può evolvere un passo alla volta.

Le interazioni del processo col mondo esterno sono limitate a scambio di messaggi

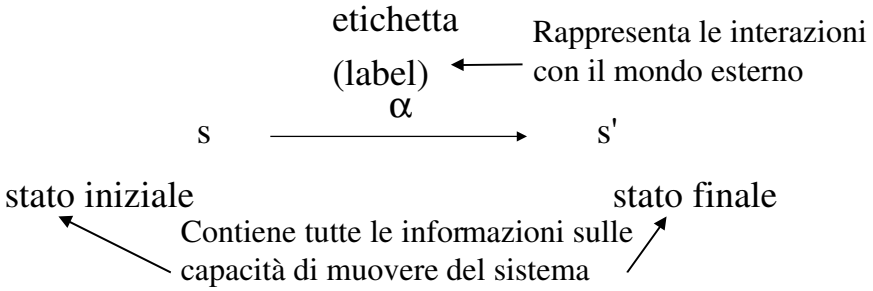
Ci interessa solo l'aspetto osservazionale, cioè quali influenze ha il processo sul/dal mondo esterno.

Quindi a livello astratto ci interessa sapere solo

- Quali capacità di evoluzione ha un processo in un dato momento
- Quali interazioni ha il processo con il mondo esterno in ogni passo della sua evoluzione

Transizioni

Rappresentiamo ciascun passo di evoluzione di un processo col concetto di transizione



Un processo corrisponderà, quindi, alle sue possibili evoluzioni a partire dal suo stato iniziale.

Il modo più conveniente di rappresentarle è mediante un albero in cui ciascun cammino rappresenti un possibile percorso evolutivo del processo.

Alberi di Transizione Etichettati

Un albero rappresenta un processo.

I nodi sono etichettati con gli stati e gli archi con le etichette.

Ciascun arco corrisponde ad una possibile transizione del sistema.

Non tutti gli alberi così etichettati sono rappresentazioni di processi.

Esempio tipico di cose che non possono accadere è l'esistenza di due nodi con la stessa etichetta ma insiemi diversi di figli (vorrebbe dire che lo stato non determina in modo univoco le capacità di evoluzione).

Il modo più compatto e semplice di descrivere come sono fatti gli alberi di transizione è usando i sistemi di transizione.

Sistemi di Transizione Etichettati
Labelled Transition System

Un sistema di transizione etichettato consiste di

- un insieme S di stati (tutti i possibili stati di tutti i processi)
- un insieme L di etichette (tutti i possibili messaggi scambiati dai processi)
- una relazione di transizione $\rightarrow \subseteq S \times L \times S$

Dato un sistema di transizione etichettato, un albero di transizione è un albero (possibilmente infinito in profondità e in larghezza), in cui

- i nodi sono etichettati con stati e gli archi con etichette
- da un nodo s esce un arco etichettato con l che arriva in un nodo s' se e solo se $(s,l,s') \in \rightarrow$ (indicato $s \xrightarrow{l} s'$)
- non si tiene conto dell'ordine dei figli e non ci sono due figli uguali di una stessa radice

X-CCS

Calculus of Concurrent Systems: una famiglia di linguaggi (molte varianti studiate) per descrivere processi concorrenti basati su scambio di messaggi.

Introdurremo i costrutti un po' alla volta e ne daremo la semantica attraverso un sistema di transizione.

$E ::= nil \mid Label.E \mid \Sigma_{i \in Ind} E_i$

Ind ::= …sintassi per insiemi di indici…

Label ::= …sintassi per le etichette…

nil non è capace di transizioni, quindi non ci sono regole per nil.

$\alpha.p$ è il processo capace di spedire α e poi diventare il processo p

$$\frac{}{\alpha.p \xrightarrow{\alpha} p}$$

$\Sigma_{i \in I} p_i$ è la scelta non deterministica fra i vari processi p_i quindi ha la capacità di muoversi come ogni p_i

Se I è infinito, questo può generare alberi infiniti in larghezza

$$\frac{p_i \xrightarrow{\alpha} p'}{\Sigma_{i \in I} p_i \xrightarrow{\alpha} p'}$$

Se I è vuoto la premessa non può essere provata, quindi $\Sigma_{i \in \varnothing} p_i$ equivale a nil

Parallelismo in CCS

Fino a questo momento i processi non possono interagire, aggiungiamo un costrutto per il parallelismo: $E ::= \ldots E \| E \ldots$

Gli scambi di messaggi fra p_1 e p_2 non sono più influenze sul mondo esterno rispetto al sistema complessivo $p_1 \| p_2$.

