5 Procedure

Le procedure sono uno meccanismo fondamentale per strutturare i programmi. Esse sono uno strumento essenziale per supportare la metodologia di sviluppo top-down, facilitano il riutilizzo del codice, e facilitano una programmazione modulare (chi usa una procedure non ha bisogno di sapere come e' implementata, e viceversa). All'inizio consideriamo il caso semplificato delle procedure non ricorsive e senza parametri, descrivendo le estensioni da apportare alla semantica operazionale statica e dinamica.

$$\begin{array}{rcl} d \in< Declaration > & ::= & \ldots \mid \mathrm{proc}\ P = c \\ c \in< Command > & ::= & \ldots \mid P \end{array}$$

Per gli identificatori adottiamo la convenzione che P e' un identificatore di procedura.

Vi sono due modalita' di gestione dell'ambiente direttamente connesse alle procedure: ambiente statico, e ambiente dinamico. Nella gestione statica dell'ambiente una chiamata di procedura P equivale ad eseguire il corpo c nell'ambiente in cui P e' dichiarata (e nello stato al momento della chiamata). Nella gestione dinamica invece si utilizza ambiente (e stato) al momento della chiamata. La gestione statica e' preferibile, perche' chi scrive il corpo c della procedure P conosce l'ambiente al momento della dichiarazione, ma non e' detto che sappia in quali ambienti P verra' chiamata. Anzi lo spirito della programmazione modulare e' proprio quello di ridurre al minimo l'interazione tra chi scrive c e chi utilizza P. Inoltre la gestione dinamica invalida principi basilari, quali il fatto che la redenominazione dei nomi locali non abbia conseguenze osservabili.

5.1 Semantica delle procedure con scoping statico

Cosideriamo le aggiunte/modifiche alla semantica statica per trattare le procedure con gestione dell'ambiente statica.

- $\rho \in Env^S = Id \stackrel{fin}{\to} \ldots \mid proc$ ambiente statico
- $s \in State^S = \ldots$ stato statico, non e' direttamente rilevante
- Le nuove regole di inferenza per la semantica statica sono:

  $$(\mathrm{proc})\ \frac{\rho, s \vdash c \Rightarrow ok}{\rho \vdash s, \mathrm{proc}\ P = c \Rightarrow (0,0), P \mapsto proc} \qquad (\mathrm{call})\ \frac{\rho(P) = proc}{\rho, s \vdash P \Rightarrow ok}$$

Per trattare procedure mutuamente ricorsive a livello di semantica statica basta sostituire la regola (block) con (block*), come gia' visto per i tipi mutuamente ricorsivi.

Per la semantica operazionale dinamica le aggiunte/modifiche sono

- $\rho \in Env = Id \stackrel{fin}{\to} \ldots \mid proc(Env, Command)$ ambiente dinamico.

  Una coppia (ρ, c) e' detta chiusura, e permette di associare ad un identificatore P di procedure non solo il corpo della procedura, ma l'ambiente al momento della dichiarazione
- Le nuove regole di inferenza per la semantica dinamica sono:

  $$(\mathrm{proc})\ \frac{}{\rho \vdash s, \mathrm{proc}\ P = c \Rightarrow s, P \mapsto proc(\rho, c)} \qquad (\mathrm{call})\ \frac{\begin{array}{c}\rho(P) = proc(\rho', c) \\ \rho' \vdash s, c \Rightarrow s'\end{array}}{\rho \vdash s, P \Rightarrow s'}$$

Per trattare procedure mutuamente ricorsive a livello di semantica dinamica occorrono alcune modifiche alla nozione di chiusura ed alla gestione dell'ambiente

- $\rho \in Env = Id \stackrel{fin}{\to} \ldots \mid proc(Env, Env, Command)$ ambiente dinamico.

  Una terna (ρ, ρ', c) e' una chiusura in cui ρ e' la parte non ricorsiva dell'ambiente, mentre ρ' e' la parte dell'ambiente definito in modo ricorsivo, cioe' gli identificatori nel dominio di ρ' possono essere usati nel codominio.
- $$(\mathrm{proc})\ \frac{}{\rho \vdash s, \mathrm{proc}\ P = c \Rightarrow s, P \mapsto proc(\rho, 0, c)}$$

  La parte ricorsiva e' vuota perche' l'ambiente ricorsivo e' noto solo dopo aver esaminato tutta la dichiarazione.

