Formal Methods: Lecture 16

José Meseguer

Computer Science Department
University of Illinois at Urbana-Champaign

Justification of the Loop Rule

We have justified other rules in Hoare's logic, but we still need a justification for the soundness of the loop rule,

$$\frac{\{A \wedge (\texttt{evalTst(S,t)} = \texttt{true})\}\ \texttt{t}\ \texttt{|}\ \texttt{p}\ \{A\} \quad \{A \wedge (\texttt{evalTst(S,t)} = \texttt{false})\}\ \texttt{t}\ \{A \wedge (\texttt{evalTst(S,t)} = \texttt{false})\}}{\{A\}\ \ \texttt{while t p}\ \ \{A \wedge (\texttt{evalTst(S,t)} = \texttt{false})\}}$$

Observations and Lemmas

We approach the justification of the loop rule by means of some observations and auxiliary lemmas that will place us in a good position to prove its soundness.

First Observation: All terms of sort `WrappedState` have a canonical form by the equations.

Second Observation: Using the assumption that the semantic rules of `JAVAX` are ground confluent, for any ground terms s of sort `WrappedState` and p of sort `BlockStatements` we have,

$$E \vdash (\forall \emptyset)\ \texttt{s | p} : \texttt{WrappedState} \quad \Leftrightarrow \quad (\exists! s' \in T_{\texttt{JAVAX},\texttt{WrappedState}})\ \texttt{s | p} \xrightarrow{*}_E s',$$

where E are the equations of `JAVAX`.

Observations and Lemmas (II)

Lemma: For any ground terms s_0 of sort `WrappedState`, t of sort `Exp` (and assuming it will evaluate to a boolean value), and p of sort `BlockStatements`, if we have,

$$E \vdash (\forall \emptyset)\ s_0 \mid \texttt{while}\ t\ p\ :\ \texttt{WrappedState}$$

then any rewriting sequence, $s_0 \mid \texttt{while}\ t\ p \longrightarrow_E^* s'$ with $s' \in T_{\texttt{JAVAX},\texttt{WrappedState}}$ must be of the form,

$$s_0 \mid \texttt{while}\ t\ p \longrightarrow_E^+ s_1\ ,\ \texttt{k(while}\ t\ p\ \texttt{->}\ \texttt{stop)} \longrightarrow_E^+ \ldots \longrightarrow_E^+ s_n\ ,\ \texttt{k(while}\ t\ p\ \texttt{->}\ \texttt{stop)} \longrightarrow_E^+ s'$$

with $n \geq 0$, with s_i of sort `WrappedState`, $0 \leq i \leq n$, with $s_n \mid \texttt{t} \longrightarrow_E^+ s'$ and $E \vdash (\forall \emptyset)\ \texttt{evalTst}(s_n, t) = false$. In general

we cannot guarantee that $E \vdash (\forall \emptyset)\ \texttt{evalTst}(s', t) = false$ holds, but in the case we look at later, it will hold, as detailed there. Also, for $0 \leq i < n$,

- s_i , $\texttt{k(}t\texttt{ -> }p\texttt{ -> stop)} \xrightarrow{*}_E s_{i+1}$
- $E \vdash (\forall \emptyset)\ \texttt{evalTst}(s_i, t) = true$.

Observations and Lemmas (III)

Proof: By induction on the number n of different occurrences of expressions of the form s_i , `k(while` t p `->` `stop)`, with s_i of sort `WrappedState`, $0 \leq i \leq n$, that appear in the sequence s_0 | `while` t $p \longrightarrow_E^* s'$. q.e.d.

Notation: given a program p, we use the notation p^n, working also for programs composed of an `Exp` t and `BlockStatements` p', $p = t\,|\,p'$, to mean:

- $p^0 = ;$
- $p^{n+1} = p\,|\,p^n$

We use composition with _|_ instead of composition with __ because the internal types in p need not match,

i.e. $(t \,|\, p')^2 = (t \,|\, p' \,|\, t \,|\, p')$ and not $(t \,|\, p'\ t \,|\, p')$ where $p'\ t$ would be illegal.

Corollary: For any ground terms s_0 of sort `WrappedState`, t of sort `Exp`, and p of sort `BlockStatements`, if we have,

$$E \vdash (\forall \emptyset)\ s_0 \mid \texttt{while}\ t\ p\ :\ \texttt{WrappedState}$$

then there is an $n \geq 0$ such that:

Observations and Lemmas (IV)

- $E \vdash (\forall \emptyset)\ s_0 \mid \texttt{while}\ t\ p = s_0 \mid (t \mid p)^n \mid t$
- $E \vdash (\forall \emptyset)\ \texttt{evalTst}(s_0 \mid (t \mid p)^i, t) = true,\ 0 \leq i < n$
- $E \vdash (\forall \emptyset)\ \texttt{evalTst}(s_0 \mid (t \mid p)^n, t) = false.$

Proof of Soundness of the Loop Rule

Theorem: The loop rule is sound.