Bisogna quindi dettagliare meglio le etichette, per catturare due idee: la differenza fra messaggi interni ed esterni al sistema e cosa vuol dire scambio di messaggi.

Fissiamo un insieme A di etichette che corrispondono alle azioni che ci interessano, ad esempio ci saranno send, insert_coin, select_coffee.... Per ciascuna esiste il punto di vista complementare (receive, get_coin,selected_coffee..), che è anch'esso un'azione compiuta da un processo del sistema, cioè c'è un insieme $\overline{A}$ che contiene le azioni complementari.

Inoltre, se p_1 e p_2 si scambiano un messaggio, cioè p_1 fa α e p_2 fa $\overline{\alpha}$, allora il sistema complessivo fa un'azione interna, che non ha nessun effetto esterno, indicato da un'etichetta speciale τ.

Label ::= …sintassi per $\{\tau\} \cup A \cup \overline{A}$. Nel seguito $Act = A \cup \overline{A}$.

Semantica del Parallelismo in CCS

Lo scambio di messaggi in CCS corrisponde ad una sincronizzazione (handshaking): i due processi coinvolti compiono simultaneamente azioni complementari

$$\frac{p_1 \xrightarrow{\alpha} p'_1 \quad p_2 \xrightarrow{\overline{\alpha}} p'_2}{p_1 \| p_2 \xrightarrow{\tau} p'_1 \| p'_2} \quad \alpha \in Act$$

Con la convenzione che $\overline{\overline{\alpha}} = \alpha$

Oltre a scambio di messaggi, i processi dovrebbero mantenere le loro capacità di compiere azioni quando messi in parallelo.

Questo può essere fatto in vari modi.

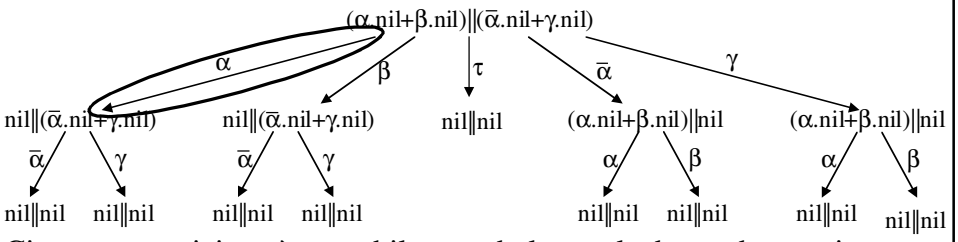
Vediamo per primo l'interleaving.

$$\frac{p_1 \xrightarrow{\alpha} p'_1}{p_1 \| p_2 \xrightarrow{\alpha} p'_1 \| p_2} \qquad \frac{p_2 \xrightarrow{\alpha} p'_2}{p_1 \| p_2 \xrightarrow{\alpha} p_1 \| p'_2}$$

Queste regole corrispondono all'esecuzione di uno solo dei due processi, quindi non è vero parallelismo

Esempio

Vediamo ad esempio l'albero di transizione per $(\alpha.nil+\beta.nil)\|(\overline{\alpha}.nil+\gamma.nil)$



Ciascuna transizione è provabile usando le regole date, ad esempio:

$$\frac{\dfrac{\dfrac{(\alpha.nil+\beta.nil)\|(\overline{\alpha}.nil+\gamma.nil) \xrightarrow{\alpha} nil\|(\overline{\alpha}.nil+\gamma.nil)}{(\alpha.nil+\beta.nil) \xrightarrow{\alpha} nil}}{\alpha.nil \xrightarrow{\alpha} nil}}{}$$

Lo "stesso" albero si può ottenere partendo da un termine senza ||

$\alpha.(\overline{\alpha}.nil + \overline{\gamma}.nil) + \beta.(\alpha.nil+\gamma.nil) + \overline{\alpha}.(\alpha.nil + \overline{\beta}.nil) + \gamma.(\alpha.nil+\beta.nil) + \tau.nil$

Questa proprietà è vera in generale e prende il nome di teorema di espansione

Altri tipi di parallelismo: sincronia totale

Si cambiano le regole di comunicazione ed esecuzione parallela

Nessun processo è mai fermo; le azioni avvengono simultaneamente (una per processo in ogni unità di tempo)

$$\frac{p_1 \xrightarrow{\alpha} p'_1 \quad p_2 \xrightarrow{\beta} p'_2}{p_1 \| p_2 \xrightarrow{\alpha \bullet \beta} p'_1 \| p'_2}$$

Perché la regola abbia senso bisogna avere un'operazione binaria $_\bullet_$ sulle etichette.