In presenza di dichiarazioni mutuamente ricorsive e' necessario rivedere anche la regola di composizione di dichiarazioni (l'ordine di dichiarazione diventa irrilevante) e quella per un blocco (che deve inserire l'ambiente ricorsivo nelle chiusure)

$$(\text{comp*})\ \frac{\begin{array}{c}\rho \vdash s, d_1 \Rightarrow s_1, \rho_1 \\ \rho \vdash s_1, d_2 \Rightarrow s_2, \rho_2\end{array}}{\rho \vdash s, d_1; d_2 \Rightarrow s_2, \rho_1 \oplus \rho_2}\ \rho_i \text{ con domini disgiunti}$$

se vi sono solo dichiarazioni di procedure, lo stato non cambia e quindi l'ordine di dichiarazione e' irrilevante

$$(\text{block})\ \frac{\begin{array}{c}\rho \vdash s, d \Rightarrow s', \rho' \\ \rho \oplus Rec\ \rho' \vdash s', c \Rightarrow s''\end{array}}{\rho \vdash s, d \text{ begin } c \text{ end} \Rightarrow s''}$$

dove $(Rec\ \rho')(P) \stackrel{\Delta}{=} (\rho, \rho', c)$ se $\rho'(P) = (\rho, \_, c)$. Quindi $Rec\ \rho'$ informa le chiusure che l'ambiente ricorsivo e' ρ'

- $$(\text{call})\ \frac{\begin{array}{c}\rho(P) = proc(\rho', \rho'', c) \\ \rho' \oplus Rec\ \rho'' \vdash s, c \Rightarrow s'\end{array}}{\rho \vdash s, P \Rightarrow s'}$$

  Anche in questo caso si deve informare le varie chiusure in ρ'' che l'ambiente ricorsivo e' ρ''

5.2 Semantica delle procedure con scoping dinamico

Lo scoping dinamico equivale ad una sostituzione testuale di c al posto di una chiamata di P (simile a cio' che avviene con la macro espansione), e quindi il significato degli identificatori usati in c (ma non dichiarati localmente a c) sono interpretati nell'ambiente del chiamante, che varia da chiamata a chiamata. Si osservi che quando c usa solo identificatori dichiarati localmente in c (o parametri formali), le due modalita' di gestione diventano equivalenti.

- $\rho \in Env^S = Id \stackrel{fin}{\to} \ldots \mid proc(Command)$ ambiente statico
- $s \in State^S = \ldots$ stato statico, non e' direttamente rilevante
- Le nuove regole di inferenza per la semantica statica sono:

  $$(\text{proc})\ \frac{}{\rho \vdash s, \text{proc } P = c \Rightarrow (0,0), P \mapsto proc(c)} \qquad (\text{call})\ \frac{\begin{array}{c}\rho(P) = proc(c) \\ \rho, s \vdash c \Rightarrow ok\end{array}}{\rho, s \vdash P \Rightarrow ok}$$

  Con lo scoping statico i controlli statici (type-checking) su c venivano effettuati una sola volta (nell'ambiente in cui P e' dichiarata), adesso vengono effettuati per ogni chiamata di P.

Per la semantica operazionale dinamica le aggiunte/modifiche sono

- $\rho \in Env = Id \stackrel{fin}{\to} \ldots \mid proc(Command)$ ambiente dinamico.
- Le nuove regole di inferenza per la semantica dinamica sono simili a quelle per la semantica statica

  $$(\text{proc})\ \frac{}{\rho \vdash s, \text{proc } P = c \Rightarrow s, P \mapsto proc(c)} \qquad (\text{call})\ \frac{\begin{array}{c}\rho(P) = proc(c) \\ \rho \vdash s, c \Rightarrow s'\end{array}}{\rho \vdash s, P \Rightarrow s'}$$

Con gestione dinamica dell'ambiente non sono richiesti particolari accorgimenti per trattare le definizioni ricorsive.

