Automated Size Analysis for OCL

Fang Yu, Tevfik Bultan, Erik Peterson

Department of Computer Science
University of California, Santa Barbara
Overview

• Target: Object Constraint Language
• Idea: Size Abstraction
• Method: Infinite State Model Checking
• Experiments: Verification of Java Card API Specifications
What is Object Constraint Language (OCL)?

- A specification language for describing constraints on object oriented models
- Developed at IBM and now part of the UML standard
- Used to describe precise constraints that cannot be presented in UML diagrams:
  - class invariants on attributes and associations
  - pre and post conditions of class methods
OCL

Why shall we care about OCL?

- UML/OCL is a widely adopted industry standard at the design stage
  - Object Management Group (OMG) standard
- Identifying design errors before the implementation stage is cost effective

There is a lack of automatic verification tools to check the correctness of OCL specifications!
OCL Collection

OCL allows collections of arbitrary types and this makes automatic verification difficult.

There are three basic collection types in OCL:

- A *Set* is a collection that contains instances of a valid OCL type but does not contain duplicate elements
- A *Bag* is like a *Set*, but it can contain duplicate elements
- A *Sequence* is like a *Bag* but the elements are ordered
OCL Example

From OCL Specifications of Java Card API 2.1.1

Class Invariant:

self.thrownExceptions->isEmpty() implies (self.theAID->size() >= 5 and self.theAID->size() <= 16)

Method Specification:

context AID::equal(anObject: Set(Integer)): Boolean
    post: self.thrownExceptions = self.thrownExceptions@pre
         and (result = (anObject->asSequence = self.theAID))

context AID::getBytes(dest: Sequence(Integer), offset: Integer, e: Integer): Integer

    post: ...
         self.theAID = dest->
         subSequence(offset, offset+self.theAID->size())
         ...

ESEC/FSE07 – p. 6/28
Size Analysis

OCL Size Analysis Framework:

Size Abstraction:

- Converts OCL expressions on collection types into arithmetic constraints on their sizes
  - Abstracts away the contents of the collections

Action Language Verifier (ALV):

- Composite model checking (BDD + Polyhedra + DFA)
- Infinite state verification using conservative approximation techniques (e.g., widening, bounded fixpoint computations)
Size Abstraction

Since we abstract away the contents, size constraints may not be precise.

For example, the size constraints generated for the expression $o_1 \rightarrow \text{union}(o_2)$:

- if $o_1, o_2$ are $Set$, then $\max(o_1.v, o_2.v) \leq o.v \leq o_1.v + o_2.v$
- if $o_1, o_2$ are $Bag$ or $Seq$, then $o.v = o_1.v + o_2.v$

where

- $o$ denotes the collection that is the result of the expression (which is the union of $o_1$ and $o_2$)
- $o.v, o_1.v, o_2.v$ denote the sizes of the collections
# Size Constraints

