SOS a piccoli passi

Invece di semplificare sempre in configurazioni finali, fattorizziamo in passi più piccoli (da cui il nome).

In questo modo la non terminazione corrisponde a una catena infinita di semplificazioni.

Inoltre abbiamo più dettagli da fissare e quindi possiamo stabilire proprietà di livello più basso (e.g. ordine di esecuzione)

A ogni produzione della grammatica corrispondono più metaregole:

- una per l'elaborazione parziale di ogni sottoespressione

$$\frac{(e,s) \square_r (e',s')}{(x=e,s) \square_r (x=e',s')}$$

- una per l'elaborazione del costrutto dopo aver semplificato tutte le sottoespressioni

$$\frac{}{(x=v,s) \square_r (v,s[v/r(x)]))}$$

Quest'ultima compare anche nel caso a grandi passi per gestire i termini le cui sottoespressioni sono già valori (ma in questo caso verrà applicata anche negli alberi corrispondenti a termini le cui sottoespressioni sono complesse)

SOS a piccoli passi per le espressioni

$$\frac{(e,s) \square_r (e',s')}{(x{=}e,s) \square_r (x{=}e',s')}$$

$$\frac{}{(x{=}v,s) \square_r (v,s[v/r(x)])}$$

$$\frac{(es,s) \square_r (es',s')}{(f(es),s) \square_r (f(es'),s')}$$

$$\frac{}{(f(lv),s) \square_r r(f)(lv,s)}$$

$$\frac{(e,s) \square_r (e',s')}{(e\ es,\ s) \square_r (e'\ es,s')}$$

$$\frac{(es,s) \square_r (es',s')}{(v\ es,\ s) \square_r (v\ es',s')}$$

$$r(x) \square Loc_T \quad \frac{}{(x,s) \square_r (s(r(x)),s)}$$

Non essendoci sottoespressioni semplificabili c'è una sola regola

$$\frac{(e,s) \square_r (e_0,s_0)}{(e+e',s) \square_r (e_0+e',s_0)}$$

$$\frac{(e,s) \square_r (e',s')}{(v+e,s) \square_r (v+e',s')}$$

$$\frac{}{(v+v',s) \square_r (a,s'')} \quad v + v' = a$$

Queste regole corrispondono a valutazioni da sinistra verso destra.

Provare per esercizio a dare altre strategie di valutazione.

Mancano tutte le condizioni a lato relative al tipaggio delle metavariabili.

Aggiungerle per esercizio

Statements 1

$$\frac{(st,\ s)\ \square_{\ r}\ (st',\ s')}{(st\ sts,\ s)\ \square_{\ r}\ (st'\ sts,\ s')}$$

$$\frac{(st,\ s)\ \square_{\ r}\ (\square,\ s')}{(st\ sts,\ s)\ \square_{\ r}\ (sts,\ s')}$$

$$\frac{(e,\ s)\ \square_{\ r}\ (e',\ s')}{(e;,\ s)\ \square_{\ r}\ (e';,\ s')}$$

$$\frac{}{(v;,\ s)\ \square_{\ r}\ (\square,\ s)}$$

$$\frac{(e,\ s)\ \square_{\ r}\ (e',\ s')}{(\mathbf{if}\ (e)\ \{sts\}\ \mathbf{else}\ \{sts'\},\ s)\ \square_{\ r}\ (\mathbf{if}\ (e')\ \{sts\}\ \mathbf{else}\ \{sts'\},\ s')}$$

$$\frac{}{(\mathbf{if}\ (false)\ \{sts\}\ \mathbf{else}\ \{sts'\},\ s)\ \square_{\ r}\ (sts',\ s)}$$

$$\frac{}{(\mathbf{if}\ (true)\ \{sts\}\ \mathbf{else}\ \{sts'\},\ s)\ \square_{\ r}\ (sts,\ s)}$$

$$\frac{(e,\ s)\ \square_{\ r}\ (e',\ s')}{(\mathbf{while}\ (e)\ \{sts\},\ s)\ \square_{\ r}\ (\mathbf{while}\ (e')\ \{sts\},s')}$$

$$\frac{}{(\mathbf{while}\ (false)\ \{sts\},\ s)\ \square_{\ r}\ (\square,\ s)}$$

$$\frac{}{(\mathbf{while}\ (true)\ \{sts\},\ s)\ \square_{\ r}\ (sts\ \mathbf{while}\ (e)\ \{sts\},\ s)}$$

Non funziona perché ci siamo persi la guardia

Statements 2

L’intuizione di partenza per la semantica del while era l’equivalenza

while (e) {sts} = **if** (e) {sts **while** (e) {sts}}

che sfortunatamente non è esprimibile nel nostro linguaggio (manca l’else).

