Azzolini Riccardo 2021-04-12

Estensione del modello di sostituzione alle classi

1 Valutazione delle espressioni di creazione

In precedenza si è definito il significato dell'applicazione di funzioni utilizzando il modello di sostituzione. Adesso, è necessario estendere tale modello al caso di espressioni che coinvolgono classi e metodi.

Per iniziare, data una classe[1]

$$\texttt{class } C\texttt{(}x_1\texttt{, ..., }x_m\texttt{)}$$

con i parametri formali $x_1, \ldots, x_m$ (la lista può essere vuota, cioè è ammesso che la classe non abbia alcun parametro), si consideri l'istanziazione (espressione di creazione) di un oggetto di tale classe tramite il *costruttore primario*,

$$\texttt{new } C\texttt{(}e_1\texttt{, ..., }e_m\texttt{)}$$

dove i parametri attuali $e_1, \ldots, e_m$ sono espressioni di tipi compatibili a quelli dei corrispondenti parametri formali. Per valutare tale espressione, si valutano i parametri attuali $e_1, \ldots, e_m$, ottenendo i valori $v_1, \ldots, v_m$, e il risultato della valutazione dell'espressione complessiva è il *valore* $\texttt{new } C\texttt{(}v_1\texttt{, ..., }v_m\texttt{)}$. Infatti, siccome il modello di sostituzione è puramente sintattico, funziona tramite operazioni di riscrittura di stringhe, anche i valori degli oggetti devono essere rappresentabili da delle stringhe. Questo è il motivo dell'importanza del costruttore primario: dato che tutti i costruttori della classe devono richiamarlo (eventualmente in modo indiretto), esso permette di definire per gli oggetti creati una rappresentazione uniforme, indipendentemente dagli specifici costruttori impiegati.

2 Valutazione dell'invocazione dei metodi

Si consideri adesso il caso in cui il corpo della classe C contiene la definizione di un metodo f:

$$\texttt{class } C\texttt{(}x_1\texttt{, ..., }x_m\texttt{) \{ ... def }f\texttt{(}y_1\texttt{, ..., }y_n\texttt{) = }B\texttt{ ... \}}$$

[1] I tipi dei parametri di classi e metodi sono sostanzialmente irrilevanti nella definizione dei meccanismi di valutazione (poiché tali meccanismi vengono applicati quando il compilatore ha già verificato che i tipi dei parametri attuali sono compatibili con i tipi dei parametri formali), dunque per brevità vengono omessi.

- $x_1, \ldots, x_m$ sono, come prima, i parametri formali della classe;
- $y_1, \ldots, y_n$ sono i parametri formali del metodo f (anche questa lista può essere vuota);
- B è il corpo del metodo f, un'espressione in cui possono comparire i parametri formali del metodo e della classe e la pseudo-variabile `this`.

Dato il valore di un oggetto di classe C, `new` $C(v_1, \ldots, v_m)$, dove $v_1, \ldots, v_m$ sono valori (già valutati), si vuole valutare l'espressione

$$\texttt{new } C(v_1, \ldots, v_m).f(e_1, \ldots, e_n)$$

Secondo la strategia call-by-value, il primo passo è valutare i parametri attuali $e_1, \ldots, e_n$ del metodo, ottenendo così i valori $w_1, \ldots, w_n$:

$$\texttt{new } C(v_1, \ldots, v_m).f(w_1, \ldots, w_n)$$

Poi, l'espressione viene riscritta come

$$[w_1/y_1, \ldots, w_n/y_n][v_1/y_1, \ldots, v_m/y_m][\texttt{new } C(v_1, \ldots, v_m)/\texttt{this}]B$$

cioè si rimpiazza l'invocazione del metodo f con il suo corpo B, al quale sono applicate tre sostituzioni:

1. la prima (quella più a sinistra) sostituisce i *parametri formali del metodo* con i valori dei corrispondenti parametri attuali specificati nell'invocazione del metodo;
2. la seconda sostituisce i *parametri formali della classe* con i valori dei corrispondenti parametri attuali specificati all'atto della creazione dell'oggetto;
3. la terza sostituisce la *self-reference* `this` con il valore dell'oggetto su cui il metodo è invocato.

