Recursion and iteration support in USE Validator with AnATLyzer

Jesús Sánchez Cuadrado
(jesus.sanchez.cuadrado@uam.es)

Miso - http://www.miso.es
Universidad Autónoma de Madrid

September 30th, 2015
OCL'15@MoDELS - Ottawa

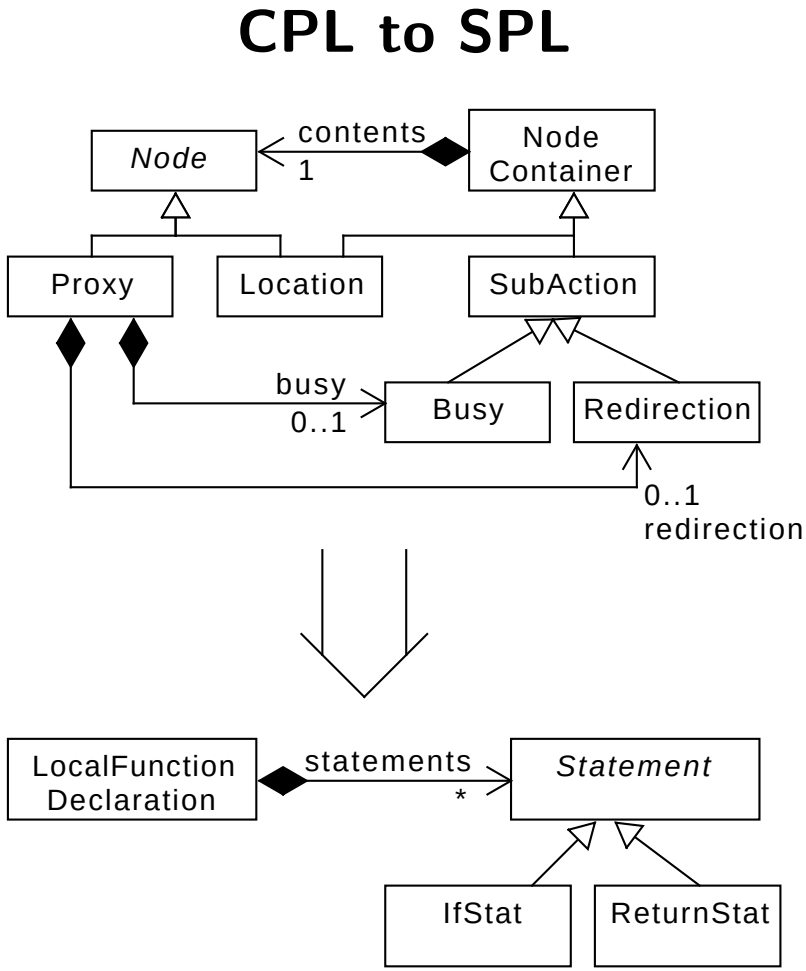
Content

Content

Motivation
Context

- anATLyzer: static analysis tool for ATL
  - Model finding techniques to improve accuracy
  - Implementation: USE Validator
- Fault localization process:
  1. Type checking
  2. Rule analysis
  3. Some problems marked as potential
  4. Extract path condition
  5. Run USE Validator to find a witness model that confirms the problem

Motivation
Example - CPL to SPL



Motivation
Example - CPL to SPL


```
-- Every Node is an statement, except a Location
helper context CPL!Node def: statement : CPL!Node =
        self;

helper context CPL!Location def: statement : CPL!Node =
        self.contents.statement;

rule SubAction2Function {
        from s : CPL!SubAction ( s.contents.oclIsKindOf(CPL!Location) )
        to t : SPL!LocalFunctionDeclaration (
          -- Is this binding fully covered by resolving rules?
          statements ← s.contents.statement
        )
}

rule Proxy2Return {
        from proxy : CPL!Proxy ( proxy.redirect.oclIsUndefined() )
        to ret : SPL!ReturnStat (
                ...
        )
}
```

Motivation
Example - CPL to SPL

Path condition

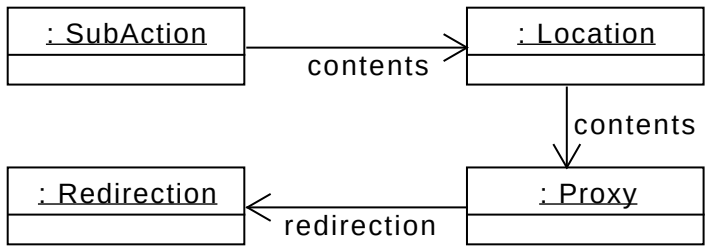
```
SubAction.allInstances()→
  select(s | s.contents.oclIsKindOf(Location))→
  exists(s |
    let problem = s.contents.statement in
      not problem.isUndefined() and
      not (if problem.oclIsKindOf(Proxy) then
             let proxy = problem.oclAsType(Proxy)
              in proxy.redirect.oclIsUndefined()
           else
             false
           endif))
```

Motivation
Example - CPL to SPL

Path condition

```
SubAction.allInstances()→
  select(s | s.contents.oclIsKindOf(Location))→
  exists(s |
    let problem = s.contents.statement in
      not problem.isUndefined() and
      not (if problem.oclIsKindOf(Proxy) then
              let proxy = problem.oclAsType(Proxy)
               in proxy.redirect.oclIsUndefined()
            else
              false
            endif))
```

Witness model



Problem

anATLyzer is limited by the unsupported features of USE Validator, notably for:

- Recursion
- *iterate*
- Sequences

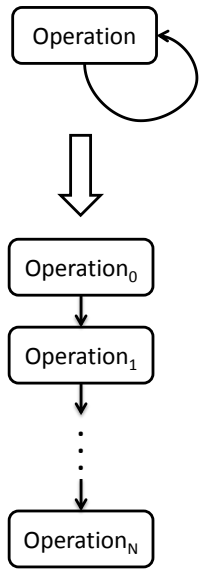
Goal

Provide support for recursion and iterate to anATLyzer by rewriting the OCL code fed into USE Validator.