Manipoliamo la grammatica in modo che diventi possibile.

Questo non vuol dire che cambiamo il linguaggio (non si può) ma che cambiamo le configurazioni, ammettendo termini su un linguaggio più esteso:

$$L = \bigcup_{s \in N} L_s(LW+) \cup \{\varepsilon\}$$

Dove LW+ è l’estensione di LW mediante

- o un costrutto skip, la cui semantica è la funzione identica (lo stato non cambia)
- o un costrutto if senza ramo else

Stat ::= …| **skip**;

$$\frac{}{(\textbf{skip};, s) \rightarrow_r (\varepsilon, s)}$$

$$\frac{}{(\textbf{while}\ (e)\ \{sts\}, s) \rightarrow_r (\textbf{if}\ (e)\ \{sts\ \textbf{while}\ (e)\ \{sts\}\}\ \textbf{else}\ \{\textbf{skip};\}, s)}$$

Stat ::= …| **if** (Exp) {Stats}

$$\frac{(e, s) \rightarrow_r (e', s')}{(\textbf{if}\ (e)\ \{sts\}, s) \rightarrow_r (\textbf{if}\ (e')\ \{sts\}, s')}$$

$$\frac{}{(\textbf{if}\ (false)\ \{sts\}, s) \rightarrow_r (\varepsilon, s)}$$

$$\frac{}{(\textbf{while}\ (e)\ \{sts\}, s) \rightarrow_r (\textbf{if}\ (e)\ \{sts\ \textbf{while}\ (e)\ \{sts\}\}, s)}$$

$$\frac{}{(\textbf{if}\ (true)\ \{sts\}, s) \rightarrow_r (sts, s)}$$

SOS a piccoli passi per le dichiarazioni di variabili

$$\frac{\text{Nuova}(l,r,s) \quad x \square L_{Id}(LW) \quad t \square L_{Type}(LW)}{(t\ x,r,s) \ \square \ (\square,\ r[l/x],s[\square/l])}$$

$$\frac{(e,s) \ \square_{\ r} \ (e',s')}{(t\ x = e,r,s) \ \square \ (t\ x = e',r,s')}$$

$$\frac{\text{Nuova}(l,r,s) \quad v \square \text{Value} \quad x \square L_{Id}(LW) \quad t \square L_{Type}(LW)}{(t\ x = v,r,s) \ \square \ (\square,r[l/x],s[v/l])}$$

$$\frac{(d,r,s) \ \square \ (d',r',s')}{(d\ ds,r,s) \ \square \ (d'\ ds,r',s')}$$

$$\frac{(d,r,s) \ \square \ (\square,r',s')}{(d\ ds,r,s) \ \square \ (ds,r',s')}$$

SOS a piccoli passi per dichiarazioni di funzioni

(finta)

Prima possibilità: fare (per quanto possibile) come nella SOS a grandi passi

…bla bla….dove $F : [\![T]\!] \square States \square \{*\} \square States$
è definita induttivamente da tutte le regole della semantica più le seguenti:

$$Nuova(l,r,s_c)\quad \frac{(sts,\ s_c[v/l]) \ \square^{*}_{r[\square/f,l/x]}\ (\square,\ s')}{F(v,s_c)\ = (*,s'-\{l\})}$$

a piccoli passi ci vogliono tanti passi per rielaborare il corpo della funzione, per cui bisogna introdurre la chiusura transitiva

$$\frac{conf \ \square\ conf'}{conf\ \square^{*}\ conf'} \qquad \frac{conf\ \square\ conf' \quad conf'\ \square^{*}\ conf''}{conf\ \square^{*}\ conf''}$$

$$\frac{F(v,s_1)\ = (*,s')}{(f(v),\ s_1)\ \square_{r[\square/f]}\ (\square,\ s')}$$

Le altre regole (standard per la chiamata di funzione) già mi permettono di rielaborare $(f(e),\ s_0)\ \square^{*}_{r[\square/f]}\ (f(v),\ s_1)$

SOS a piccoli passi per dichiarazioni di funzioni

(vera 1)

Però la prima strategia proposta non ha il “sapore” dei piccoli passi.

