Design of Information Systems

OCL Collection Concepts and Collection Operations

Martin Gogolla
University of Bremen, Germany
Database Systems Group
Collections

- Common in modeling and programming languages
- "A collection (or container) is a grouping of some variable number of data items (possibly zero) that ... need to be operated upon together in some controlled fashion." Wikipedia
- Examples: set, list, multi-set (allowing duplicates), stack, ...
- UML collections: Set, Bag, Sequence, OrderedSet, Tuple
- Parametrized with element type(s) and access option (for Tuple)
Example collections in SocialNetwork

merkel.inviter:  \texttt{Set}(Profile)

merkel.posting:  \texttt{Set}(Posting)

merkel.posting.commenter:  \texttt{Bag}(Profile)

-- !create merkel,putin,trump:Profile
Sequence\{merkel,putin,trump\}:  \texttt{Sequence}(Profile)

OrderedSet\{merkel,putin,trump\}:  \texttt{OrderedSet}(Profile)

Sequence\{merkel,putin,trump,may\}.yearE = Sequence\{2005,2000,2016,2016\}
-- yearE:  year of first election; imaginable for example model


-- Paper::authors:OrderedSet(Author); more precise than Sequence(Author)

Sequence\{may,merkel\}->collect(p|\texttt{Tuple}\{L:p.lastN,I:p.initials\}) =
  Sequence\{\texttt{Tuple}\{L='May', I='TM'\},\texttt{Tuple}\{L='Merkel', I='AM'\}\}:  \texttt{Sequence}(\texttt{Tuple}(L:\text{String},I:\text{String}))
Example collections in ConferenceWorld
Collection parameters and collection syntax

- Type kinds with type parameters: Set(T), Bag(T), Sequence(T), OrderedSet(T), Tuple(A1:T1,...,An:Tn); access Ai

- Abstract type kind (no instances) Collection(T) generalization of Set(T), Bag(T), Sequence(T), OrderedSet(T)

- Parameter actualization in order to build types

- Types always written with parentheses ( )

\[
\text{Set(Posting)}, \text{Bag(Profile)},
\text{Sequence(Profile)}, \text{OrderedSet(Integer)},
\text{Tuple}(L:\text{String},I:\text{String})
\]

- Instantiations always written with braces { }

\[
\text{Set}\{\text{merkel, trump}\}, \text{Bag}\{\text{trump, putin, trump}\},
\text{Sequence}\{\text{merkel, putin, trump}\}, \text{OrderedSet}\{2005, 2000, 2016\},
\text{Tuple}\{L='Merkel', I='AM'\}
\]

- Tuple access Tuple\{L='Merkel', I='AM'\}.I='AM'
Collection properties (for homogeneous collections)

- Two criteria in order to distinguish between collections: Insertion order and insertion frequency

- Is the insertion order relevant for distinguishing collections?
  \[
  \text{COL} \rightarrow \text{including}(E_1) \rightarrow \text{including}(E_2) = \text{COL} \rightarrow \text{including}(E_2) \rightarrow \text{including}(E_1)
  \]
  if required, collection is called order-blind, else order-aware

- Is the insertion frequency relevant for distinguishing collections?
  \[
  \text{COL} \rightarrow \text{includes}(E) \implies \text{COL} \rightarrow \text{including}(E) = \text{COL}
  \]
  if required, collection is called frequency-blind, else frequency-aware

<table>
<thead>
<tr>
<th>frequency</th>
<th>order</th>
<th>blind</th>
<th>aware</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>blind</td>
<td>Set(T)</td>
<td>OrderedSet(T)</td>
</tr>
<tr>
<td></td>
<td>aware</td>
<td>Bag(T)</td>
<td>Sequence(T)</td>
</tr>
</tbody>
</table>
Collection type hierarchy and properties

- order-blind and frequency-blind: Set(T)
- order-blind and frequency-aware: Bag(T)
- order-aware and frequency-aware: Sequence(T)
- order-aware and frequency-blind: OrderedSet(T)

OCL 1.3 only had Set(T), Bag(T), Sequence(T)
OCL 1.4 added OrderedSet(T)

