Formal Methods for Java

Lecture 17: Verification of Data Structures in Jahob

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

December 18, 2012
The Jahob system

Focus of Jahob: verifying properties of data structures.

Developed at
- EPFL, Lausanne, Switzerland (Viktor Kuncak)
- MIT, Cambridge, USA (Martin Rinard)
- Freiburg, Germany (Thomas Wies)

References
- Jahob webpage: http://lara.epfl.ch/w/jahob_system
- Viktor Kuncak’s PhD thesis
Core syntax of HOL

Jahob’s assertion language is a subset of the interactive theorem prover Isabelle/HOL which is built on the simply typed lambda calculus.

Terms and Formulas:

\[ f \ ::= \lambda x :: t. f \]  
lambda abstraction (\(\lambda\) is also written %)

\[ f_1 \, f_2 \]  
function application

\[ x \]  
variable or constant

\[ f :: t \]  
typed formula

Types:

\[ t ::= \text{bool} \]  
truth values

\[ \text{int} \]  
integers

\[ \text{obj} \]  
uninterpreted objects

\[ t_1 \Rightarrow t_2 \]  
total functions

\[ t \, \text{set} \]  
sets

\[ t_1 \times t_2 \]  
\textit{pairs}
A function with two arguments $g(x, y)$ has the type

$$g : (t_1 \times t_2) \Rightarrow t_3$$

In HOL, usually one defines a function with two arguments as

$$f : t_1 \Rightarrow t_2 \Rightarrow t_3,$$

and the application as

$$f \ x \ y = g(x, y)$$

Note that $\Rightarrow$ is right-associative and function application is left-associative:

$$(t_1 \Rightarrow t_2 \Rightarrow t_3) = (t_1 \Rightarrow (t_2 \Rightarrow t_3))$$ and $$f \ x \ y = (f \ x) y.$$
Lambda Abstraction

Suppose, you want to define a function or relation:

\[ inc \ x = x + 1 \quad \text{or} \quad succ \ x \ y \equiv (y = x + 1). \]

With lambda abstraction these can be written as

\[ inc \ = \ (\lambda \ x. \ x + 1) \quad \text{resp.} \quad succ \ = \ (\lambda \ x \ y. \ y = x + 1). \]

This is especially useful if you need a function argument:

\[ rtrancl\_pt \ succ \ 0 \ z \]

can be written as

\[ rtrancl\_pt \ (\lambda \ x \ y. \ y = x + 1) \ 0 \ z \]
Statically verify data structure consistency properties.

**Example**

**Internal Data Structure Consistency**

- field `prev` is inverse of field `next`
- field `next` is acyclic

→ inconsistency can cause program crashes.
External Consistency Properties

Example (Library)

- if a book is loaned to a person, then
  - the person is registered with the library, and
  - the book is in the catalog

- Can loan a book to at most one person at a time

- correlate multiple data structures
- depend on internal consistency
- capture design constraints (object models)

→ inconsistency can cause policy violations.
Goal

Proof data structure consistency properties

- for all program executions (sound)
- with high level of automation
- both internal and external consistency properties
- both implementation and use of data structures.
Overview of the Jahob Approach

- Reasoning about program in terms of simpler interfaces
  - uses of interfaces
  - global consistency

scalable analyses

Application (Data Structure Client)

A interface

B interface

A implementation

B implementation

Checking that interfaces reflect implementations and internal consistency is preserved - precise analyses
Overview of the Jahob Approach

Key question in automating approach (while keeping it useful)

How to choose interface language?