Ne vogliamo vedere una seconda, più vicina alla idea di “macchina”.

Abbiamo due obiettivi didattici:

- Elaborare regole per la semantica delle funzioni (blocchi) consistenti con la filosofia dei piccoli passi
- Imparare ad affrontare problemi simili, cioè come si fa a “inventare” le regole

Idea intuitiva: la chiamata di funzione si traduce nel suo corpo modificato con le opportune associazioni fra parametri formali ed attuali.

Per implementarla bisogna

- cambiare la definizione di ambiente in modo da associare ai nomi di funzione quello che serve per poter fare l’espansione quando si incontra una chiamata;
- cambiare le regole corrispondenti alla chiamata (fra quelle per la valutazione di espressione)

SOS a piccoli passi per dichiarazioni di funzioni

(vera 2)

Cominciamo a fare le modifiche senza pretesa di “azzeccare” subito la soluzione

Bisogna cambiare la definizione di ambiente: la componente Fun deve essere modificata in modo da memorizzare i “pezzi” necessari per espandere le chiamate

$$\text{Fun} = \underbrace{L_{PDecs}}_{\text{parametri}}(LW) \,\square\, \underbrace{L_{Stats}}_{\text{body}}(LW) \;\; [\square\, \underbrace{L_{Exp}}_{\text{risultato (opzionale)}}(LW)]$$

$$\frac{}{(\text{void } f(lp)\ \{sts\}, r, s) \;\square\; (\square, r[(lp,sts)/f], s)}$$

$$\frac{}{(\text{RT } f(lp)\ \{sts\ \textbf{result}\ e\}, r, s) \;\square\; (\square, r\,[(lp,sts,e)/f], s)}$$

Il lavorio necessario a calcolare F nell’altro approccio, qui sparisce, ma viene pagato dalla complicazione del meccanismo di chiamata (che era facile).

SOS a piccoli passi per dichiarazioni di funzioni

(meccanismo di chiamata)

Intuitivamente (ed ingenuamente) dovremmo avere (consideriamo il caso di un unico parametro)

$$\frac{(e,s) \rightarrow_r (e',s')}{(f(e),s) \rightarrow_r (f(e'),s')} \qquad\qquad \frac{}{(f(v),s) \rightarrow_r (st,s[v/r(x)])}\ r(f)=(T\ x,st)$$

C'è un problema da risolvere:
quando è stato allocato spazio per x (da chi) e perché nell'ambiente globale?
Se cerchiamo di risolvere con una "badilata":

$$\frac{}{(f(v),r,s) \rightarrow (st,r[l/x],s[v/l])} \quad \begin{array}{l} r(f)=(T\ x,st) \\ Nuova(l,r,s) \end{array}$$

Si creano due situazioni inaccettabili:
i parametri locali delle funzioni sono visibili a livello globale, perché non verranno mai tolti (non posso mettere una regola ad hoc per eliminarli, dato che quando mi trovo a valutare (st,r,s) non posso sapere se e quali locazioni sono "locali" ad una valutazione di funzione),
si è persa la proprietà che le espressioni (e gli statement) non modificano l'ambiente (che era l'unica giustificazione della distinzione fra ambiente e stato)

In effetti non solo non tornano i dettagli, ma manca proprio un concetto generale: l'ambiente di valutazione locale.

SOS a piccoli passi per dichiarazioni di funzioni

(gestione ambiente locale)

L'idea, quindi, è che vogliamo elaborare le chiamate di funzione in configurazioni che consistono non solo del body, dell'ambiente (globale) e dello stato ma anche di un ambiente locale.

Il modo più conveniente di implementare questa idea è permettere nelle configurazioni degli pseudo termini che sintetizzano termini e relativi ambienti locali di valutazione.

Estendiamo quindi ulteriormente LW (= aggiungiamo produzioni a LW+) mediante

- strumenti sintattici per rappresentare (in modo finito) gli ambienti locali in corso di elaborazione (all'inizio della chiamata sarà vuoto, man mano che si aggiungono i parametri cresce)
- pseudo termini composti di termini e ambienti

Env ::= [Links] + (PDecs □ Exps) | [Links].

Links ::= □ | Loc/Id Links | Fun/Id Links.

Loc ::= ...