Proof: We have to prove that the assumption

$$\{A \wedge (\texttt{evalTst(S,t)} = \texttt{true})\}\ \texttt{t | p}\ \{A\} \quad \{A \wedge (\texttt{evalTst(S,t)} = \texttt{false})\}\ \texttt{t}\ \{A \wedge (\texttt{evalTst(S,t)} = \texttt{false})\}$$

implies

$$(\dagger)\ \{A\}\ \texttt{while t p}\ \{A \wedge (\texttt{evalTst(S,t)} = \texttt{false})\}$$

By the semantics of Hoare triples applied to $(\dagger)$, we only need to show,

$$T_{\texttt{JAVAX}} \models (\forall V)\ A \wedge \texttt{(S|loop):WrappedState} \Rightarrow A\texttt{(S/S | loop)} \wedge \texttt{(evalTst(S | loop,t)} = \texttt{false)},$$

Proof of Soundness of the Loop Rule (II)

where, by definition, `loop` = `while t p`. Putting aside the termination assumption (`(S|loop):WrappedState`) we have,

$$(\ddagger)\ \ T_{\texttt{JAVAX}} \models (\forall V)\ A \Rightarrow A(\texttt{S/S | loop}) \wedge (\texttt{evalTst(S | loop,t)} = \texttt{false}),$$

But, by definition of satisfaction of a universally quantified formula in an initial algebra, this is equivalent to proving, from the above assumption,

$$(\forall s \in T_{\texttt{JAVAX,WrappedState}})\ T_{\texttt{JAVAX}} \models (\forall V - \{\texttt{S}\})\ A(\texttt{S}/s) \Rightarrow A(\texttt{S}/s\texttt{|loop}) \wedge (\texttt{evalTst}(s\texttt{|loop}, t) = \texttt{false})$$

That is, for each $s \in T_{\texttt{JAVAX,WrappedState}}$, we have to show,

$$(\flat)\ \ T_{\texttt{JAVAX}} \models (\forall V - \{\texttt{S}\})\ A(\texttt{S}/s) \Rightarrow A(\texttt{S}/s\texttt{|loop}) \wedge (\texttt{evalTst}(s\texttt{|loop}, t) = \texttt{false}).$$

Proof of Soundness of the Loop Rule (III)

By the termination assumption and the last corollary we know that, for each ground substitution $\theta : V - \{\mathtt{S}\} \longrightarrow T_{\Sigma_{\mathtt{JAVAX}}}$, if

$$(\star)\ \ T_{\mathtt{JAVAX}} \models (\forall \emptyset)\ (\theta(A(\mathtt{S}/s)),$$

then there is an $n \geq 0$ such that:

1. $E \vdash (\forall \emptyset)\ s \mid\ \mathtt{loop} = s\,|\,(\mathtt{t}\,|\,\mathtt{p})^n\,|\,\mathtt{t}$
2. $E \vdash (\forall \emptyset)\ \mathtt{evalTst}(s\,|\,(\mathtt{t}\,|\,\mathtt{p})^i, t) = true,\ 0 \leq i < n$
3. $E \vdash (\forall \emptyset)\ \mathtt{evalTst}(s\,|\,(\mathtt{t}\,|\,\mathtt{p})^n, t) = false.$

We also know that $(4) E \vdash (\forall \emptyset)\ \mathtt{evalTst}(s\,|\,(\mathtt{t}\,|\,\mathtt{p})^n\,|\,\mathtt{t}, t) = false$ holds if A holds. That is because (3) tells us that the test is

false in that state and then with the second assumption we see that the test will stay false after being executed. At the positions where we use (4) A holds.

Since (1) and (4) take care of the implication of the second conjunct in $(\flat)$ (A is on the left-hand side of the implication), we have reduced the whole matter to showing that, if $(\star)$ holds, then we must have,

Proof of Soundness of the Loop Rule (IV)

$$(\natural)\ \ T_{\texttt{JAVAX}} \models \theta(A(\texttt{S}/s\,|\,(\texttt{t}\,|\,\texttt{p})^n\,|\,\texttt{t})).$$

But this now follows by substitutivity from the first assumption and (1)–(3), by the chain of implications, where the last step requires (4) and the second assumption (A obviously holds).

$$T_{\texttt{JAVAX}} \models \theta(A(\texttt{S}/s)) \Rightarrow \theta(A(\texttt{S}/s)) \wedge \texttt{evalTst}(s,t) = true \Rightarrow \theta(A(\texttt{S}/s\,|\,\texttt{t}\,|\,\texttt{p})) \wedge \texttt{evalTst}(s\,|\,\texttt{t}\,|\,\texttt{p},t) = true \ldots$$

$$\Rightarrow \theta(A(\texttt{S}/s\,|\,(\texttt{t}\,|\,\texttt{p})^{n-1})) \wedge \texttt{evalTst}(s\,|\,(\texttt{t}\,|\,\texttt{p})^{n-1},t) = true$$

$$\Rightarrow \theta(A(\texttt{S}/s\,|\,(\texttt{t}\,|\,\texttt{p})^{n})) \wedge \texttt{evalTst}(s\,|\,(\texttt{t}\,|\,\texttt{p})^{n},t) = false \Rightarrow \theta(A(\texttt{S}/s\,|\,(\texttt{t}\,|\,\texttt{p})^{n}\,|\,\texttt{t}))$$

q.e.d.