Perché abbia senso usare per $_\bullet_$ la notazione infissa deve essere associativa (cioè $(Act, _\bullet_)$ è un semigruppo)

Diverse interpretazioni di $_\bullet_$ corrispondono a diversi tipi di comunicazione, ad esempio si recupera la comunicazione di prima introducendo τ e imponendo relazioni fra composizione e complemento

$\overline{\alpha \bullet \beta} = \overline{\alpha} \bullet \overline{\beta}$ (la sequenza dei complementari è il complementare della sequenza)

$\overline{\overline{\alpha}} = \alpha$ e $\overline{\tau} = \tau$ (come prima) e $\alpha \bullet \overline{\alpha} = \tau$

Se si vuole che il parallelo sia commutativo, bisogna anche che $\alpha \bullet \beta = \beta \bullet \alpha$

Se si vuole che τ non sia visibile (come sembra ragionevole) $\alpha \bullet \tau = \tau \bullet \alpha = \alpha$

In altre parole $(\{\tau\} \cup Act, \tau, \overline{_}, _\bullet_)$ deve formare un gruppo abeliano

Sincronia totale e deadlock

Con le regole date finora $\alpha.nil \| nil$ non si riesce a muovere (perché non c'è nessun p' per cui $nil \rightarrow p'$)

Prima soluzione: immaginiamo che un processo che non fa nulla sia osservazionalmente equivalente ad un processo che fa un'azione interna che lo riporta nello stesso stato:

$$\frac{}{nil \xrightarrow{\tau} nil}$$

Non abbiamo però più modo di distinguere fra un processo fermo (idle) e uno in loop interno infinito.

Seconda soluzione: aggiungiamo regole per far muovere un solo pezzo del parallelo quando l'altro ha finito

$$\frac{p_1 \xrightarrow{\alpha} p'_1}{p_1 \| nil \xrightarrow{\alpha} p'_1 \| nil} \qquad \frac{p_2 \xrightarrow{\alpha} p'_2}{nil \| p_2 \xrightarrow{\alpha} nil \| p'_2}$$

Questo non è sufficiente, perché ci sono altri processi idle che non sono della forma $nil(\|nil\| \ldots \|nil)$, ad esempio $\alpha.nil \|(nil+nil)$ non riesce a muovere.

Bisogna dare regole che coprano tutte le forme sintattiche possibili.

Il modo più semplice è definire un predicato idle sui processi (che sia vero se il processo non può muovere) e le regole

$$\frac{p_1 \xrightarrow{\alpha} p'_1}{p_1 \| p_2 \xrightarrow{\alpha} p'_1 \| p_2} \; idle(p_2) \qquad \frac{p_2 \xrightarrow{\alpha} p'_2}{p_1 \| p_2 \xrightarrow{\alpha} p_1 \| p'_2} \; idle(p_1)$$

Restrizione e ridenominazione

Aggiungiamo un costrutto per la restrizione delle azioni possibili

(si usa per forzare la sincronizzazione fra due parti di un sistema impedendo di usare quella capacità come comunicazione con l'esterno).

Sintassi: E\L, dove L è un sottoinsieme di Act

Semantica: data dalla regola:

$$\frac{p \xrightarrow{\alpha} p'}{p\backslash L \xrightarrow{\alpha} p'\backslash L} \quad \alpha \notin L$$

Aggiungiamo un costrutto per la rietichettatura dei processi

Sintassi: E[f], dove f è una funzione di relebelling, cioè una funzione unaria su $\{\tau\} \cup Act$ che soddisfa due proprietà:

$f(\tau)=\tau$ e $f(\overline{\alpha}) = \overline{f(\alpha)}$

Semantica: data dalla regola

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

Ricorsione/Iterazione

Finora non abbiamo costrutti che introducano processi con evoluzione infinita (non abbiamo né ricorsione, né iterazione).