Content

Unfolding recursion
Approach

1. Copy the original operation n times
2. Version i invokes version $i + 1$
3. The last operation returns some bottom value
4. Note: take polymorphic calls into account
   - Copy every possible invokable operation in the class hierarchy



Unfolding recursion
Approach

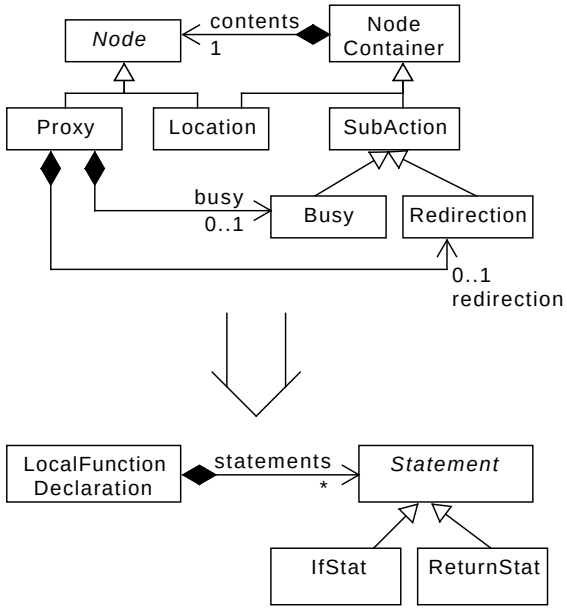
N = Number of unfoldings
OP = Original operation

$OP_0 = OP$
for i = 1 **to** N
  CS_{i-1} = callSites(OP_{i-1})
  foreach cs in CS_{i-1}
    cs.operationName = OP.operationName + "_" + i
  end

  OP_i = copy(OP)
  OP_i.operationName = OP.operationName + "_" + i
end

OP_N.body = OclUndefined

Unfolding recursion
Approach



–– Every Node is an statement, except a Location

helper context CPL!Node **def**: statement : CPL!Node =
    self;

helper context CPL!Location **def**: statement : CPL!Node =
    self.contents.statement;

Unfolding recursion
Approach


```
abstract class Node
operations
  statement() : Node = self
  statement_1() : Node = self
  statement_2() : Node = self
  statement_3() : Node = self
end

class Location < Node, NodeContainer
operations
  statement() : Node = self.contents.statement_1()
  statement_1() : Node = self.contents.statement_2()
  statement_2() : Node = self.contents.statement_3()
  statement_3() : Node = OclUndefined
end
```

Content

Unfolding iterate
What is iterate?

Iterate

General iteration operation:

col−>iterate(it; acc = <init> | <body>)

Example (Select implemented with iterate)

−− Given: col−>select(it | <body>) where col is a Set

```
col−>iterate(it; acc = Set { } |
  if <body> then
    acc−>including(it)
  else
    acc
  endif)
```

Unfolding iterate
Approach

Unfold n-times:

1. For each call to *iterate*, generate n "iteration step operation"
2. Each iteration step operation:
   - Check if the collection is empty, and returns *acc* if so.
   - Pick an element of the collection
   - Compute the rest of the collection
   - Evaluate the body to compute *acc*
   - Invoke the next operation

Unfolding iterate
Approach


```
-- Given an expression: col->iterate(it : T_it; acc : T_acc = <init> | <body>)
-- where:
--      T_col: the type of the elements of the collection
--     T_it: the type of the iteration variable, which must be compatible with T_col
--      T_acc: the type of the accumulator variable
class ThisModule
operations
  def iterate_aux_i(col : Set(T_col), acc : T_acc) : T_acc =
    if col->isEmpty() then
      acc
    else
      let it : T_it = col->any(_ | true) in
      let rest : Set(T_col) = col->select(v | v <> it) in
      let value : T_col = <body>
       in iterate_aux_{i+1}(rest, value)
    endif
  ...

  def iterate_aux_n(col : Set(T_col), acc : T_acc) : T_acc = OclUndefined

end
```

Content

Discussion

- First tests indicate that the approach may be useful.
  - Further experiments are needed.
  - How many unfoldings are typically enough?
- How to represent *bottom* in OCL?
  - OclUndefined is the only option in USE Validator.
  - OclInvalid is a promising candidate: *its unique instance conforms to any other type but any call applied to it results in invalid itself*.
- Additional problems related to the lack of support for sequences (e.g., at, first, etc.).

Content

Demo

- anATLyzer in action.
- Checking a problem with recursion with USE

The End

Thank you!

Any questions?

Additional information:

- **AnATLyzer**: `http://www.miso.es/tools/anATLyzer.html`