Fun ::= (PDecs,Env, Stats [, Exp])

servirà dopo

PTerm ::= □Env÷ Stats□| □Env÷ Stats, Exp □| □Env÷ Exp □

Loc deve essere una rappresentazione sintattica delle locazioni, cioè L_{Loc}(LW+) deve essere isomorfo all'insieme Loc delle locazioni usate come dominio semantico.

Le funzioni che fanno passare da un insieme all'altro saranno omesse (o meglio la notazione scelta per loro è invisibile)

SOS a piccoli passi per dichiarazioni di funzioni

(meccanismo di chiamata 2)

$$\frac{(es,s) \Box_{r} (es',s')}{(f(es),s) \Box_{r} (f(es'),s')}$$

semplificazione dei parametri attuali

$$\frac{}{(f(lv),s) \Box_{r} (\Box[] + (ld \Box\ lv) \div st\Box,s)}$$

r(f)=(ld,st)

attivazione chiamata

creazione ambiente locale

$$\frac{}{(\Box rl] + (Tx\ ld \Box\ v\ lv) \div st\Box,s) \Box_{r} (\Box rl\ l/x] + (ld \Box\ lv) \div st\Box,s[v/l])}$$

Nuova(l,r[rl],s)

$$\frac{}{(\Box rl] + (Tx \Box\ v) \div st\Box,s) \Box_{r} (\Box rl\ l/x] \div st\Box,s[v/l])}$$

Nuova(l,r[rl],s)

ultimo passaggio della creazione ambiente locale

$$\frac{(st\ ,s) \Box_{r[rl]} (st'\ ,s')}{(\Box rl] \div st\Box,s) \Box_{r} (\Box rl] \div st'\Box,s')}$$

esecuzione del corpo

$$\frac{(st\ ,s) \Box_{r[rl]} (\Box\ ,s')}{(\Box rl] \div st\Box,s) \Box_{r} (\Box,s'')}$$

s”=s’-{l | IsIn(l,rl)}

ultimo passaggio della esecuzione del corpo

SOS a piccoli passi per dichiarazioni di funzioni

(binding statico e dinamico)

Le regole date finora non corrispondono alla semantica scelta nel caso a grandi passi e denotazionale.
Infatti, gli identificatori liberi nel corpo di una funzione vengono legati alla (ultima) dichiarazione presente nell'ambiente al momento della chiamata (binding dinamico) mentre nelle semantiche precedenti venivano legati alla dichiarazione presente nell'ambiente al momento della dichiarazione (binding statico).

```
int x = 3;
void f(int z;) {x = z;}
void g(int x;) {f(x);}
```

Con binding statico una chiamata di g ha l'effetto di assegnare alla variabile globale x il valore del parametro attuale
Con binding dinamicouna chiamata di g ha l'effetto di assegnare al parametro di g il valore del suo parametro attuale, cioè non ha alcun effetto
Per esercizio calcolare la semantica nei due casi.

La semantica con binding dinamico richiede una diversa semantica statica.
Modifichiamo la semantica data in modo che corrisponda al binding statico.
Bisogna associare nell'ambiente ad ogni nome di funzione anche l'ambiente al momento della dichiarazione (o quanto meno la parte che serve per gestire gli identificatori liberi che compaiono nel corpo)

SOS a piccoli passi per dichiarazioni di funzioni

(binding statico)

Fun = $L_{Env}(LW)$ □ $L_{Decs}(LW)$ □ $L_{Stats}(LW)$ [□ $L_{Exp}(LW)$]

$$\frac{}{(\text{void } f(lp)\ \{sts\},r,s)\ □\ (□, r[(r',lp,sts)/f],s)} \quad r' = r-\{f\}$$

PTerm ::= □Env,Env÷ Stats□| □Env,Env÷ Stats, Exp □| □Env,Env÷ Exp □

$$\frac{}{(f(lv),s)\ □_{r}\ (□r_d, [] + (ld\ □\ lv) \div st□,s)} \; r(f)=(r_d,ld,st) \qquad \text{attivazione chiamata}$$

Siccome userò r_d per aggiornare l'ambiente al momento della chiamata ed ottenere l'ambiente in cui valutare il corpo con ripristinate le associazioni fra identificatori liberi e loro definizioni in r_d, devo stare attenta a non ripristinare anche eventuali associazioni di f con altre definizioni vecchie, perché questo renderebbe impossibile fare chiamate ricorsive.