Le sostituzioni vengono applicate nell'ordine da sinistra a destra,[2] il che è importante per il rispetto delle regole di adombramento: i parametri formali del metodo adombrano eventuali parametri formali della classe aventi gli stessi nomi. Infatti, i parametri del metodo vengono sostituiti per primi, e nel farlo i loro nomi "scompaiono" dall'espressione B, quindi eventuali parametri di classe con lo stesso nome non vengono poi sostituiti.[3]

[2] L'applicazione delle sostituzioni avviene da sinistra a destra nonostante esse siano scritte a sinistra dell'espressione a cui si applicano perché, formalmente, scrivere le sostituzioni una di fianco all'altra rappresenta implicitamente l'operatore di composizione delle sostituzioni, dunque l'applicazione $[\ldots][\ldots][\ldots]B$ va letta come $\big([\ldots][\ldots][\ldots]\big)B$ e non come $[\ldots]\big([\ldots]\big([\ldots]B\big)\big)$.

[3] È garantito che i nomi dei parametri sostituiti scompaiano perché tali parametri vengono sostituiti con dei semplici valori, che *non* sono espressioni che possono contenere nomi. Ciò permette di trattare le sostituzioni in modo semplificato, senza bisogno di definire formalmente il comportamento dell'operatore di composizione — in particolare, non è necessario considerare il caso in cui il valore sostituito da una sostituzione contiene nomi che vengono a loro volta sostituiti dalla sostituzione successiva.

2.1 Esempio

Si consideri una versione semplificata della classe `Rational`:

```
class Rational(x: Int, y: Int) {
  def num = x
  def den = y
  // ...
  def less(that: Rational) =
    this.num * that.den < that.num * this.den
  // ...
}
```

Si vuole valutare l'espressione `new Rational(1, 2).num`. Qui i parametri attuali dell'oggetto sono già valutati, quindi si può passare direttamente alla valutazione dell'accesso al campo. Dal punto di vista del modello di sostituzione un campo, soprattutto se definito con `def` (come è stato fatto, per semplicità, nella classe `Rational` di questo esempio), è equivalente a un metodo senza argomenti, dunque l'accesso a un campo può essere valutato esattamente come l'invocazione di un metodo:

$$\begin{aligned}&\texttt{new Rational(1, 2).num}\\ &\quad\rightarrow []\texttt{[1/x, 2/y][new Rational(1, 2)/this]x}\\ &\quad\rightarrow \texttt{1}\end{aligned}$$

Si noti che le parentesi quadre vuote, $[]$, rappresentano la sostituzione vuota, che non sostituisce nulla, poiché `num` non ha argomenti. Tali parentesi potrebbero essere omesse, ma qui sono indicate proprio per mettere in evidenza il fatto che lo schema di valutazione per un campo è esattamente quello che si usa per un metodo.

Un campo definito con `val`, invece che con `def`, potrebbe per semplicità essere trattato allo stesso modo: in assenza di effetti collaterali, usare `val` o `def` non cambia il risultato, e trattando tutti i campi come se fossero definiti con `def` si evita il problema di come esprimere la valutazione delle espressioni associate ai campi `val`, che avviene al momento della creazione dell'oggetto e non al momento dell'accesso ai campi stessi (quindi il meccanismo di valutazione dei metodi non sarebbe più adeguato).

Come altro esempio, sempre sulla classe `Rational`, si vuole valutare l'espressione

```
new Rational(1, 2).less(new Rational(2, 3))
```

che questa volta è l'invocazione di un "vero" metodo, con un argomento. Anche qui i parametri attuali della classe sono già valutati, dunque si applica subito la regola di

valutazione per i metodi:

```
new Rational(1, 2).less(new Rational(2, 3))
    → [new Rational(2, 3)/that][1/x, 2/y][new Rational(1, 2)/this]
      this.num * that.den < that.num * this.den
    → new Rational(1, 2).num * new Rational(2, 3).den <
      new Rational(2, 3).num * new Rational(1, 2).den
```

A questo punto, ciascuno degli accessi ai campi viene valutato come nell'esempio precedente (per brevità, sono mostrati direttamente i risultati di tali valutazioni), e infine vengono applicati gli operatori:

```
    ↠ 1 * 3 < 2 * 2
    ↠ 3 < 4
    → true
```