Tipi e costrutti avanzati di OCL

Lezione 24 di Ingegneria del software

Docente: Giuseppe Scollo

Università di Catania, sede di Comiso (RG)
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Studi in Informatica applicata, AA 2007-8

Indice

tipi di base predefiniti, il tipo Boolean

OCL ha 4 tipi di base predefiniti (i soliti), con relative operazioni:

Boolean, Integer, Real, String

le definizioni delle operazioni non sono sempre le solite...

operazioni su valori **Boolean**, con valore di ritorno **Boolean** :

disgiunzione:	a **or** b
congiunzione:	a **and** b
or esclusivo:	a **xor** b
negazione:	**not** a
eguaglianza:	a = b
ineguaglianza:	a <> b
implicazione:	a **implies** b

i tipi di base Integer e Real

operazioni standard di **Integer** e **Real** :

operazione	notazione	tipo del risultato
eguaglianza	a = b	**Boolean**
ineguaglianza	a <> b	**Boolean**
minore	a < b	**Boolean**
maggiore	a > b	**Boolean**
minore o uguale	a <= b	**Boolean**
maggiore o uguale	a > b	**Boolean**
addizione	a + b	**Integer** or **Real**
sottrazione	a - b	**Integer** or **Real**
moltiplicazione	a * b	**Integer** or **Real**
divisione	a / b	**Real**
modulo	a **.mod**(b)	**Integer**
divisione intera	a **.div**(b)	**Integer**
valore assoluto	a **.abs**()	**Integer** or **Real**
massimo fra due	a **.max**(b)	**Integer** or **Real**
minimo fra due	a **.min**(b)	**Integer** or **Real**
miglior arrotondamento	a **.round**()	**Integer**
arrotondamento per difetto	a **.floor**()	**Integer**

il tipo String

sequenza finita di caratteri fra apici:

```
'ad esempio questa stringa'
```

operazioni standard di **String**, dove s, t sono di tipo **String**, e m, n sono di tipo **Integer**:

operazione	notazione	tipo del risultato
concatenazione	s **.concat**(t)	**String**
lunghezza	s **.size**()	**Integer**
conversione minuscolo	s **.toLower**()	**String**
conversione maiuscolo	s **.toUpper**()	**String**
sottostringa	s **.substring**(m,n)	**String**
eguaglianza	s = t	**Boolean**
ineguaglianza	s <> t	**Boolean**

sintassi delle espressioni di base

per ridurre il numero di parentesi:

precedenza fra operatori OCL

in ordine decrescente:

operazioni	operatori
nome di path	**::**
valore pre- in postcondizione	**@pre**
operazioni 'dot', 'arrow' e di messaggi	**., ->, ^, ^^**
operazioni unarie	**-, not**
moltiplicazione, divisione	***, /**
addizione, sottrazione	**+, -**
operazioni relative di confronto	**<, >, <=, >=, <>, =**
operazioni booleane	**and, or, xor**
implicazione booleana	**implies**

operatori (binari) **infissi**:

+, -, *, /, <, >, <=, >=, <>, =, and, or, xor, implies

N.B.: solo la forma infissa di questi operatori è ammessa

caratteristiche dei tipi definiti dall'utente

tipi definiti dall'utente: quelli presenti in **elementi del modello**

le loro **caratteristiche:**

- **attributi**
- **operazioni**
- attributi **di classe**
- operazioni **di classe**
- **estremi di associazioni**

attributi e operazioni sono utilizzabili in espressioni OCL

- **N.B.** operazioni: solo **query**

estremi di associazioni: v. appresso

associazioni, tipi enumerativi

uso di estremi di associazioni, (o di nomi di classi associate, in assenza di nomi di estremi):

- in **espressioni di navigazione**
  - cioè con "." o "->" a seconda delle molteplicità nel percorso
- la navigazione attraverso **classi di associazione** sembra alquanto controversa...
- navigazione attraverso relazioni di generalizzazione?
- no, non ha senso, non almeno con gli operatori "." e "->"
- **associazioni qualificate:** navigazione **indicizzata**
  - sintassi: `oggetto.navigazione[valoreQual,...]`

tipi enumerativi:

- definiti dall'utente con lo stereotipo `<<enumeration>>`
- sintassi per l'uso in espressioni OCL: `Tipo::idValore`

struttura generale dei tipi di collezioni

un **supertipo astratto Collection**

4 (sotto)**tipi concreti: Set, Bag, OrderedSet, Sequence**

questi **ereditano** dal supertipo:

operazioni **comuni** standard
operazioni con **significato variante**, ridefinito nei sottotipi
OrderedSet e **Sequence** lo **estendono** ulteriormente con altre **operazioni varianti**, definite solo per essi

i tipi collezione sono **polimorfi:**

il tipo degli elementi della collezione è generico
specificandolo, si hanno **tipi concreti monomorfi:**
ad es. **Set(Integer), Bag(Boolean), Sequence(String), Set(Studente), Sequence(OrderedSet(Integer)), Sequence(Sequence(String))**, etc.

