### Formal Methods for Java Lecture 19: Proofs in Jahob

#### Jochen Hoenicke



Software Engineering Albert-Ludwigs-University Freiburg

Jan 8, 2013

## Static Checking vs. Theorem Proving

Goal:

- finds bugs at compile-time,
- proves that there is no violation.

Static Checking:

- e.g. Jahob and ESC/Java
- fully automatic (after annotation)
- can only verify simple properties

Theorem Proving:

- Needs lot of manual interaction
- complete calculus, can verify any property.

#### Goals

- Improve the strength of the provable properties.
- Still fully automatic (after annotation).
- Have intermediate proof steps in annotation.

#### Paper:

• Karen Zee, Viktor Kuncak, and Martin Rinard. An integrated proof language for imperative programs. In ACM Conf. Programming Language Design and Implementation (PLDI), 2009.

#### Note command

We already know one command

note  $\ell$  : *F* 

which abbreviates

```
assert \ell : F; assume \ell : F
```

- $\ell$  is a label (or name) for the formula F
- When F cannot be proven Jahob tells that the check for  $\ell$  failed.
- $\ell$  can also be used to tell the Jahob which formulas are relevant:

assert  ${\it G}$  from  $\ell$ 

Why is this rule correct?

### Correctness of Proof Commands

An incorrect program must not be verified succesfully.

If  $P \rightarrow wp(S_1; \text{note } F; S_2, Q)$ then  $P \rightarrow wp(S_1; S_2, Q)$ 

This is the case if we can proof that for all H

 $wp(note F, H) \rightarrow H$ 

The note *F* command is correct:

```
wp(\text{note } F, H) \leftrightarrow wp(\text{assert } F; \text{assume } F, H)\leftrightarrow F \land (F \to H)\leftrightarrow F \land H\rightarrow H
```

### Proving implications

Suppose you want to argue that F implies G by a implication chain

```
F \to F_1 \to F_2 \to G.
```

In Jahob there is a special syntax:

assuming F in ( note  $F_1$ note  $F_2$ note G)

This command adds the assumption

assume  $F \to G$ 

# General syntax of assuming

The general syntax is

```
assuming F in
( :
note G)
```

This is an abbreviation for

```
( assume F

:

assert G

assume false

assume F \rightarrow G

)
```

#### • : stands for arbitrary proof statements

The implication rule is correct, provided the proof statements used in between are correct.

$$\begin{split} & \textit{wp}((\text{assume } F; p; \text{assert } G; \text{assume false} \square \text{ assume } F \to G, H) \\ & \equiv (F \to \textit{wp}(p, G)) \land ((F \to G) \to H) \\ & \to [\text{assuming that proof statements } p \text{ are correct}] \\ & (F \to G) \land ((F \to G) \to H) \\ & \to H \end{split}$$

۱

### **Case Splits**

One can split cases, e.g.

cases  $x \ge 0, x < 0$  for  $abs(x) \ge 0$ 

cases  $F_1, \ldots, F_n$  for G

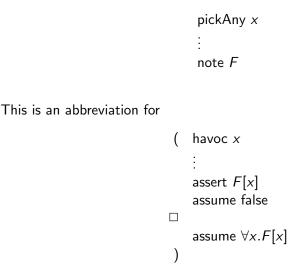
is an abbreviation for

assert  $F_1 \lor \cdots \lor F_n$ ; assert  $F_1 \to G$ ; ... assert  $F_n \to G$ ; assume G

- Proof that  $F_1, \ldots, F_n$  are all possible cases.
- Proof for each case G separately.
- Assume G holds.

### Proving Universal Quantifiers

To prove a universal quantified formula the syntax is



The inverse operation removes universal quantifiers:

```
instantiate \forall x.F[x] with t
```

This is an abbreviation for

assert  $\forall x.F[x]$ assume F[t]

#### To prove an existential quantified formula the syntax is

witness t for  $\exists x.F[x]$ 

This is an abbreviation for

assert F[t]assume  $\exists x.F[x]$ 

# Removing Existential Quantifiers

The syntax is

```
pickWitness x for F[x]

: where x does not occur in G

note G
```

This is an abbreviation for

```
( assert ∃x.F[x]
    havoc x
    assume F[x]
    :
    assert G
    assume false
    assume G
)
```