Java+ITP: A Verification Tool Based on Hoare Logic and Algebraic Semantics

Ralf Sasse, José Meseguer

Department of Computer Science
University of Illinois at Urbana-Champaign

6th International Workshop on Rewriting Logic and its Applications, 2006

Outline

1. Motivation
   - Long-Term Goals
   - General Idea
   - Previous Work
2. Our Work
   - Preliminaries
   - Main Results
   - An Example

Long-Term Goals

- Investigate modularity and extensibility of
    - programming languages
    - Hoare logics
- Source-code level reasoning.
- Generic and modular program logics wanted.
- Develop theorem proving technology on top of the logics.
- This case study is a first step in this direction.

Problem Considered

- Sequential Java subset:
  - arithmetic expressions, assignments, sequential composition, loops
- Hoare logic for this programming language with side-effects, mathematically justified.
- Standard Hoare rules break down!
- Mechanization available, supporting:
  - Compositional reasoning to decompose Hoare triples
  - Generation of first-order verification conditions
  - Discharging verification conditions with Maude's inductive theorem prover (ITP)

ASIP+ITP

- We have benefitted from previous experience at UCM on ASIP+ITP, a tool that works on Hoare logics for a simple toy language.
- However, in ASIP+ITP:
  - no side-effects in conditions
  - no variable declarations and shadowing
  - simple state infrastructure: direct mapping of variables to values
  - no compositional decomposition of Hoare triples supported

Java Semantics

- Algebraic semantics (only equations used)
- Continuation passing style approach
- Extracted from a larger rewriting logic semantics for Java (JavaFAN)

Hoare Triples

- Consist of:
  - pre condition `A`
  - post condition `B`
  - program `p`
- Written: `{A} p {B}`
- Implicit state `S` used, made explicit when necessary.
- Meaning: $\mathtt{A(S)} \Rightarrow \mathtt{B(S/(S|p))}$

Our Hoare Rules

- Standard sequential composition rule:

$$\frac{\{A\}\ \mathtt{p}\ \{B\} \qquad \{B\}\ \mathtt{q}\ \{C\}}{\{A\}\ \mathtt{p\ \ q}\ \{C\}}$$

- Standard conditional rule, breaks down here:

$$\frac{\{A \wedge \mathtt{t}\}\ \mathtt{p}\ \{B\} \quad \{A \wedge \neg\mathtt{t}\}\ \mathtt{q}\ \{B\}}{\{A\}\ \mathtt{if\ t\ p\ else\ q}\ \{B\}}$$

- Correct conditional rule, for side-effects in condition:

$$\frac{\{A \wedge \mathtt{t}\}\ \mathtt{t\ \ ;\ \ p}\ \{B\} \quad \{A \wedge \neg\mathtt{t}\}\ \mathtt{t\ \ ;\ \ q}\ \{B\}}{\{A\}\ \mathtt{if\ t\ p\ else\ q}\ \{B\}}$$

Our Hoare Rules

- Standard sequential composition rule:

$$\frac{\{A\}\ \mathtt{p}\ \{B\} \qquad \{B\}\ \mathtt{q}\ \{C\}}{\{A\}\ \mathtt{p\ q}\ \{C\}}$$

- Standard conditional rule, breaks down here:

$$\frac{\{A \wedge \mathtt{t}\}\ \mathtt{p}\ \{B\} \quad \{A \wedge \neg\mathtt{t}\}\ \mathtt{q}\ \{B\}}{\{A\}\ \mathtt{if\ t\ p\ else\ q}\ \{B\}}$$

- Correct conditional rule, for side-effects in condition:

$$\frac{\{A \wedge \mathtt{t}\}\ \mathtt{t} \mid \mathtt{p}\ \{B\} \quad \{A \wedge \neg\mathtt{t}\}\ \mathtt{t} \mid \mathtt{q}\ \{B\}}{\{A\}\ \mathtt{if\ t\ p\ else\ q}\ \{B\}}$$

Our Hoare Rules

- Standard sequential composition rule:

$$\frac{\{A\}\,\texttt{p}\,\{B\} \qquad \{B\}\,\texttt{q}\,\{C\}}{\{A\}\ \texttt{p q}\ \{C\}}$$

- Standard conditional rule, breaks down here:

$$\frac{\{A \wedge \texttt{t}\}\,\texttt{p}\,\{B\} \quad \{A \wedge \neg\texttt{t}\}\,\texttt{q}\,\{B\}}{\{A\}\ \texttt{if t p else q}\ \{B\}}$$