<table>
<thead>
<tr>
<th>OCL Expression</th>
<th>Type $o, o_1, o_2$</th>
<th>Size Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>$o_1$-&gt;including($e$)</td>
<td>$s, s$</td>
<td>$o_1.v \leq o.v \leq o_1.v + 1 \land (o_1.v = 0 \Rightarrow o.v = 1) \land o_1.c$</td>
</tr>
<tr>
<td></td>
<td>$m, m$</td>
<td>$o.v = o_1.v + 1 \land o_1.c$</td>
</tr>
<tr>
<td>$o_1$-&gt;append($e$)</td>
<td>$m, m$</td>
<td>$o.v = o_1.v + 1 \land o_1.c$</td>
</tr>
<tr>
<td>$o_1$-&gt;excluding($e$)</td>
<td>$s, s$</td>
<td>$max(0, o_1.v - 1) \leq o.v \leq o_1.v \land o_1.c$</td>
</tr>
<tr>
<td></td>
<td>$m, m$</td>
<td>$max(0, o_1.v - 1) \leq o.v \leq o_1.v \land o_1.c$</td>
</tr>
<tr>
<td>$o_1$-&gt;union($o_2$)</td>
<td>$s, s, s$</td>
<td>$max(o_1.v, o_2.v) \leq o.v \leq o_1.v + o_2.v \land o_1.c \land o_2.c$</td>
</tr>
<tr>
<td></td>
<td>$m, s/m, m/s$</td>
<td>$o.v = o_1.v + o_2.v \land o_1.c \land o_2.c$</td>
</tr>
<tr>
<td>$o_1$-&gt;intersection($o_2$)</td>
<td>$s, s/m, m/s$</td>
<td>$0 \leq o.v \leq min(o_1.v, o_2.v) \land o_1.c \land o_2.c$</td>
</tr>
<tr>
<td></td>
<td>$m, m, m$</td>
<td>$0 \leq o.v \leq min(o_1.v, o_2.v) \land o_1.c \land o_2.c$</td>
</tr>
<tr>
<td>$o_1$-$o_2$</td>
<td>$s, s, s$</td>
<td>$max(0, o_1.v - o_2.v) \leq o.v \leq o_1.v \land o_1.c \land o_2.c$</td>
</tr>
<tr>
<td>$o_1$-&gt;subSequence($i_1, i_2$)</td>
<td>$m, m$</td>
<td>$(o_1.v \geq i_2 \geq i_1 \land o.v = i_2 - i_1 + 1) \land o_1.c$</td>
</tr>
<tr>
<td>$o_1$-&gt;at($i$)</td>
<td>$m, m$</td>
<td>$(o_1.v \geq i \geq 0 \Rightarrow o.v = 1) \land o_1.c$</td>
</tr>
<tr>
<td>$o_1$-&gt;asSet</td>
<td>$s, s$</td>
<td>$o.v = o_1.v \land o_1.c$</td>
</tr>
<tr>
<td></td>
<td>$s, m$</td>
<td>$((o_1.v &gt; 0 \land 1 \leq o.v \leq o_1.v) \lor (o_1.v = o.v = 0)) \land o_1.c$</td>
</tr>
</tbody>
</table>
OCL Specification

class OwnerPIN::update(newpin: Sequence(Integer),
offset:Integer,length: Integer, e:Integer)

pre: newpin->notEmpty()
    and offset >= 0
    and offset+length <= newpin->size()
    and length >= 0

post:

1: thrownExceptions=thrownExceptions@pre
2: and self.pin->subSequence(0,length)
    =newpin->subSequence(offset, offset+length)
)or(
3: thrownExceptions=thrownExceptions@pre->including(e)
4: and length > self.maxPINSize
)or(
5: thrownExceptions=thrownExceptions@pre->including(e)
6: and systemInstance->notEmpty()
)
Size Constraints

newpin->notEmpty()
newpin.size > 0

self.pin->subSequence(0, length)
result.size = length - 0 + 1 and pin.size >= length and length >= 0

thrownExceptions = thrownExceptions@pre->including(e)
thrownExceptions.size = result.size and result.size = thrownExceptions@pre.size + 1
module updateMod()
    updateMod:
        pre: newpin > 0 and offset >= 0 and length + offset <= newpin
        and length >= 0 and
        post:(
            1: (thrownExceptions’ = thrownExceptions
            2: and tmp8 = tmp9
                and tmp8 = length - 0 + 1 and pin’ >= length and length >= 0
                and tmp9 = length + offset - offset + 1
                and newpin’ >= length + offset and length + offset >= offset)
            or (  
            3: thrownExceptions’ = tmp10 and tmp10 = thrownExceptions + 1
            4: and length > maxPINSize’
            ) or (  
            5: thrownExceptions’ = tmp11 and tmp11 = thrownExceptions + 1
            6: and systemInstance’ > 0)
            ); endmodule
OCL Formalization