Also used: order-insensible/-sensible, frequency-insensible/-sensible
Collection properties: Insertion order and frequency

\[
\text{Collection}\{7,8\} \xrightarrow{?} \text{Collection}\{8,7\}
\]

\[
\text{Collection}\{7,8,7\}
\]

\[
\text{Set}\{7,8\} = \text{Set}\{8,7\}
\]

\[
\text{Set}\{7,8,7\}
\]

\[
\text{OrderedSet}\{7,8\} \not\sim \text{OrderedSet}\{8,7\}
\]

\[
\text{OrderedSet}\{7,8,7\}
\]

\[
\text{Bag}\{7,8\} \not\sim \text{Bag}\{8,7\}
\]

\[
\text{Bag}\{7,8,7\}
\]

\[
\text{Sequence}\{7,8\} \not\sim \text{Sequence}\{8,7\}
\]

\[
\text{Sequence}\{7,8,7\}
\]

**Properties:**

- \(C \sim \text{includes}(E) \implies C \sim \text{including}(E) = C\)
- \(C \sim \text{including}(E_1) \sim \text{including}(E_2) = C \sim \text{including}(E_2) \sim \text{including}(E_1)\)
Collection properties

use> !C:=Set{Set{7,8}, Set{8,7},
    Set{7,8,8}, Set{8,7,7}}

use> ?C
    Set{Set{7,8}} : Set(Set(Integer))

use> !D:=Set{Bag{7,8}, Bag{8,7},
    Bag{7,8,8}, Bag{8,7,7}}

use> ?D
    Set{Bag{7,8}, Bag{7,7,8}, Bag{7,8,8}} : Set(Bag(Integer))

use> !E:=Set{OrderedSet{7,8}, OrderedSet{8,7},
    OrderedSet{7,8,8}, OrderedSet{8,7,7}}

use> ?E
    Set{OrderedSet{7,8}, OrderedSet{8,7}} : Set(OrderedSet(Integer))

use> !F:=Set{Sequence{7,8}, Sequence{8,7},
    Sequence{7,8,8}, Sequence{8,7,7}}

use> ?F
    Set{Sequence{7,8}, Sequence{8,7},
    Sequence{7,8,8}, Sequence{8,7,7}} : Set(Sequence(Integer))

use> ?Sequence{C->size(), D->size(), E->size(), F->size()}
    Sequence{1, 3, 2, 4} : Sequence(Integer)
Collection operations on all collection kinds

Constructors and `destructors'
- Set{...}, Bag{...}, Sequence{...}, OrderedSet{...}
- including(...), excluding(...)

Basic boolean and integer query operations
- =, <>
- includes(...), excludes(...), includesAll(...), excludesAll(...)
- isEmpty(), notEmpty(), size(), count(...)

Advanced boolean query operations
- forAll(...), exists(...), one(...)
- isUnique(...)

Advanced collection-valued query operations
- select(...), reject(...)
- any(...)
- union(...)
- collect(...), collectNested(...)
- flatten()
- sortedBy(...)

Complex query operations: iterate(...), closure(...)

Coercions: asSet(), asBag(), asSequence(), asOrderedSet()
Collection operations on special collection kinds

- `first()`, `last()`, `at(pos)`, `reverse()`
  for order-aware, i.e. `Sequence(T)`, `OrderedSet(T)`

- `subSequence(startPos,endPos)` on `Sequence(T)`

- `subOrderedSet(startPos,endPos)` on `OrderedSet(T)`

- `intersection(...)` for order-blind, i.e. `Set(T)`, `Bag(T)`

- `sum()`, `min()`, `max()` on `Collection(Integer)`, `Collection(Real)`

- Few further operations (e.g. `indexOf`): see OCL standard

Not mentioned yet (and to be discussed further down):
collection operations in the context of `generalization`
(e.g. for Chess example, `c:Character` and `c.oclIsTypeOf(Knight)`)

Constructors and `destructors'
- Set\{7,8\}, Bag\{7,8,8\}, Sequence\{7,8,7\}, OrderedSet\{8,7,7\}
- Set{}, Bag{}, Sequence{}, OrderedSet{}
- Set\{7..9\}, Bag\{7..9\}, Sequence\{7..9\}, OrderedSet\{7..9\}
- Set{}->including(8)->including(7), Bag\{8,9,7,8,9\}->excluding(9)

