Modular specification of frame properties in JML

(or : What semantics for the modifies in JML ?)

P. Müller, A. Poetzsch-Heffter and G.T. Leavens

Julien Charles

Friday, november the 3rd 2006

Disclaimer

- There is informal in informal article meetings
- Not as long as my last article meeting,
  I promise - only 10 slides (vs. 20 last time)
- How did I came accross it ?
  → Semantics of modifies for void-returning specification-methods
- The only article with some kind of definition for the modifies

General Plan

1. JML problems
2. Modularity problems, solutions and restrictions

JML problems

1. Reminder
2. A naive definition
3. Problems that matter (1) : Information hiding
4. Problems that matter (2) : modification of concrete location and extended state

JML reminder

JML stands for Java modelling Language
(a well-known fact)
You can express all the things you would dream of especially method contracts :

- precondition
- postcondition
- what it modifies

Example :

```
/*@ modifies first ;
  @ ensures first == null ; @*/
public void initializeFirst() {
   first = null ;
}
```

A naive pathway to the sun

"Only the locations listed in a method's modifies clause may be changed by execution of that method (when its precondition is satisfied). No other locations may be modified by such an execution."

3 problems :

- The information hiding problem
- The modification of Concrete locations
- The extended state problem

Information hiding

POO precept : *"Thou shalt not see specification over private fields, etc..."*

$\longrightarrow$ Solution : abstract field modifications

```
//@ public model Seq listValue ;
//@ represents listValue  ← (first == null ? new Seq() : first.values) ;
/*@ modifies listValue ;
  @ ensures listValue.isEmpty() ; @*/
public void initializeFirst() {
   first = null ;
   //@ assert listValue.isEmpty() ;
}
```

Concrete locations and extended states

The previous solution conflicts with the naive semantics

$\longrightarrow$ Solution : explicit dependencies clause

//@ depends listValue $\leftarrow$ first ;

yes ! ! a new definition for modifies :

"Only the locations listed in a method's modifies clause and their dependees may be changed by execution of that method (when its precondition is satisfied). No other locations may be modified by such an execution."

The modularity problem

1. The problem
2. Context (aka. Ownership)
3. New semantics

Modularity is thine Enemy !

Problem : the depends can change the modifies of already annotated proven and defined class

Solution : scope dependent modifies clause [Leino and Nelson 2000]
$\longrightarrow$ Dubious soundness : smaller scopes properties vs. larger scopes properties

New (better) solution proposed here :

1. Contexts
2. Relevant locations
3. (and under-specification)

Context(s)

Ownership model invariant :

“Every reference chain from objects in the root context to an object in a context C different from the root context passes through C’s owner. For purpose of this invariant, local variables and method parameters are treated like locations of the corresponding this-object.”

$\longrightarrow$ If an object is not in the context it is considered as read-only

More semantics

Relevant locations definition :

"A location L is relevant to the execution of a method m with receiver object X iff L is either in the context C containing X or in a descendant context of C"

New modifies semantics :

"Of the relevant locations for a call to a method, only those locations listed in the method's modifies clause and their dependees may be changed by execution of that method (when its precondition is satisfied). No other relevant locations may be modified by such a call."

Conclusion

What we have not seen :

- the method is claimed to be sound
- new JML keywords ! (rep and read-only)
- underspecified depends relations

Vs. the normal semantic

$\longrightarrow$ set is harder to specify because context dependent
(that's the discussion on the next slide)

Discussion

Q : How to implement JML's set with
specification methods ?

A : Void specification-methods $+$
modifies semantics !