5.3 Procedure con parametri

La flessibilita' delle procedure e' notevolmente accresciuta se e' possibile parametrizzarle rispetto a dei parametri che possono essere istanziati in modo diverso da chiamata a chiamata. In effetti la parametrizzazione e' un concetto del tutto generale, che puo' essere utilmente applicato ad altre categorie sintattiche oltre ai comandi (espressioni, variabili, dichiarazioni). Vi e' anche molta discrezionalita' su cosa puo' essere utilizzato come parametro. Un liguaggio che offre buoni meccanismi di parametrizzazione facilita enormemente il riutilizzo del codice. Si consideri un algoritmo di ordinamento basato su confronti, p.e. mergesort o quicksort. I parametri naturali dell'algoritmo sono un insieme X con un ordine totale $\leq : X, X \to Bool$ ed una sequenza A costituita da n elementi di X. Quindi vorremo poter scrivere

```
proc mergesort (type X; fun le(X,X):Bool; const n; var A:X[n])
```

Addirittura potremo aspettarci che i parametri X ed n siano ricavati automaticamente dagli altri due. Da questo punto di vista sia il C che il Pascal lasciano abbastanza a desiderare, p.e. i parametri X ed n non sono consentiti (poiche' il C tratta gli array come puntatori e' possibile passare un array senza fornire le sue dimenzioni). Vi sono alcune estensioni del Pascal che consentono il parametro n, e vredremo che ML (ed altri linguaggi funzionali) consentono di inferire il parametro X.

Consideriamo il seguente linguaggio, dove abbiamo introdotto un meccanismo uniforme di parametrizzazione per comandi, espressioni e variabili. Inoltre per illustrare meglio la simmetria tra parametrizzazione e dichiarazione abbiamo adottato la keyword "newvar" per la dichiarazione di una nuova variabile (che richiede l'allocazione di memoria), ed utilizzato la keyword "var" per una nuova forma di dichiarazione che permette di associare un identificatore ad una variabile gia allocata.

$$
\begin{array}{l}
u, x, X, P, F, S \in < Identifier >::= \ldots \\
T \in < Type >::= X \mid \ldots \\
\begin{array}{l|l}
d \in < Declaration >::= & fp \in < FormalParam >::= \\
\text{type } X = T \mid & \\
\text{newvar } x{:}\,X \mid & \\
\text{var } x = v \mid & \text{var } x{:}\,X \mid \\
\text{const } u = e \mid & \text{const } u{:}\,X \mid \\
\text{proc } P(fp^*) = c \mid & \text{proc } P(fp^*) \mid \\
\text{fun } F(fp^*) = e \mid & \text{fun } F(fp^*){:}\,X \mid \\
\text{sel } S(fp^*) = v & \text{sel } S(fp^*){:}\,X
\end{array} \\
ap \in < ActualParam >::= v \mid e \mid P \mid F \mid S \\
v \in < Variable >::= \ldots \mid S(ap^*) \\
e \in < Expression >::= \ldots \mid F(ap^*) \\
c \in < Command >::= \ldots \mid P(ap^*)
\end{array}
$$

Per gli identificatori adottiamo la convenzione che P e' un identificatore di procedura, F di funzione, e S di selettore.

A livello di semantica statica e dinamica (con scoping statico) le principali modifiche riguardano l'ambiente. Per semplicita' ci limitiamo a trattare il caso di procedure con parametri, poiche' le altre forme di parametrizzazione sono del tutto analoghe. Per la semantica statica abbiamo:

- $\rho \in Env^S = Id \stackrel{fin}{\to} \ldots \mid proc(FD^*)$ ambiente statico, FD e' l'insieme dei descrittori ft di parametro formale $FD = var(N) \mid const(N) \mid proc(FD^*)$, dove N e' l'insieme dei nomi di tipo.
- $\rho, s \vdash fp \Rightarrow \rho'$ e' una asserzione che associa un ambiente ad una dichiarazione di parametro formale

  $$(\text{var})\ \frac{\rho(X) = type(m)}{\rho, s \vdash \text{var } x{:}\,X \Rightarrow x \mapsto var(m)} \qquad (\text{const})\ \frac{\rho(X) = type(m)}{\rho, (n, g) \vdash \text{const } u{:}\,X \Rightarrow u \mapsto const(m)}\ g(m) = enum$$

  $$(\text{proc})\ \frac{\rho, s \vdash fp_i \Rightarrow x_i \mapsto ft_i}{\rho, s \vdash \text{proc } P(fp_i) \Rightarrow P \mapsto proc(ft_i)}\ g(m) = enum$$