Esercizi proposti

- Modificare la semantica data in modo da memorizzare il minimo indispensabile dell'ambiente al momento della definizione.
- Definire la semantica statica compatibile con il binding dinamico
- Posporre la semplificazione di ciascun parametro attuale al momento in cui diventa indispensabile

SOS a piccoli passi per dichiarazioni di funzioni

(versione finale per il caso con tipo non vuoto)

$$\frac{}{(\text{rt f(lp) \{sts result e\}},r,s) \;\square\; (\square, r[(r-\{f\},lp,sts,e)/f],s)}$$ dichiarazione

$$\frac{(es,s) \;\square_r\; (es',s')}{(f(es),s) \;\square_r\; (f(es'),s')}$$ semplificazione dei parametri attuali

$$\frac{}{(f(lv),s) \;\square_r\; (\square r_d, []+ (ld \;\square\; lv) \div sts,e\square,s)}\; r(f)=(r_d,ld,sts,e)$$ attivazione chiamata

creazione ambiente locale

$$\frac{}{(\square r_d,[rl] + (Tx\ ld \;\square\; v\ lv) \div st,e\square,s) \;\square_r\; (\square r_d,[rl\ l/x] + (ld \;\square\; lv) \div st,e\square,s[v/l])}\; Nuova(l,r[rl],s)$$

$$\frac{}{(\square r_d,[rl] + (Tx \;\square\; v) \div st,e\square,s) \;\square_r\; (\square r_d,[rl\ l/x] \div st,e\square,s[v/l])}\; Nuova(l,r[rl],s)$$ ultimo passaggio della creazione ambiente locale

$$\frac{(st\ ,s) \;\square_{r[r_d][rl]}\; (st'\ ,s')}{(\square r_d,[rl] \div st,e\square,s) \;\square_r\; (\square r_d,[rl] \div st',e\square,s')}$$ esecuzione del corpo

$$\frac{(st\ ,s) \;\square_{r[r_d][rl]}\; (\square\ ,s')}{(\square r_d,[rl] \div st,e\square,s) \;\square_r\; (\square r_d,[rl] \div e\square,s')}$$ ultimo passaggio della esecuzione del corpo

$$\frac{(e\ ,s) \;\square_{r[r_d][rl]}\; (e'\ ,s')}{(\square r_d,[rl] \div e\square,s) \;\square_r\; (\square r_d,[rl] \div e'\square,s')}$$ valutazione del risultato

$$\frac{}{(\square r_d,[rl] \div v\square,s) \;\square_r\; (v,s'')}\; s''=s-\{l \mid IsIn(l,rl)\}$$ ultimo passaggio della valutazione del risultato

Tipi di chiamata

La semantica delle funzioni vista finora corrisponde alla chiamata per valore.

Corrisponde al concetto matematico di funzione, cioè di meccanismo che dati dei valori presi all'interno di un insieme (il *dominio*) produce un risultato scelto all'interno di un insieme (il *codominio*).

Perciò un passo preliminare all'esecuzione di una chiamata è la trasformazione degli argomenti (=parametri attuali) da termini del linguaggio a valori (siano essi termini di forma particolare, come nella SOS, o elementi di un carrier dell'algebra semantica, come nella semantica denotazionale)

Adesso, vogliamo invece rappresentare un meccanismo dal sapore più sintattico, che cattura l'intuizione di chiamata come macro-espansione, cioè la chiamata viene sostituita dal corpo della funzione con i parametri attuali sostituiti al posto dei parametri formali e poi semplificati assieme al resto.

Call by need 1

Invece di sostituire le espressioni usate come parametri attuali della chiamata direttamente nel testo, risulta pù facile associarle ai parametri formali nell'ambiente locale di valutazione.

Questo richiede di cambiare la nozione di stato, permettendo di associare ad una locazione non solo (termini che rappresentano) valori, ma anche termini generici.

Cioè cambiamo nuovamente Value:

Value è l'insieme dei termini/espressioni: Value = $L_{Exp}(LW)$

Le regole restano come erano tranne che

- La regola per la semplificazione dei parametri prima della chiamata viene omessa
- Le regole per attivazione della chiamata, creazione dell'ambiente locale avendo modificato Value adesso si applicano con v (lv) instanziato su (liste di) espressioni qualunque.