An OCL class specification is $C = (P, A, M)$, where

- **P**: a set of properties representing class invariants
- **A**: a set of attributes (fields of the class)
- **M**: a set of methods. For each $m \in M$,
  - $m.pre : A$ is pre condition
  - $m.post : A \times A@pre$ is post condition

We define the formal semantics of OCL as a transition system.
Transition System

The corresponding transition system is $\|C\| = (S, I, R)$, where $S$ is the set of states, $I \subseteq S$ is the initial states, and $R$ is the transition relation, and

- $S = \text{dom}(A_1) \times \ldots \times \text{dom}(A_n)$
- For all $p \in P$, $I \subseteq \|p\|
- $R_m = \{(s_1, s_2) \mid s_1 \in \|m.pre\| \land (s_1, s_2) \in \|m.post\|\}$
  - $R = \bigcup_{m \in M} R_m$
  - Given a set of states $Q \subseteq S$, $R(Q) = \{s_2 \mid \exists s_1 \in Q, (s_1, s_2) \in R\}$
  - $R^*$ denotes the reflexive transitive closure of $R$, and $R^*(I)$ is the set of all reachable states in the transition system
OCL Correctness

An OCL class specification is correct if and only if its class invariants are consistent with the pre and post conditions of its methods.

We formalize this as follows:

**Definition:**

An OCL class specification $C = (P, A, M)$ with the corresponding transition system $\|C\| = (I, S, R)$ is correct, if and only if, all the reachable states of the class satisfy all the class invariants, i.e., for all $p \in P$, $R^*(I) \subseteq \|p\|$.
Reachability Analysis

- Concrete States: $I, R(I), R^*(I), ||P||$
- Abstract States (Over Approximation): $I', R'(I'), R'^*(I'), ||P'||$
- $R^*(I) \subseteq ||P||$, correct
- $R'^*(I') \subseteq ||P'||$, correct? We do not know.
- We use $\neg||\neg P||'$ as an under approximation of $||P||$.  

ESEC/FSE07 – p. 16/28
Size Abstraction

Size abstraction satisfies the following properties:

\[ \forall s \in S, s \in I \Rightarrow \text{abs}(s) \in \text{abs}(I) \]
\[ \forall s_1, s_2 \in S, (s_1, s_2) \in R \Rightarrow (\text{abs}(s_1), \text{abs}(s_2)) \in \text{abs}(R) \]
\[ \forall s \in S, \forall p \in P, \text{abs}(s) \in \|\neg\text{abs}(\neg p)\| \Rightarrow s \in \|p\| \]

Based on these properties, we can prove the following theorem:

**Theorem:**
Given a class specification \( C \), if \( \text{abs}(C) \) is correct, then \( C \) is correct.
Case Study: Java Card API

We applied our technique to verification of the Java Card Application Programming Interface (API) specifications.

- Java Card is a platform for developing applications that run on smart cards
- The OCL specifications of Java Card API 2.1.1 are given by [Larsson and Mostowski, ENTCS04].
- These include 31 classes and 150 methods in javacard.framework, javacard.security, javacard.framework.service, and javacardx.crypto packages.
Case Study: Java Card API

Almost any type of smart card can benefit from Java Card technology:

- Subscriber Identity Module (SIM) cards, used in cell phones on most wireless networks
- Financial cards supporting both online and offline transactions
- Government and health-care identity cards
- Smart tickets for mass transit

However, if there are errors in the Java Card API specifications, these applications may be vulnerable!
Identified Errors

We identify the following types of errors in the OCL specifications of the Java Card API:

- **Frame Error (FE):**
  - missing frame constraints, such as a=a@pre

- **Unsound Implication (UI):**
  - In an implication structure, i.e., $\bigwedge_i b_i \rightarrow s_i$, if $\bigvee_i b_i$ is not universal, the undefined state $(\bigwedge_i \neg b_i)$ is allowed to make unexpected transitions

- **Design Error (DE):**
  - missing pre
  - using an unrestricted parameter to define a restricted variable
Verification Results

All class specifications of Java Card API are either verified (26/31) or falsified (5/31) within 10 secs and 20MB.