Basic boolean and integer query operations
- Set\{7,8\}=Set\{8,7,8,7\}, OrderedSet\{7,8\}<>OrderedSet\{8,7\}
  Set\{7,8\}<>Bag\{7,8\}, OrderedSet\{7,8\}<>Sequence\{8,7\}
- Set\{7,8\}->includes(8), Set\{7,8\}->excludes(9),
  Set\{7,8\}->includesAll(Set\{8,8,7,7\}), Set\{7,8\}->excludesAll(Set\{6,9\})
- Set{}->isEmpty(), Set\{7,8\}->notEmpty(), Set\{8,8,7,7\}->size()=2
  Set\{7,8,7\}->count(7), Bag\{7,8,7\}->count(7)
  Sequence\{7,8,7\}->count(7), OrderedSet\{7,8,7\}->count(7)
Demonstrating OCL expressions without having objects (Part B)

Advanced boolean query operations
- Set{7..9}→\textbf{forall}(i|i \geq 0), Bag{7..9}→\textbf{exists}(i|i \mod(2) = 0)
- Sequence{7..9}→\textbf{one}(i|i \mod(2) = 0)
- OrderedSet{-9..-8}→\textbf{including}(8)→\textbf{including}(9)→\textbf{isUnique}(i|i*i) = \text{false}

Advanced collection-valued query operations
- Set{21..42}→\textbf{select}(i|i \mod(3) = 0 \text{ and } i \mod(7) = 0)
- Bag{21..42}→\textbf{reject}(i|i \mod(2) = 0 \text{ or } i \mod(3) = 0)
- Set{21..42}→\textbf{any}(i|i \mod(2) = 1)
- Set{7,8,8}→\textbf{union}(Set{9,9,8}), Bag{7,8,8}→\textbf{union}(Bag{9,9,8})
  Sequence{7,8,8}→\textbf{union}(Sequence{9,9,8})
  OrderedSet{7,8,8}→\textbf{union}(OrderedSet{9,9,8})
- Set{-2..2}→\textbf{collect}(i|i*i), Set{-2..2}→\textbf{collect}(i|Sequence{i,i*i})
  Set{-2..2}→\textbf{collectNested}(i|Sequence{i,i*i})
- Set{-2..2}→\textbf{collectNested}(i|Sequence{i,i*i})→\textbf{flatten}()
- Set{-6,-5,-4,7,8,9}→\textbf{sortedBy}(i|i*i)
Demonstrating OCL expressions without having objects (Part C)

Complex query operations
- `Set{-2..2} -> iterate(i:Integer;r:Set(Sequence(OclAny))=Set{}} |` 
  `r -> including(Sequence{i,i*i,if i.mod(2)=0 then 'E' else 'O' endif}))`
- Capitals: M[adrid], P[aris], A[msterdam], B[erlin], Z[urich], V[ienna]
  let TupleSet = 
  `Set{Tuple{s:'M',t:'P'},Tuple{s:'P',t:'A'},Tuple{s:'A',t:'B'},` 
  `Tuple{s:'M',t:'Z'},Tuple{s:'Z',t:'V'},Tuple{t:'B',s:'V'}} in`
  TupleSet -> closure(T1)`
  `TupleSet -> select(T2 | T1.t = T2.s) ->` 
  `collect(T2 | Tuple{s:T1.s,t:T2.t}) ->` 
  `asSet())`
Demonstrating OCL expressions without having objects (Part D)

Coercions
- Sequence{8,7,8}->asSet()=Set{8,7}
- OrderedSet{8,7,8}->asBag()=Bag{8,7}
- Set{7,8}->asSequence()=Sequence{8,7}
  or Set{7,8}->asSequence()=Sequence{7,8}
- Bag{8,8,7,7}->asOrderedSet()=OrderedSet{7,8}
  or Bag{8,8,7,7}->asOrderedSet()=OrderedSet{8,7}
- Set{-2..2}->collect(i|i*i)->asSet()
Thanks for your attention!