Per semplificare le regole da dare, introduciamo i processi di questo tipo come se fossero dichiarazioni di nomi di processi:

Sintassi: $[fix_j]\{x_i = p_i \mid i \in I\}$ dove $j \in I$ e dove in ogni p_i possono comparire le varie x_j

Semantica: si aggiunge una regola come la seguente per ogni fix che si usa nel sistema (le varie x_i sono nel sistema associate ai processi soluzione dell'equazione di punto fisso)

$$\frac{p_j \xrightarrow{\alpha} p'}{x_j \xrightarrow{\alpha} p'} \quad \{x_i = p_i \mid i \in I\}$$

Alternativamente si dà la semantica del costrutto nel caso generale:

$$\frac{p_j[fix_k\{x_i = p_i \mid i \in I\}/x_k] \xrightarrow{\alpha} p'}{fix_j\{x_i = p_i \mid i \in I\} \xrightarrow{\alpha} p'}$$

Esempio con ricorsione

Consideriamo l'albero per il termine p che rappresenta un buffer di lunghezza 1 a valori booleani (inizialmente vuoto)

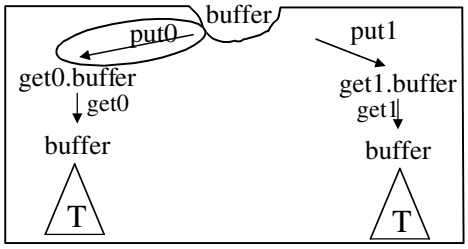
Aggiungo al sistema l'uso di un fix:

{buffer = put0.get0.buffer+put1.get1.buffer}

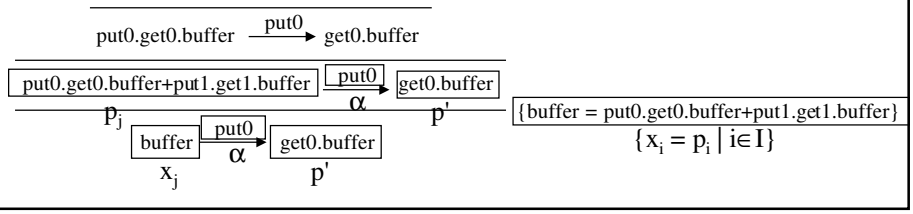
La cosa è possibile solo perché T è infinito.

Di solito si disegna di solito l'albero con l'unfolding dei cicli fino al livello necessario e poi …

Vediamo dove si usa la regola per il fix



T



Semantica dei processi

In prima battuta si potrebbe pensare che la semantica di un termine CCS sia l'albero di transizione che ha quel termine come (etichetta della) radice.

Però questa scelta non corrisponde all'idea che per decidere se due processi sono equivalenti ci interessano solo le influenze sul mondo esterno.

Addirittura nil e nil+nil hanno alberi diversi perché l'etichetta della radice è diversa.

Seconda ipotesi (semantica a tracce): due termini CCS sono equivalenti (= hanno la stessa semantica, = rappresentano lo stesso processo) se hanno lo stesso insieme di tracce.

Una traccia di un termine p è la stringa (eventualmente infinita) delle etichette degli archi che visito in un cammino nell'albero di transizione associato a p, dalla radice ad una delle foglie.

Ad esempio, le tracce per l'esempio del buffer sono (stringhe infinite in cui il primo elemento è un put, dopo un put avviene il get dello **stesso** simbolo e dopo un get avviene un put)

$\{f:\mathbb{N}\to\{put0,get0,put1,get0\} \mid f(0)\in\{put0, put1\} \wedge$

$(f(i)= put0 \Rightarrow f(i+1)= get0) \wedge (f(i)= put1 \Rightarrow f(i+1)= get1)$

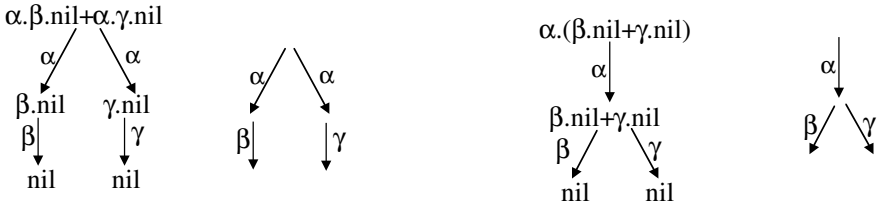
$\wedge (f(i)\in\{get0, get1\} \Rightarrow f(i+1)\in\{put0, put1\}) \}$

Dire che due termini sono equivalenti se hanno le stesse tracce corrisponde in molti casi alla nostra intuizione, ma non tiene conto del punto in cui avviene una scelta.