Questa scelta permette (ma non richiede) di modificare anche altri aspetti, ad esempio la regola per la dichiarazione può essere semplificata

$$\frac{}{(t\ x = e,r,s) \rightarrow (r[l/x],s[e/l])} \quad \begin{array}{l} \text{Nuova}(l,r,s) \\ e \in L_{Exp}(LW) \\ x \in L_{Id}(LW) \\ t \in L_{Type}(LW) \end{array}$$

Vantaggi della pigrizia

La strategia proposta prende anche il nome di lazy valuation, perché gli argomenti vengono valutati solo se necessari.

Questo altera effettivamente la semantica della chiamata quando la valutazione di un argomento non termina ma questo argomento non serve, ad esempio, perché compare solo in un ramo di un if_then_else che non viene eseguito.

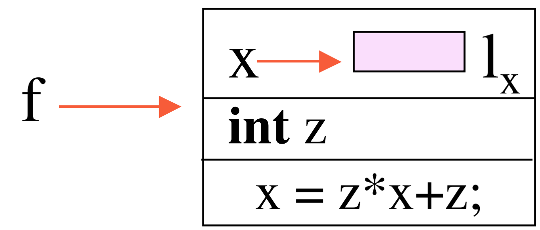
```
bool x;
int f(int z;) {if (z<10){z = z;} else {z=f(z);} result z}
void g(int a; int b;) {if (a=0){x= true;} else {x=(a=b);}}
…..
g(0,f(112))
```

Call by need / Lazy valuation

Consideriamo un esempio.

int x =1;
void f(**int** z;) {x = z*x+z;}

Questo modifica l'ambiente introducendo l'associazione



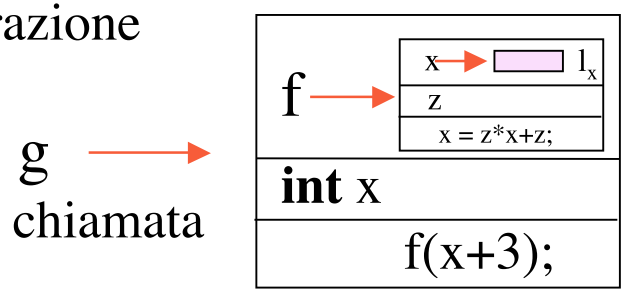
Ogni successiva chiamata dovrebbe essere espansa:

f(x+3) $\longrightarrow$ x =(x+3)*x+(x+3);

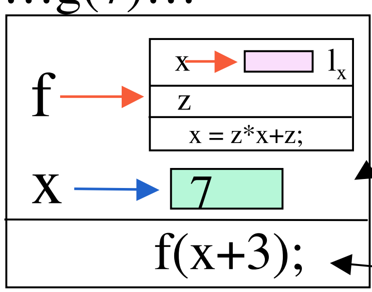
soliti problemi di ambiguità dovuti alla rappresentazione a stringa. In realtà è come se ci fossero le parentesi

Nel caso di binding statico questo causa un grosso problema: identificatori liberi nei parametri attuali vengono legati alle dichiarazioni al momento della definizione

Supponiamo che ci sia una nuova dichiarazione
void g(**int** x;) {f(x+3);}
Che modifica a sua volta l'ambiente



Vediamo cosa succede se si incontra una chiamata
…g(7)…



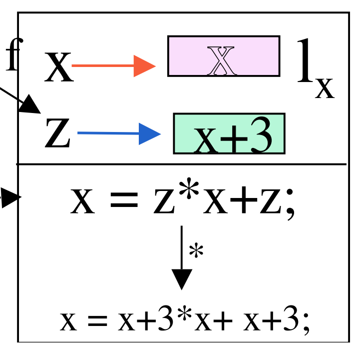
Creazione ambiente locale di valutazione per g

Valutazione del corpo di g

Chiamata della f

Creazione ambiente locale di valutazione per f

Valutazione del corpo di f



Non ci sono soluzioni tranne usare binding dinamico

Esercizi proposti

Dare la semantica dinamica operazionale a piccoli passi di altri linguaggi visti.

Linguaggio applicativo

Suggerimento: non cercare di individuare i termini che rappresentano valori - per i tipi funzionali non si può fare - ma dare solo la relazione di semplificazione; come configurazioni usare coppie termine+ambiente; un ambiente associa espressioni a identificatori; per gestire le funzioni sarà necessario estendere il linguaggio come nel caso imperativo