Architetture software ad oggetti

Modellazione e specifica di vincoli con OCL

Tipi e costrutti avanzati di OCL

rif.: J. Warmer & A. Kleppe, The Object Constraint Language 2nd Ed., Addison-Wesley, Pearson Education, 2003. Capp. 7–10

Architetture software

Università di Verona, Corso di Laurea in Informatica

G. Scollo, Verona, 23 febbraio 2006

Sommario

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. . .

operazione	notazione	tipo del risultato
disgiunzione	a **or** b	***Boolean***
congiunzione	a **and** b	***Boolean***
or esclusivo	a **xor** b	***Boolean***
negazione	**not** a	***Boolean***
eguaglianza	a **=** b	***Boolean***
ineguaglianza	a **<>** b	***Boolean***
implicazione	a **implies** b	***Boolean***

Table: operazioni standard di ***Boolean***

i tipi di base *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*** o ***Real***
sottrazione	a **–** b	***Integer*** o ***Real***
moltiplicazione	a ***** b	***Integer*** o ***Real***
divisione	a **/** b	***Real***
modulo	a.**mod**(b)	***Integer***
divisione intera	a.**div**(b)	***Integer***
valore assoluto	a.**abs**()	***Integer*** o ***Real***
massimo fra due	a.**max**(b)	***Integer*** o ***Real***
minimo fra due	a.**min**(b)	***Integer*** o ***Real***
miglior arrotondamento	a.**round**()	***Integer***
arrotondamento per difetto	a.**floor**()	***Integer***

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

il tipo *String*

sequenza di caratteri fra apici:

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

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***

Table: operazioni standard di ***String***

dove `s` e `t` sono di tipo ***String***
`m` e `n` sono di tipo ***Integer***

sintassi delle espressioni di base

per ridurre il numero di parentesi:

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`

Table: ***precedenza*** fra operatori OCL (in ordine decrescente)

operatori (binari) **infissi**:

operatori: `+, –, *, /, <, >, <=, >=, <>, =, 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 secondo 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 standard sulle collezioni

definite dal supertipo astratto ***Collection***

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 . . .

Table: operazioni standard di ***Collection***

dove e è un oggetto del tipo degli elementi nella collezione
c e s sono collezioni di elementi dello stesso tipo

operazioni varianti, su (quasi) tutte le collezioni

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	–	–	–

Table: operazioni varianti su sottotipi di ***Collection***

dove `e` è un oggetto del tipo degli elementi nella collezione
`c` e `s` sono collezioni "compatibili" di elementi dello stesso tipo

operazioni varianti definite solo su collezioni ordinate

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

Table: operazioni varianti 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)

operazioni iterative sulle collezioni

- hanno come parametro un'espressione OCL, il ***body parameter***, che valutano 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***(...)
  - ***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 ^ ("isSent"), booleano:
  oggetto^messaggio(...) = *true* se 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 e conversione di tipo, tipo universale

- **conversione di tipo:** detta anche ***casting***

  se e è di tipo `tipo1`, e `tipo2` è un sottotipo di `tipo1`, 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(tipo2)`

- **conformità di tipo:** cf. principio di sostituzione (di Liskov)
  - se `tipo2` è un sottotipo di `tipo1` allora è conforme a `tipo1`
  - la relazione di conformità fra tipi è riflessiva e transitiva
  - se `tipo2` è conforme a `tipo1`, allora `Collection(tipo2)` è conforme a `Collection(tipo1)`, e `CT(tipo2)` è conforme a `CT(tipo1)` 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