Limiti della semantica a tracce

Consideriamo i due termini $\alpha.(\beta.nil+\gamma.nil)$ e $\alpha.\beta.nil+ \alpha.\gamma.nil$.

Hanno le stesse tracce ($\{\alpha.\beta, \alpha.\gamma\}$) però gli alberi di transizione hanno forme diverse:



Nel primo caso dopo aver fatto α si è obbligati a fare γ o β a seconda del ramo scelto, mentre nel secondo dopo aver fatto α si ha ancora la possibilità di scegliere fra γ e β.

Quindi se si mettono in un contesto in cui dopo non si può fare β il primo può andare in deadlock (etichetta di una foglia che non è uno stato finale), il secondo no:

$\alpha.(\beta.nil+\gamma.nil)\backslash\{\beta\}$ e $(\alpha.\beta.nil+\alpha.\gamma.nil)\backslash\{\beta\}$

(provare a farne gli alberi per esercizio)

Vogliamo una semantica che distingua questi casi.

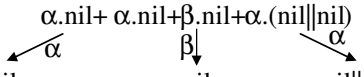
Alberi di sincronizzazione

Vogliamo dire che due termini sono uguali se hanno gli stessi alberi di sincronizzazione.

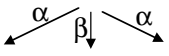
Un albero di sincronizzazione è un albero di transizione, in cui si sono tolte le etichette dei nodi e si sono eliminati eventuali doppioni.

Vediamo un esempio di calcolo: $\alpha.nil+ \alpha.nil+\beta.nil+\alpha.(nil\|nil)$

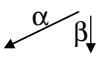
Prima di tutto si calcola l'albero di transizione del termine:



Poi si cancellano gli stati



Infine si eliminano eventuali doppioni



Scegliere come semantica dei termini i loro alberi di sincronizzazione, va decisamente meglio (ed è la soluzione finale adottata nel corso)

Piccolo problema: se l'albero è infinito calcolarlo e manipolarlo non è banale.

Ci vuole una tecnica di prova che ci permetta di decidere in modo finito se due termini hanno la stessa semantica anche se il loro behaviour è infinito.

Semantica di bisimulazione

Vogliamo una tecnica di prova che ci permetta di identificare due termini con le stesse capacità di evoluzione.

Per un singolo passo di evoluzione questo vuol dire che possono fare le stesse mosse, cioè mosse con le stesse etichette che li portano in termini equivalenti (cioè con le stesse capacità di evoluzione).

Quindi vogliamo una relazione binaria (di equivalenza) R sui termini che soddisfi la seguente proprietà:

Per ogni etichetta α, per ogni coppia di termini CCS p e q

p R q se e solo se

$p \xrightarrow{\alpha} p'$ implica che esiste q' tale che $q \xrightarrow{\alpha} q'$ e p' R q'

anche q ha la capacità di fare α gli stati di arrivo sono equivalenti

e viceversa $q \xrightarrow{\alpha} q'$ implica che esiste p' tale che $p \xrightarrow{\alpha} p'$ e p' R q'

Le proprietà richieste ci dicono che q è in grado disimulare il comportamento di p e viceversa, cioè sono bi-simili.

Questa però è una proprietà non una definizione.

Nel prossimo lucido vedremo come definire una relazione che gode di questa proprietà (guadagnando una tecnica di prova facile gratis).

Bisimulazione

La ragione per cui la proprietà richiesta non è una definizione è che R compare da entrambi i lati del se e solo se.

Per ovviare questo problema per prima cosa definiamo un costruttore di relazioni $\mathcal{F}$:

Data una relazione R fra termini CCS, la relazione $\mathcal{F}(R)$ è definita da:

$\forall p,q \in L_E(CCS)$, p $\mathcal{F}(R)$ q $\Leftrightarrow$

1) $\forall p' \in L_E(CCS)$ $p \xrightarrow{\alpha} p'$ $\exists q' \in L_E(CCS)$, $q \xrightarrow{\alpha} q' \wedge$ p' R q'
2) $\forall q' \in L_E(CCS)$ $q \xrightarrow{\alpha} q'$ $\exists p' \in L_E(CCS)$, $p \xrightarrow{\alpha} p' \wedge$ p' R q'

Una relazione è una bisimulazione se e solo se $R \subseteq \mathcal{F}(R)$.

