Formal Methods for Java

Lecture 20: Jahob Syntax

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

Jan 13, 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

Jahob Specification

Jahob and JML have similar keywords:

JML	Jahob
ensures	ensures
modifies	modifies
requires	requires
assert	assert
assume	assume
invariant	invariant
loop_invariant	invariant
models	specvar
represents	vardefs
ghost	ghost specvar
assert; assume	noteThat

The position of the specification slightly differs.

Example: Jahob Specification

```
public class ArrayList
{
    private Object[] elementData;
    private int size;

    /*: public specvar init :: bool;
        public specvar content :: "(int * obj) set";
        ...
        vardefs "init == elementData ~= null"
        vardefs "content == {(i, n). 0 <= i & i < size & n = elementData.[i
        ...
        invariant InitInv:
          "~init --> size = 0 & elementData = null"
        ...
    */
```

Example: Jahob Specification (cont.)

```
    private boolean add(Object o1)
    /*: requires "init & theinvs"
        modifies "Array.arrayState", elementData, msize, "ArrayList.hidden",
        ensures "..."
    */
    {
        elementData[size] = o1;
        size = size + 1;
        //: noteThat "old csize < size";
        return true;
    }
}
```

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 (λ is also written %)
	$\mid$	$f_1\ f_2$	function application
	$\mid$	x	variable or constant
	$\mid$	$f :: t$	typed formula
		Types:	
t	$::=$	bool	truth values
	$\mid$	int	integers
	$\mid$	obj	uninterpreted objects
	$\mid$	$t_1 \Rightarrow t_2$	total functions
	$\mid$	t set	sets
	$\mid$	$t_1 * t_2$	*pairs*

Predefined constants in HOL

Core syntax is enriched with predefined constants:

- Boolean connectives: `~ F, F & G, F | G, F --> G, F <-> G`
- (dis)equality: `f = g, f ~= g`
- sets and set operations:
  `{f_1, ..., f_n}, {x. F}, f : S, S Un T, S Inter T, S - T`
- quantification: `ALL x. F, EX x. F`
- reflexive transitive closure of predicates: `rtrancl_pt P a b`
- the null object: `null`
- ...

Java specific HOL-syntax

Field access is handled by functions, alternatively .. can be used:

```
/*:
  requires "Node.next obj ~= null"
  or requires "obj..Node.next ~=null"
  */
```

Array access also requires an additional dot:

```
/*:
  requires "arr.[k] ~= null"
  */
```

Example HOL-formulas

On input, array *arr* is not null and contains non-null elements.

```
/*:
  requires "arr ~= null &
    (ALL k. (0 <= k & k < (Array.length arr)) --> arr.[k] ~= null)"
  */
```

On output, array *arr* is sorted.

```
/*:
  ensures
    "(ALL k. ((0 <= k & k < ((Array.length arr) - 1)) -->
    (arr.[k]..InsertionSortNode.key
                 <= arr.[k+1]..InsertionSortNode.key)))"
  */
```

Demo: Insertion Sort