Jack - haRVey - Jack

Lilian Burdy

INRIA Sophia-Antipolis
Lilian.Burdy@sophia.inria.fr

Overview

- Introduction
- Jack architecture
- The haRVey plug-in
- Future works

Introduction 1/2

A crucial point in validation tools is the proof facilities.
The proof language/engine/interface should:

- have a good expression power (high order logic, ...) in order to ease the formalization;
- have a good automatic proof ratio in order to minimize proof effort;
- have an usefull (simple, ...) interactive interface in order to minimize proof effort;
- help user to find rapidly (easily, ...) errors through unvalid lemmas.

Introduction 2/2

Assumption: All those requirements cannot be covered by a uniq prover.

Different provers have to be interfaced.

But, it would be usefull to keep a same interface for users.

Jack architecture

Jack eclipse plugin:

- Jack implements a weakest precondition calculus from Java/JML to an “internal language”.
- Lemmas are displayed to user translated into a language close to JML.
- It can be extended throught interfaces that allow to define:
  - a language;
  - how to display a lemma in this language;
  - how to generate an output file, run an executable on this file and update proof status of lemmas.

Prover plugins

At the moment Jack plugin extension exist for:

- B
- Simplify
- haRVey (not finalized)

Coq plugin is under construction.

PVS plugin is planned.

Jack - haRVey - Example 1/7

```
public class D2 {

   Object a;

   //@ ensures \typeof(a) <: \type(D2);
      void m() {
         a = new D2();
      }
}
```

Jack - haRVey - Example 2/7

```
(forall A I E (= (select (store A I E) I) E))
(forall A I J E
 (-> (not (= I J))
     (= (select (store A I E) J)
        (select A J))))

(not (= c_Object c_Exception))
(not (= c_Object c_Throwable))
(not (= c_Object c_String))
(not (= c_Object c_RuntimeException))
(not (= c_Object c_StringBuffer))
(not (= c_Object c_NullPointerException))
...
(not (= c_Comparator c_Serializable))
```

Jack - haRVey - Example 3/7

```
(forall x
 (<->
  (subtype x (ref c_Exception))
  (or
   (= x (ref c_Exception))
   (= x (ref c_RuntimeException))
   (= x (ref c_NullPointerException))
   (= x (ref c_ArithmeticException))
   (= x (ref c_ArrayIndexOutOfBoundsException))
   (= x (ref c_IndexOutOfBoundsException))
   (= x (ref c_NegativeArraySizeException))
   (= x (ref c_ClassCastException))
   (= x (ref c_ArrayStoreException))
   (= x (ref c_UnsupportedEncodingException))
   (= x (ref c_IOException)))))
```

Jack - haRVey - Example 4/7

```
(forall x y (<-> (= x y) (= (ref x) (ref y))))

(not (= tt ff))

(forall c
 (-> (= (instances c) tt)
     (REFERENCES c)))

(not (= (instances null) tt))
```

Jack - haRVey - Example 5/7

```
; specific to the class
(forall x
 (-> (= (instances x) tt)
     (-> (subtype (select typeof x) (ref c_D2))
         (-> (not (= (select f_a x) null))
             (= (instances (select f_a x)) tt))

(forall x
 (-> (= (instances x) tt)
     (-> (subtype (select typeof x) (ref c_D2))
         (-> (not (= (select f_a x) null))
             (subtype (select typeof
                                  (select f_a x))
                          (ref c_Object))))))
```

Jack - haRVey - Example 6/7

```
; specific to the lemma
(not (= (instances newObject_1) tt))

(not (= newObject_1 null))

(-> (not (= at_6 newObject_1))
 (= (select f_a at_6) (select f_D2_a_1 at_6)))

(= (instances this) tt)

(subtype (select typeof this) (ref c_D2))
;--End of hypothesis--
```

Jack - haRVey - Example 7/7

```
(subtype
   (select
     (store typeof newObject_1(ref c_D2))
     (select
       (store f_D2_a_1 this newObject_1)
       this))
   (ref c_D2))

; ensures (subtype
;            (select typeof (select f_a this))
;            (ref c_D2))
```

haRVey - Jack - Future works

haRVey returns the set of clauses that are not satisfiable when it fails to prove a lemma.

It would be usefull to translate back this information to users in a Java syntax.

It can help to find easily, if the lemma is valid or not, and if it is not valid where is the error (code or specification).

Conclusion

The last development of Jack makes it a plugin that allows extension.

Provers are no more in the core of the tool.

New plugin can be developped by anybody whishing to prove with its proof system.

haRVey can be helpfull if the users can beneficiate from the information that it returns.