Quindi se R è una bisimulazione $\forall p,q \in L_E(CCS)$ p R q ($\Rightarrow$ p $\mathcal{F}(R)$ q) $\Rightarrow$ 1) e 2) ma potrebbero esserci dei p,q che soddisfano 1) e 2) senza essere in relazione con R.

A noi in realtà interessa una bisimulazione abbastanza "grande" da valere R = $\mathcal{F}(R)$, cioè un punto fisso di $\mathcal{F}$.

Vogliamo dimostrare che facendo l'unione di tutte le bisimulazioni, si ottiene una relazione, detta bisimulazione forte che è ancora una bisimulazione ed è il massimo punto fisso di $\mathcal{F}$.

Bisimulazione Forte

Lemma 1 (prova per esercizio):

L'unione su un insieme arbitrario di indici I di bisimulazioni R_i è una bisimulazione.

Lemma 2 (prova per esercizio):

$\mathcal{F}$ è un operatore monotono, cioè $R_1 \subseteq R_2$ implica $\mathcal{F}(R_1) \subseteq \mathcal{F}(R_2)$

Sia $\sim = \cup_{R \text{ bisimulazione}} R$ detta bisimulazione forte .

Teorema 1 $\sim = \mathcal{F}(\sim)$.

Prova

1) Per il lemma 1 $\sim$ è una bisimulazione, cioè $\sim \subseteq \mathcal{F}(\sim)$.

2) Allora, per il lemma 2 $\mathcal{F}(\sim) \subseteq \mathcal{F}(\mathcal{F}(\sim))$, cioè $\mathcal{F}(\sim)$ è una bisimulazione.

Per cui, per definizione di $\sim$, $\mathcal{F}(\sim) \subseteq \sim$.

Teorema 2 $\sim$ è il massimo punto fisso di $\mathcal{F}$.

Prova

Sia R un punto fisso di $\mathcal{F}$, cioè $R = \mathcal{F}(R)$.

Allora $R \subseteq \mathcal{F}(R)$ cioè R è una bisimulazione e quindi per definizione $R \subseteq \sim$.

Prove di Bisimulazione Forte

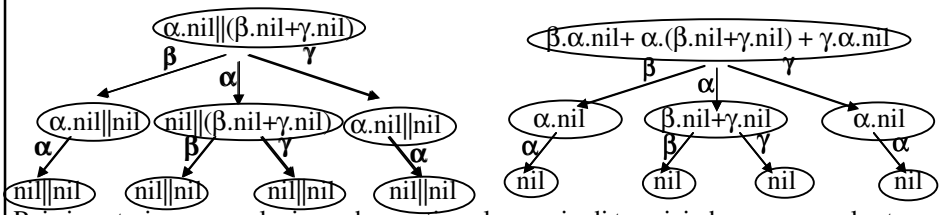
Per provare che due termini sono in bisimulazione forte, per definizione di bisimulazione, basta (ed è l'unico modo) provare che esiste una relazione R che li contiene e che è una bisimulazione.

Vediamo un paio di esempi concreti.

Caso facile: i due termini hanno alberi di transizione finiti.

Consideriamo $\alpha.nil \| (\beta.nil+\gamma.nil)$ e $\beta.\alpha.nil+ \alpha.(\beta.nil+\gamma.nil) + \gamma.\alpha.nil$

Per prima cosa si costruiscono gli alberi di transizione



Poi si costruisce una relazione che contiene le coppie di termini che occupano le stesse posizioni nell'albero

$R = \{(\alpha.nil \| (\beta.nil+\gamma.nil), \beta.\alpha.nil+ \alpha.(\beta.nil+\gamma.nil) + \gamma.\alpha.nil), (\alpha.nil \| nil, \alpha.nil)$
$(nil \| (\beta.nil+\gamma.nil), \beta.nil+\gamma.nil), (nil \| nil, nil)\}$

Ultimo passo si dovrebbe provare che R è effettivamente una relazione (ma dati gli alberi è una banalità)