![Graphs showing verification results](image)
Details for Falsified Classes

AID class:

<table>
<thead>
<tr>
<th>Method</th>
<th>Err.</th>
<th>R</th>
<th>trans+ver.</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>AID</td>
<td>None</td>
<td>V</td>
<td>0.02+0.09s</td>
<td>2523k</td>
</tr>
<tr>
<td>equal</td>
<td>(FE)</td>
<td>V</td>
<td>0s+0s</td>
<td>299k</td>
</tr>
<tr>
<td>equals</td>
<td>UI</td>
<td>F</td>
<td>0.02s+0.02</td>
<td>610k</td>
</tr>
<tr>
<td>getBytes</td>
<td>DE</td>
<td>F</td>
<td>0.02s+0.02s</td>
<td>676k</td>
</tr>
<tr>
<td>getPartialBytes</td>
<td>(FE)</td>
<td>V</td>
<td>0.01s+0.02s</td>
<td>418k</td>
</tr>
<tr>
<td>partialEquals</td>
<td>UI</td>
<td>F</td>
<td>0.02s+0.01s</td>
<td>545k</td>
</tr>
<tr>
<td>RIDEquals</td>
<td>(FE)</td>
<td>V</td>
<td>0s+0s</td>
<td>324k</td>
</tr>
</tbody>
</table>

JCSystem Class:

<table>
<thead>
<tr>
<th>Method</th>
<th>Err.</th>
<th>R</th>
<th>trans+ver.</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>abortTransaction</td>
<td>None</td>
<td>V</td>
<td>0s+0.01s</td>
<td>266k</td>
</tr>
<tr>
<td>beginTransaction</td>
<td>(FE)</td>
<td>V</td>
<td>0s+0.06s</td>
<td>266k</td>
</tr>
<tr>
<td>commitTransaction</td>
<td>(FE)</td>
<td>V</td>
<td>0s+0.01s</td>
<td>266k</td>
</tr>
<tr>
<td>getAppletSharable-ObjectInterface</td>
<td>UI</td>
<td>F</td>
<td>0.06s+0.03s</td>
<td>815k</td>
</tr>
<tr>
<td>getTransactionDepth</td>
<td>(FE)</td>
<td>V</td>
<td>0s+0s</td>
<td>270k</td>
</tr>
<tr>
<td>isTransient</td>
<td>(FE)</td>
<td>V</td>
<td>0s+0.01s</td>
<td>270k</td>
</tr>
<tr>
<td>lookupAID</td>
<td>(FE)</td>
<td>V</td>
<td>0.03s+0.07s</td>
<td>1028k</td>
</tr>
<tr>
<td>MakeTransientBooleanArray</td>
<td>(FE)</td>
<td>V</td>
<td>0.09s+1.61s</td>
<td>1147k</td>
</tr>
<tr>
<td>MakeTransientByteArray</td>
<td>(FE)</td>
<td>V</td>
<td>0.06s+1.73s</td>
<td>1487k</td>
</tr>
<tr>
<td>MakeTransientObjectArray</td>
<td>(FE)</td>
<td>V</td>
<td>0.06s+1.72s</td>
<td>1495k</td>
</tr>
<tr>
<td>MakeTransientShortArray</td>
<td>(FE)</td>
<td>V</td>
<td>0.07s+1.72s</td>
<td>950k</td>
</tr>
</tbody>
</table>

KeyEncryption Class:

<table>
<thead>
<tr>
<th>Method</th>
<th>Err.</th>
<th>R</th>
<th>trans+ver.</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>getKeyCipherMod</td>
<td>(FE)</td>
<td>V</td>
<td>0s+0s</td>
<td>115k</td>
</tr>
<tr>
<td>setKeyCipherMod</td>
<td>DE</td>
<td>F</td>
<td>0s+0s</td>
<td>123k</td>
</tr>
</tbody>
</table>

OwnerPin/Pin Class:

<table>
<thead>
<tr>
<th>Method</th>
<th>Err.</th>
<th>R</th>
<th>trans+ver.</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>getValidatedFlag</td>
<td>(FE)</td>
<td>V</td>
<td>0s+0.01s</td>
<td>385k</td>
</tr>
<tr>
<td>setValidatedFlag</td>
<td>(FE)</td>
<td>V</td>
<td>0.01s+0s</td>
<td>381k</td>
</tr>
<tr>
<td>OwnerPIN</td>
<td>(FE)</td>
<td>V</td>
<td>0.01s+0.05s</td>
<td>590k</td>
</tr>
<tr>
<td>update</td>
<td>(FE)</td>
<td>V</td>
<td>0.02s+0.7s</td>
<td>782k</td>
</tr>
<tr>
<td>resetAndUnblock</td>
<td>(FE)</td>
<td>V</td>
<td>0s+0.01s</td>
<td>381k</td>
</tr>
<tr>
<td>getTriesRemaining</td>
<td>(FE)</td>
<td>V</td>
<td>0.01s+0s</td>
<td>385k</td>
</tr>
<tr>
<td>isValidated</td>
<td>(FE)</td>
<td>V</td>
<td>0.01s+0s</td>
<td>381k</td>
</tr>
<tr>
<td>reset</td>
<td>None</td>
<td>V</td>
<td>0.01s+0s</td>
<td>381k</td>
</tr>
<tr>
<td>check</td>
<td>UI</td>
<td>F</td>
<td>0.03s+0.06s</td>
<td>877k</td>
</tr>
</tbody>
</table>
Example Errors

Class Invariant:

\[ \text{self.thrownExceptions->isEmpty() implies } \text{(self.theAID->size() } \geq 5 \text{ and self.theAID->size() } \leq 16) \]

Method Specification:

\text{context AID::equal(anObject: Set(Integer)): Boolean}

\text{post: self.thrownExceptions = self.thrownExceptions@pre}

\text{and (result = (anObject->asSequence= self.theAID))}

\textbf{To fix:} Add self.theAID=self.theAID@pre

\text{context AID::getBytes(dest: Sequence(Integer), offset: Integer, e: Integer): Integer}

\text{post: ...}

\text{self.theAID = dest->}

\text{subSequence(offset, offset+self.theAID->size())}

\text{...}

\textbf{To fix:} Change to self.theAID = dest->subSequence(offset, offset+self.theAID@pre->size())
Pros and Cons

Our size analysis tool

- Is more automated than tools that use theorem proving
  - KeY Tool [Ahrendt et al. SSM05]
- Provides stronger guarantees than simulation
  - USE, OCLE [Gogolla, Bohling, Richters, UML03]
  - or bounded verification
  - Alloy [Jackson, ACM TOSEM02]

No free lunch!! We focus on size properties rather than providing a general verification framework.
Size Does Matter

Analyzing size properties is important and promising for several reasons:

- No extra specification effort needed from the software developers
- Violation of size properties is the cause of security vulnerabilities such as buffer overflows
- Effective automated verification can be achieved for size properties
Related Work

Size Analysis:

• Size Types [Hughes, Pareto, Sabry, POPL96], [Chin et al., ICSE05]

• C String Static Verifier [Dor, Rodeh, Sagiv, PLDI03, SAS01]

Specification and Verification of Java Card API:

• KeY Tool [Mostowski, Verify07]

• VerifiCard [Berg, Jacobs, Poll, TACAS01]
Conclusion

• We proposed a novel size abstraction and analysis for object-oriented models
• We conducted a case study on OCL specifications of Java Card API
• The experiments indicate our abstraction is precise enough to verify/falsify target systems, while coarse enough to perform efficient infinite state model checking
Thank you for your attention

Question?

- Fang Yu: yuf@cs.ucsb.edu
- Tevfik Bultan: bultan@cs.ucsb.edu
- VLab@UCSB http://www.cs.ucsb.edu/~bultan/~vlab