- Correct conditional rule, for side-effects in condition:

$$\frac{\{A \wedge \texttt{t}\}\,\texttt{t | p}\,\{B\} \quad \{A \wedge \neg\texttt{t}\}\,\texttt{t | q}\,\{B\}}{\{A\}\ \texttt{if t p else q}\ \{B\}}$$

Our Hoare Rules

- Standard while loop rule, breaks down here:

$$\frac{\{A \wedge \mathtt{t}\}\ \mathtt{p}\ \{A\}}{\{A\}\ \mathtt{while\ t\ p}\ \{A \wedge \neg\mathtt{t}\}}$$

- Correct while loop rule, for side-effects in condition:

$$\frac{\{A \wedge \mathtt{t}\}\ \mathtt{t}\ \mid\ \mathtt{p}\ \{A\} \quad \{A \wedge \neg\mathtt{t}\}\ \mathtt{t}\ \{A \wedge \neg\mathtt{t}\}}{\{A\}\ \mathtt{while\ t\ p}\ \{A \wedge \neg\mathtt{t}\}}$$

Our Hoare Rules

- Standard while loop rule, breaks down here:

$$\frac{\{A \wedge \mathtt{t}\}\ \mathtt{p}\ \{A\}}{\{A\}\ \mathtt{while\ t\ p}\ \{A \wedge \neg\mathtt{t}\}}$$

- Correct while loop rule, for side-effects in condition:

$$\frac{\{A \wedge \mathtt{t}\}\ \mathtt{t}\ \mathtt{|}\ \mathtt{p}\ \{A\} \quad \{A \wedge \neg\mathtt{t}\}\ \mathtt{t}\ \{A \wedge \neg\mathtt{t}\}}{\{A\}\ \mathtt{while\ t\ p}\ \{A \wedge \neg\mathtt{t}\}}$$

Java+ITP tool: javax-inv

- The `javax-inv` command applies multiple rules and creates first-order proof obligations.
- Target Hoare triple is `{P} init loop {Q}`.

$$\frac{\{P\}\ \texttt{init}\ \{I\} \qquad \dfrac{I \Rightarrow I \quad \{I\}\ \texttt{loop}\ \{I \wedge \neg \texttt{t}\} \quad (I \wedge \neg \texttt{t}) \Rightarrow Q}{\{I\}\ \texttt{loop}\ \{Q\}}}{\{P\}\ \texttt{init}\ \texttt{loop}\ \{Q\}}$$

- $\texttt{loop} = \texttt{while t p}$:

$$\frac{\{I \wedge \texttt{t}\}\ \texttt{t}\ |\ \texttt{p}\ \{I\} \quad \{I \wedge \neg \texttt{t}\}\ \texttt{t}\ \{I \wedge \neg \texttt{t}\}}{\{I\}\ \texttt{while t p}\ \{I \wedge \neg \texttt{t}\}}$$

Java+ITP tool: javax-inv

- The `javax-inv` command applies multiple rules and creates first-order proof obligations.
- Target Hoare triple is `{P} init loop {Q}`.

$$\frac{\{P\}\ \mathtt{init}\ \{I\} \qquad \dfrac{I \Rightarrow I \quad \{I\}\ \mathtt{loop}\ \{I \wedge \neg \mathtt{t}\} \quad (I \wedge \neg \mathtt{t}) \Rightarrow Q}{\{I\}\ \mathtt{loop}\ \{Q\}}}{\{P\}\ \mathtt{init}\ \mathtt{loop}\ \{Q\}}$$

- $\mathtt{loop} = \mathtt{while\ t\ p}$:

$$\frac{\{I \wedge \mathtt{t}\}\ \mathtt{t}\ \mid\ \mathtt{p}\ \{I\} \quad \{I \wedge \neg \mathtt{t}\}\ \mathtt{t}\ \{I \wedge \neg \mathtt{t}\}}{\{I\}\ \mathtt{while\ t\ p}\ \{I \wedge \neg \mathtt{t}\}}$$

Java+ITP tool: Compositionality

- Hoare triples can be added as proof goals without translating them to first-order goals right away.
- Java+ITP supports decomposition of such Hoare triples.
- Simpler Hoare triples can be translated to first-order goals on user command.

