

Prover plugins

B

Simplify

Coq

harRVey

PVS

Plugin Prover: overview

- JPOL
 - Java/JML construction
 - Background theory
 - Field update
 - Assignable clause
- Provers
 - Obvious
 - Automatic
 - Interactive interface

JPOL

- Java Proof Obligation Language
 - 70 constructors/terminals
 - Mix between
 - Java (27)
 - Jml (9)
 - Ad hoc construction (20)
 - Set Theory (due to B historical prover) (14)
 - Generated by the WP calculus
 - Front end for the provers plugin

JPOL

- Java structure
 - Arithmetic: +, *, -, /, %
 - Comparaison: ==, <=, >=, <, >, !=
 - Logical: !, &&, ||
 - Question: a ? b : c
 - Object *constants*: this, null
 - Constants: true, false, string litteral, numeric litteral

JPOL

- JML construction:
 - `\old`, `\result`
 - `\typeof`, `\elemtype`, `\TYPE`, `\type`, `<:`
 - `==>`, `\forall`, `\exists`

JPOL

- Ad hoc construction
 - IS_ARRAY, IS_MEMBER_FIELD, IS_STATIC_FIELD
 - EQUALS_ON_OLD_INSTANCE, ARRAY_RANGE
 - LOCAL_VAR_DECL, NEW_OBJECT
 - ARRAY_ACCESS, ...

JPOL

- Set theory
 - singleton, union, belonging
 - integer interval
 - function application, function overriding
 - ...

Background theory

- Integer types: t_int, t_short, ...
 - Conversion fonctions : j_int2char, ...
- References type : REFERENCES
 - Particular reference : null
 - Allocated references : instances
 - Property : null does not belongs to instances

Background theory

- Set of all classes
 - Subtype relation between classes
- Reference type : class or array
 - Typeof relation between instances and reference type
- Array elements

Static fields

- Static built-in type field: `static int sf`
is converted in `t_int f_sf`
- Static reference field: `static MyClass smc`
is converted in
 - `REFERENCES f_smc`
 - `f_smc != null`
`==> typeof(f_smc) <: type(c_MyClass)`

Member fields

- `Class MyClass {int i; YourClass yc; ...}`

```
f_i: {inst: instances |  
      typeof(inst) <: type(c_MyClass)}  
      --> t_int
```

```
f_yc: {inst: instances |  
       typeof(inst) <: type(c_MyClass)}  
       --> REFERENCES
```

```
\forallall instances inst;  
  typeof(inst) <: type(c_MyClass);  
  typeof(f_yc(inst)) <: type(c_YourClass)
```

Field Update

- Member field affectation: `a.i = 3`
 - `f_i [WITH a := 3]`
 - `f_i <+ {a |-> 3}`
 - `let x in if x = a then 3 else (f_i x).`
 - `(store f_i a 3)`

Assignable clauses 1/2

- `\everything`

`true`

- `\nothing`

```
\forall r : \old(instances) r;  
  typeof(r) <: type(C);  
  f(r) = \old(f(r))
```

- `a.i`

```
\forall r : \old(instances) r;  
  typeof(r) <: type(C);  
  r != a ==> f(r) = \old(f(r))
```

Assignable clauses 2/2

- **a.b[c..d]**

```
\forall \old(instances) r;  
  typeof(r) <: type(C);  
  r != a  
  ==> elements(r) == \old(elements(r))  
\forall t_int i;  
  0 <= i < c or d < i < length(b);  
  elements(a)(i)  
  == \old(elements(a)(i))
```

Provers

- Jack contains an minimal prover
 - $A == A$
 - A and $!A$ in hypothesis
 - $A == B$ and $A != B$ in hypothesis
- Allow to avoid some combinatory explosion
 - Incompatible branch are cut before being created

Obvious provers

- Plugins can implement obvious provers
 - Must be efficient
 - Call after lemma generation before saving
 - Must be transparent to user
- Simplify can be used as obvious prover, others are too slow

Automatic prover

- Call by the user on a Java file
 - Prove lemma one by one
 - Inform user for progression
 - Store proof status in JPO file
 - Proved or not
 - Potentially proof tactics or scripts
- Coq, B, Simplify
- PVS and haRVey are other candidates

Interactive prover

- Call on one lemma in the lemma viewer
- Launch an other ide
 - CoqIDE, proofgeneral, Pcoq, ... for Coq
 - emacs for PVS
 - Click'n Prove for B
- Perspective : proofgeneral as an eclipse plugin !