operazioni comuni sulle collezioni

definite dal supertipo astratto **Collection**, dove
e è un oggetto del tipo degli elementi nella collezione
c, s sono collezioni di elementi dello stesso tipo

operazione	tipo del risultato
`c->count(e)`	**Integer**
`c->excludes(e)`	**Boolean**
`c->excludesAll(s)`	**Boolean**
`c->includes(e)`	**Boolean**
`c->includesAll(s)`	**Boolean**
`c->isEmpty()`	**Boolean**
`c->notEmpty()`	**Boolean**
`c->size()`	**Integer**
`c->sum()`	**Integer** o **Real** o ...

operazioni varianti su collezioni

operazioni varianti su sottotipi di **Collection**, dove
e è un oggetto del tipo degli elementi nella collezione
c, s sono collezioni "compatibili" di elementi dello stesso tipo

operazione	Set	OrderedSet	Bag	Sequence
=	Y	Y	Y	Y
<>	Y	Y	Y	Y
-	Y	Y	-	-
c->asBag()	Y	Y	Y	Y
c->asOrderedSet()	Y	Y	Y	Y
c->asSequence()	Y	Y	Y	Y
c->asSet()	Y	Y	Y	Y
c->excluding(e)	Y	Y	Y	Y
c->including(e)	Y	Y	Y	Y
c->flatten()	Y	Y	Y	Y
c->union(s)	Y	Y	Y	Y
c->intersection(s)	Y	-	Y	-
c->symmetricDifference(s)	Y	-	-	-

operazioni varianti su collezioni ordinate

le seguenti operazioni sono definite solo su collezioni ordinate, dove
e è un oggetto del tipo degli elementi nella collezione
i, l e u sono interi positivi (indici di posizione)

operazione	OrderedSet	Sequence
c->append(e)	Y	Y
c->at(i)	Y	Y
c->first()	Y	Y
c->indexOf(e)	Y	Y
c->insertAt(i,e)	Y	Y
c->last()	Y	Y
c->prepend(e)	Y	Y
c->subOrderedSet(l,u)	Y	-
c->subSequence(l,u)	-	Y

operazioni iterative sulle collezioni

hanno un **body parameter** : un'espressione OCL da valutare su ogni elemento della collezione
possono avere un altro parametro, la **variabile d'iterazione**, riferita all'elemento su cui si valuta il body, e che può occorrere nel body

sintassi: **opname**(E), oppure **opname**(v | E)

nella forma sintattica più semplice, eccole di seguito, dove

B è un'espressione di tipo **Boolean**
E è un'espressione OCL di qualsiasi tipo
O è un'espressione OCL di un tipo sul quale sia definita l'operazione <

any(B)
collect(E)
collectNested(E)
exists(B)
forAll(B)
isUnique(E)
iterate(...) (v. appresso)
one(B)
select(B)
reject(B)
sortedBy(O)

l'operazione iterate

è la più **generale** delle operazioni iterative
tutte le altre operazioni iterative possono definirsi per il tramite di **iterate**

sintassi:

```
c->iterate(element:Type1; result:Type2 = <expression>
                | <expression-with-element-and-result>)
```

esempi:

se **c** è di tipo **Collection(Integer)**:

```
c->sum() = c->iterate(i:Integer; somma:Integer = 0 | somma + i)
```

per ogni collezione **c** di tipo **Collection(t)** ed espressione booleana **B(x)**, con **x:t**

```
c->forAll(B(x)) = c->iterate(e:t; r:Boolean = true | r and B(e))
```

per qualsiasi tipo concreto di collezione **CT** (uno dei 4 sottotipi di **Collection**),
per ogni collezione **c** di tipo **CT(t)** ed espressione booleana **B(x)**, con **x:t**

```
c->select(B(x)) = c->iterate(e:t; r:CT(t) = CT{}
                        | if B(e) then r->including(e) else r endif
```

costrutti per postcondizioni

operatore suffisso **@pre**

variabile predefinita **result**

operazione booleana **oclIsNew()** su oggetto

operatore infisso ^ ("hasSent"), booleano:

> **oggetto^messaggio(...) = true** sse l'istanza contestuale ha inviato **messaggio(...)** a **oggetto** durante l'esecuzione dell'operazione

operatore infisso ^^ ("messaggi"):

> **oggetto^^pattern_messaggio(...)** dà la sequenza di messaggi (oggetti del tipo predefinito **OclMessage**), inviati a **oggetto** dall'istanza contestuale durante l'esecuzione dell'operazione, che corrispondono al **pattern_messaggio(...)**

conformità, casting, tipo universale

conversione di tipo: detta anche **casting**

se **e** è di tipo **t1**, e **t2** è un sottotipo di **t1**, allora la conversione di tipo per **e** può effettuarsi con la meta-operazione OCL **oclAsType(t:Type)** che ha un tipo come parametro:

> **e.oclAsType(t2)**

conformità di tipo: cf. principio di sostituzione (di Liskov)

se **t2** è un sottotipo di **t1**, allora è conforme a **t1**

la relazione di conformità fra tipi è riflessiva e transitiva

se **t2** è conforme a **t1**, allora **Collection(t2)** è conforme a **Collection(t1)**, e **CT(t2)** è conforme a **CT(t1)** per i 4 sottotipi concreti **CT** di **Collection**

sottotipi concreti distinti di **Collection** **non** sono conformi fra loro

ogni tipo è sottotipo di **OclAny** : il **tipo universale** in OCL