Application (Data Structure Client)

A interface

A implementation

B interface

B implementation
The Jahob Approach through an Example

Data structures to record who borrowed which book. These consist of:

- a set of persons, implemented by a linked list. Each person has a unique id.
- a set of books, implemented by a linked list. Each book has a unique id.
- a relation `borrows`, implemented by an array indexed by the person unique id. Array contains a linked list of books borrowed by that person.
class Library {
    public static Set persons;
    public static Set books;
    public static Relation borrows;
    ...
}

class Relation {
    private Set[] a;
    private int size;
    ...
    public void add(int i, Object o1){
        ...
    }
}

class Set {
    private Node first;
    ...
    public void add(Object o1){
        Node n = new Node();
        n.data = o1;
        n.next = first;
        first = n;
    }
}
if a person has borrowed a book, then

- the person is registered with the library, and
- the book is in the catalog

\[ \forall p \ b \ . \ (p, b) \in \text{borrows.content} \rightarrow \]
\[ p \in \text{persons.content} \land b \in \text{books.content} \]

**Specification Variables**

- \( \text{Set.content} = \{ x \mid \exists \ n \ . \ n \in \text{first.next}^* \land n.\text{data} = b \} \)
- \( \text{Relation.content} = \{ (x, y) \mid a[x] \neq \text{null} \land y \in a[x].\text{content} \} \)
Defining Interfaces using Specification Variables

class Node {
    Object data;
    Node next;
}
class Set {
    public Node first;
    /*: public specvar content :: objset;
    ...*

How can we define the set of data values in the linked list?

    content == first.next*.data

Jahob supports reflexive transitive closure but with a different syntax:

Definition (rtrancl_pt)

Let $R : \alpha \Rightarrow \alpha \Rightarrow \text{bool}$ be a relation on some type $\alpha$, then $\text{rtrancl}\_pt\; R$ is the reflexive transitive closure of $R$:

$r\text{trancl}\_pt\; R \times y$ holds if there is a sequence $x = x_0, \ldots, x_n = y$, $n \geq 0$ such that $R\; x_i\; x_{i+1}$ holds for $0 \leq i < n$. 
Using the rtrancl_pt predicate

Definition (rtrancl_pt)

Let $R : \alpha \Rightarrow \alpha \Rightarrow \text{bool}$ be a relation on some type $\alpha$, then $\text{rtrancl} \_ pt \ R$ is the reflexive transitive closure of $R$:

$r\text{trancl} \_ pt \ R \ x \ y$ holds if there is a sequence $x = x_0, \ldots, x_n = y$, $n \geq 0$ such that $R \ x_i \ x_{i+1}$ holds for $0 \leq i < n$.

Define the successor relation using the field Node.next:

$$R = (\% \ x \ y. \ x..\text{Node}.\text{next} = y)$$

Note: $\%$ is $\lambda$-abstraction.

The set of all nodes on the list is:

$$\text{nodes} = \{n. \ \text{rtrancl} \_ pt (\% \ x \ y. \ x..\text{Node}.\text{next} = y) \ \text{first} \ n\}$$

and the set of all values on the list is:

$$\text{contents} = \{x. \ \text{EX} \ n. \ n..\text{Node}.\text{data} = x$$

& $\text{rtrancl} \_ pt (\% \ v1 \ v2. \ v1..\text{Node}.\text{next} = v2) \ \text{first} \ n\}$$
class Set {
    private Node first;
    ...
    /*: public specvar content :: objset;
    vardefs "content == {x. EX n. n..Node.data = x &
    rtrancl_pt (% v1 v2. v1..Node.next = v2) first n}";
    ...
    invariant "tree [Node.next]";
    */
    public void add(Object o1)
        /*: requires "o1 ~: content"
        modifies "content"
        ensures "content = old content Un {o1}"*/
        { ... }
}
class Library {
    public static Set persons;
    ...
    /*: invariant "ALL p b. (p,b) : borrows..Relation.content -->
       p : persons..Set.content & b : books..Set.content" */

    public static void checkOutBook(Person p, Book b)
    /*:
       requires "p ~= null & b ~= null &
         b : books..Set.content & p : persons..Set.content"
       modifies "borrows..Relation.content"
       ensures "((ALL p1. (p1,b) ~: old borrows..Relation.content) -->
         borrows..Relation.content =
         old (borrows..Relation.content) Un {(p,b)})
       & (EX p1. (p1,b) : old borrows..Relation.content -->
         borrows..Relation.content = old borrows..Relation.content)"
    */
    { ... }
}