- Le nuove regole di inferenza per la semantica statica sono:

  $$(\text{proc})\ \frac{\rho, s \vdash fp_i \Rightarrow x_i \mapsto ft_i \quad \rho \oplus (x_i \mapsto ft_i), s \vdash c \Rightarrow ok}{\rho \vdash s, \text{proc } P(fp_i) = c \Rightarrow (0, 0), P \mapsto proc(ft_i)} \qquad (\text{call})\ \frac{\rho(P) = proc(ft_i) \quad \rho, s \vdash ap_i{:}\,ft_i}{\rho, s \vdash P(ap_i) \Rightarrow ok}$$

  dove $\rho, s \vdash ap{:}\,ft$ controlla la compatibilita' tra parametro formale ed attuale, ed e' un abbreviazione per: $\rho, s \vdash_v ap \Rightarrow m$ se $ft = var(m)$, $\rho, s \vdash_e ap \Rightarrow m$ se $ft = const(m)$, e $\rho(ap) = ft$ se $ft = proc(ft_i)$.

Per la semantica dinamica abbiamo:

- $\rho \in Env = Id \stackrel{fin}{\to} \ldots \mid proc(Id^*, Env, Env, Command)$ ambiente dinamico, dove adesso una chiusura specifica anche gli identificatori dei parametri formali
- Le nuove regole di inferenza per la semantica dinamica sono:

  $$(\text{proc})\ \frac{x_i = \text{identificatore dichiarato in } fp_i}{\rho \vdash s, \text{proc } P(fp_i) = c \Rightarrow s, P \mapsto proc(x_i, \rho, 0, c)} \qquad (\text{call})\ \frac{\rho(P) = proc(x_i, \rho', \rho'', c) \quad \rho, s \vdash ap_i \Rightarrow a_i \quad \rho' \oplus (x_i \mapsto a_i) \oplus Rec\ \rho'' \vdash s, c \Rightarrow s'}{\rho \vdash x, P(ap_i) \Rightarrow s'}$$

  dove $\rho, s \vdash ap \Rightarrow a$ calcola il parametro attuale da associare al corrispondente identificatore di parametro formale, p.e. si deve avere $\rho, s \vdash_v ap_i var(l, t)$ quando il parametro formale fp_i e' var $x{:}\,X$. Per semplicita' abbiamo ignorato alcuni controlli di tipo a run-time.

5.4 Esercizi

1. Supponiamo di non ammettere ne' parametri di tipo procedura ne' procedure ricorsive, ovvero una procedura P puo' chiamare una procedura Q solo se Q e' visibile secondo le regole dello scoping statico e la definizione di Q precede (testualmente) quella di P. Si faccia vedere che in fase di esecuzione sullo stack non vi possono essere 2 record di attivazione per la stessa procedura.

2. Usando il precedente risultato discutere la possibilita' di gestione statica della memoria nelle ipotesi che: il layout dei tipi sia determinato staticamente (e non vi siano puntatori), le procedure non siano ricorsive e non possano essere passate come parametro.

3. Far vedere con un esempio che il requisito di non avere procedure passate come parametro non e' eliminabile.

4. Applicare le regole della semantica operazionale dinamica per procedure ricorsive (con scoping statico) al seguente programma

   ```
   begin
   proc P=Q;
   proc Q=P;
   P
   end
   ```

   serve a familiarizzarsi con l'uso dell'albiente mutuamente ricorsivo

5. Applicare le regole della semantica operazionale dinamica con scoping statico e con scoping dinamico al seguente programma, e dire cosa viene stampato nei due casi

   ```
   begin
   var x:int;
   proc P=(x:=1);
   x:=0;
   begin
     var x:int;
     x:=0;
     P; write(x)
   end
   P;write(x)
   end;
   ```

6. (difficile) Rivedere la semantica operazione dinamica per il linguaggio che ammette dichiarazioni di procedure e procedure come parametro (per semplicita' non si ammetta la ricorsione) in modo da trattare correttamenta i controlli al momento della chiamata (evitare quei controlli che sono gia' fattibili a tempo di compilazione).

7. (difficile) Si vuole modificare la gestione dei tipi union in modo da migliorare gli aspetti legati alla sicurezza dei dati: il campo tag $v.x$ e' trattato come una costante (cioe' puo' essere letto ma non modificato in modo diretto), quando si effettua un assegnamento ad una variante $v.x_i$ si modifica di conseguenza il tag $v.x$, quando si legge una variante $v.x_i$ si controlla che il tag $v.x$ abbia il valore giusto. Spiegare come vanno modificate la semantica statica e dinamica.