Example Program

Binomial coefficient program for $\binom{n}{k}$, with inputs `'N` and `'K`.

```
int 'N ; int 'Nfac ; int 'K ; int 'Kfac ;
int 'N-Kfac ; int 'BC ; int 'I ;

'I = #i(0) ; 'Nfac = #i(1) ;
while ('I < 'N)
  { 'I = 'I + #i(1) ; 'Nfac = 'Nfac * 'I ; }
'I = #i(0) ; 'Kfac = #i(1) ;
while ('I < 'K)
  { 'I = 'I + #i(1) ; 'Kfac = 'Kfac * 'I ; }
'I = #i(0) ; 'N-Kfac = #i(1) ;
while ('I < ('N - 'K))
  { 'I = 'I + #i(1) ; 'N-Kfac = 'N-Kfac * 'I ; }
'BC = 'Nfac / ('Kfac * 'N-Kfac) ;
```

Verification Property

Property to be verified, with `S` the state and `N` and `K` the inputs:

$$\{\texttt{S['N]} = \texttt{N} \wedge \texttt{S['K]} = \texttt{K}$$

$$\wedge\ \texttt{0 <= N} \wedge \texttt{0 <= K} \wedge \texttt{0 <= N - K}\}$$

binomial-coefficient-program

$$\{\texttt{S['Nfac]} = \texttt{N!}$$

$$\wedge \texttt{S['Kfac]} = \texttt{K!}$$

$$\wedge \texttt{S['N-Kfac]} = \texttt{(N - K)!}$$

$$\wedge \texttt{S['BC]} = \texttt{bc(N, K)}\}$$

Example Roadmap

- Load all necessary parts into Maude:
  - our modified ITP
  - our Java semantics
  - the module defining the binomial coefficient code
- Enter the property we want to show, as seen above.
- Decompose the proof obligation.
- Transform proof obligations to first-order proof obligations, done for one proof obligation here.
- Use ITP to discharge the first-order proof obligations.

Enter Proof Obligation

```
select ITP-TOOL .
loop init-itp .

(add-hoare-triple CHOOSE-JAVAX :
--- specification variables
(N:Int ; K:Int)
--- precondition
 ((int-val(S:WrappedState['N])) = (N:Int)
& (int-val(S:WrappedState['K])) = (K:Int)
& (0 <= N:Int) = (true)
& (0 <= K:Int) = (true)
& (0 <= N:Int - K:Int) = (true))
--- program
choose
```

Enter Proof Obligation (II)

```
--- postcondition
 ((int-val(S:WrappedState['Nfac]))
   = ((N:Int)!)
& (int-val(S:WrappedState['Kfac]))
   = ((K:Int)!)
& (int-val(S:WrappedState['N-Kfac]))
   = ((N:Int - K:Int)!)
& (int-val(S:WrappedState['BC]))
   = (choose(N:Int, K:Int)))
--- initialization code
choose-init
.)
```

Decompose Proof Obligation

```
(decomp:
---- name of the HT
choose@0
---- prefix code
facN
---- midcondition
 ((int-val(S:WrappedState['N])) = (N:Int)
& (int-val(S:WrappedState['K])) = (K:Int)
& (0 <= N:Int) = (true)
& (0 <= K:Int) = (true)
& (0 <= N:Int - K:Int) = (true)
& (int-val(S:WrappedState['Nfac]))
    = ((N:Int)!))
---- rest of code
choose/facN .)
```

First Decomposed Proof Obligation

```
==================================
hoare-label: choose@1.0
==================================
{ (int-val(S:WrappedState['N])= N:Int)
 &(int-val(S:WrappedState['K])= K:Int))
 &(0 <= N:Int - K:Int = true))
 &(0 <= N:Int = true)) &(0 <= K:Int = true)}
facN
{ (int-val(S:WrappedState['Nfac])= N:Int !)
 &(int-val(S:WrappedState['N])= N:Int)
 &(int-val(S:WrappedState['K])= K:Int)
 &(0 <= N:Int - K:Int = true)
 &(0 <= N:Int = true) &(0 <= K:Int = true)}
```

Create First-Order Proof Obligation for `choose@1.0`

```
(create-FO-goal-hoare-inv:
---- name of the HT
choose@1.0
--- invariant
((int-val(S:WrappedState['Nfac]))
= ((int-val(S:WrappedState['I]))!)
& (0 <= int-val(S:WrappedState['I]))= (true)
& (int-val(S:WrappedState['I])
   <= int-val(S:WrappedState['N]))= (true)
& (int-val(S:WrappedState['N]))= (N:Int)
& (int-val(S:WrappedState['K]))= (K:Int)
& (0 <= N:Int) = (true)
& (0 <= K:Int) = (true)
& (0 <= N:Int - K:Int) = (true) ) .)
```

Summary

- Proof of concept for Hoare logics for a non-trivial language.
- Compositionality on the source-code level.
- Useful in teaching about algebraic semantics and Hoare logics.

- Outlook
  - Extend this to a larger Java fragment.
  - Generalize it, independent of the actual language, each rule only dependent on a single language feature.