Proving Hoare Triples in the Java+ITP Tool

The *latest* version (with support for this Java subset) of the ITP downloadable from

`http://banyan.cs.uiuc.edu/itp`

has an extension of the list of commands of the ITP specifically designed to support Hoare logic reasoning in our Java fragment.

To prove a Hoare triple

$$\{P\}\,\texttt{foo}\,\{Q\}$$

using this tool, one gives a `javax` command.

Proving Hoare Triples in the Java+ITP Tool (II)

Essentially, what the `javax` command does is to *translate* the Hoare triple goal into its semantically equivalent inductive theorem proving goal.

For example, a goal consisting of the Hoare triple

$$\{P\}\,\texttt{foo}\,\{Q\}$$

is translated into the (universally quantified) ITP goal

$$P \;\Rightarrow\; Q(S/(S\,|\;\texttt{foo}))$$

where S is the distinguished variable of sort `WrappedState`.

An Example

We can illustrate the use of the `javax` command with an absolute value program `absx`. We define the module `absx.maude` as follows:

```
fmod ABSX-JAVA is
including JAVAX .
op absolute : Int -> Int .
var I : Int .
ceq absolute(I) = I if I >= 0 = true .
ceq absolute(I) = - I if I >= 0 = false .
ops absx-init absx : -> BlockStatements .
eq absx-init = (int 'X ; int 'Z ;) .
eq absx = if (#i(0) <= 'X) ('Z = 'X ;) else ('Z = - 'X ;) .
endfm
```

Note that, in addition to defining `absx` and `absx-init`, we have also defined the `absolute` auxiliary function.

An Example (II)

The point is that using the `javax` command we can now give our desired Hoare triple directly to the ITP. We do so by: (1) loading `java-es.maude` and `absx.maude`; (2) loading `itp-tool.maude`; and (3) giving the `javax` command:

```
(javax ABSX-JAVA :
--- specification constants
(I:Int)
--- precondition
((S:WrappedState['X]) = (int(I:Int)))
--- program
absx-init
absx
--- postcondition
((S:WrappedState['Z]) = (int(absolute(I:Int))))
.)
```

An Example (III)

The ITP then responds by translating this Hoare triple into the semantically equivalent ITP goal:

```
==================================
label-sel: absx@0
==================================
A{I:Int}(
    (state(e(['X,any-loc(0)]['Z,any-loc(1)]),n(0),m([any-loc(0),int(
    any-int(0))][any-loc(1),int(any-int(1))]),out(noOutput))['X]
    = int(I:Int))
    ==>
    (state(e(['X,any-loc(0)]['Z,any-loc(1)]),n(0),m([any-loc(0),
    int(any-int(0))][any-loc(1),int(any-int(1))]),out(noOutput))
    |(if #i(0)<= 'X 'Z = 'X ; else 'Z = - 'X ;)['Z]
    = int(absolute(I:Int))))
++++++++++++++++++++++++++++++++++
```

An Example (IV)

Giving the auto command we then obtain:

```
Maude> (auto .)

==================================
label-sel: absx@0
==================================
(k(bool(0 <= I*Int)-> ?('Z = 'X ;,'Z = - 'X ;)-> stop),state(e(['X,any-loc(
    0)]['Z,any-loc(1)]),m([any-loc(0),int(I*Int)][any-loc(1),int(any-int(
    1))]),n(0),out(noOutput)))['Z]= int(absolute(I*Int))
++++++++++++++++++++++++++++++++++
```

An Example (V)

We can now give the following `split` command:

```
Maude> (split on (0 <= (I*Int)) .)

==================================
label-sel: absx@1.0
==================================
(k(bool(0 <= I*Int)-> ?('Z = 'X ;,'Z = - 'X ;)-> stop),state(e(['X,any-loc(
    0)]['Z,any-loc(1)]),m([any-loc(0),int(I*Int)][any-loc(1),int(any-int(
    1))]),n(0),out(noOutput)))['Z]= int(absolute(I*Int))
```

An Example (VI)

Using the `auto` command discharges this goal, leaving only the other identical goal in the split:

```
Maude> (auto .)

==================================
label: absx@2.0
==================================
(k(bool(0 <= I*Int)-> ?('Z = 'X ;,'Z = - 'X ;)-> stop),state(e(['X,any-loc(
    0)]['Z,any-loc(1)]),m([any-loc(0),int(I*Int)][any-loc(1),int(any-int(
    1))]),n(0),out(noOutput)))['Z]= int(absolute(I*Int))
```

An Example (VII)

Which is again discharged by the `auto` command:

```
Maude> (auto .)

q.e.d

++++++++++++++++++++++++++++++++++

Maude>
```