Types, Local Variables, Procedures

Claude Marché

Cours MPRI 2-36-1 “Preuve de Programme”

9 janvier 2012

Remainder of the last 2 lectures

- Simple IMP programs
  - only the integer datatype
  - only global variables
  - program = sequence of statements
- Hoare logic:
  - Deduction rules for triples $\{Pre\}s\{Post\}$
- Weakest Precondition:
  - $\{Pre\}s\{Post\}$ valid if $Pre \Rightarrow \mathrm{WP}(s, Post)$
- Extensions: labels, exceptions

Next Extensions

- More base types
- Scopes, local variables
- Procedures (call by value)

Outline

Base datatypes

We can assume that the language provides more base types:

- reals, standard operations +, -, *
- booleans, standard operations and, or, not, if-expression "`if then else`"
- From now on, the condition of the `if then else` and the `while do` in programs is a Boolean expression

Local logic variables

We extend the syntax of expressions by

$$e ::= \texttt{let}\ x = e_1\ \texttt{in}\ e_2$$

Example: approximated cosine

```
val x : ref real
val res : ref real

res := let y = x*x in 1.0 − 0.5 * y + 0.04166666 * y * y
```

Types, Typing Judgememt

- Types:

$$\begin{array}{rcl} \tau & ::= & b \mid \texttt{ref}\ b \\ b & ::= & \texttt{int} \mid \texttt{real} \mid \texttt{bool} \end{array}$$

  `ref` denotes types of mutable variables, or *references*
- Note: no reference to reference
- Typing judgement:

$$\Gamma \vdash e : \tau$$

  where Γ maps identifiers to types

Typing rules

Constants:

$$\frac{}{\Gamma \vdash n : \texttt{int}} \qquad \frac{}{\Gamma \vdash r : \texttt{real}}$$

$$\frac{}{\Gamma \vdash \mathit{true} : \texttt{bool}} \qquad \frac{}{\Gamma \vdash \mathit{false} : \texttt{bool}}$$

Variables:

$$\frac{x : b \in \Gamma}{\Gamma \vdash x : b} \qquad \frac{x : \texttt{ref}\ b \in \Gamma}{\Gamma \vdash x : b}$$

Let binding:

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \{x : \tau_1\} \cdot \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \tau_2}$$

- All expressions have a base type, not reference
- Practice: Why3, as in OCaml, would require to write !x for references

Semantics

Program states are augmented with a stack of local (immutable) variables

- Σ: maps (reference variable,label) to values
- Γ: maps immutable variables to values

$$
\begin{array}{rcl}
[\![n]\!]_{\Sigma,\Gamma} & = & n \\
[\![x]\!]_{\Sigma,\Gamma} & = & \Sigma(x) \\
 & & \text{if } x : \texttt{ref}\ \tau \\
[\![x]\!]_{\Sigma,\Gamma} & = & \Gamma(x) \\
 & & \text{if } x \text{ not a reference} \\
[\![e_1\ op\ e_2]\!]_{\Sigma,\Gamma} & = & [\![e_1]\!]_{\Sigma,\Gamma}\ [\![op]\!]\ [\![e_2]\!]_{\Sigma,\Gamma} \\
[\![\texttt{let}\ x = e_1\ \texttt{in}\ e_2]\!]_{\Sigma,\Gamma} & = & [\![e_2]\!]_{\Sigma,\Gamma'} \\
 & & \text{where } \Gamma' = \{x = [\![e_1]\!]_{\Sigma,\Gamma}\} \cdot \Gamma
\end{array}
$$

Hoare Logic, Weakest Preconditions

Since only the language of expressions is changed, rules for Hoare logic and WP remain the same

Practical Notes:

- theorem provers should support these extensions
- Why3 manages to transform formulas when needed (e.g. transformation of if-expressions and/or let-expressions into equivalent formulas)

Local program variables

We extend the syntax of statements with

$$
\begin{array}{rcl}
s & ::= & \texttt{let}\ id = e\ \texttt{in}\ s \\
 & | & \texttt{let}\ id = \texttt{ref}\ e\ \texttt{in}\ s
\end{array}
$$

Example: isqrt revisited

```
val x, res : ref int

isqrt:
  res := 0;
  let sum = ref 1 in
  while sum ≤ x do
    res := res + 1; sum := sum + 2 ∗ res + 1
  done
```

Typing rules for statements

- Judgement: $\Gamma \vdash s : wf$ ("well-formed")

$$\frac{x : \texttt{ref}\ b \in \Gamma \qquad \Gamma \vdash e : b}{\Gamma \vdash x := e : wf}$$

$$\frac{\Gamma \vdash e : b \qquad \{x : b\} \cdot \Gamma \vdash s : wf}{\Gamma \vdash \texttt{let}\ x = e\ \texttt{in}\ s : wf}$$

$$\frac{\Gamma \vdash e : b \qquad \{x : \texttt{ref}\ b\} \cdot \Gamma \vdash s : wf}{\Gamma \vdash \texttt{let}\ x = \texttt{ref}\ e\ \texttt{in}\ s : wf}$$

- Remaining rules are standard

Operational semantics

- One-step execution is a relation of the form

$$\Sigma, \Gamma, s \rightsquigarrow \Sigma', \Gamma', s'$$

- Previous rules are unchanged (with additional Γ, unmodified)

$$\Sigma, \Gamma, \texttt{let}\ x = e\ \texttt{in}\ s \rightsquigarrow \Sigma, \{x = [\![e]\!]_{\Sigma,\Gamma}\} \cdot \Gamma, s$$

$$\Sigma, \Gamma, \texttt{let}\ x = \texttt{ref}\ e\ \texttt{in}\ s \rightsquigarrow \{(x, \mathit{Here}) = [\![e]\!]_{\Sigma,\Gamma}\} \cdot \Sigma, \Gamma, s$$

Local program variables: Hoare rules

Exercise: propose rules for

$$\texttt{let}\ x = e\ \texttt{in}\ s$$

and

$$\texttt{let}\ x = \texttt{ref}\ e\ \texttt{in}\ s$$

Local program variables: Hoare rules

$$\frac{\{P\}s\{Q\}}{\{P[x \leftarrow e]\}\texttt{let}\ x = e\ \texttt{in}\ s\{Q\}}$$

$$\frac{\{P\}s\{Q\}}{\{P[x \leftarrow e]\}\texttt{let}\ x = \texttt{ref}\ e\ \texttt{in}\ s\{Q\}}$$

Local program variables: Hoare rules

Alternative:

$$\frac{\{P\}s\{Q\}}{\{\texttt{let}\ x = e\ \texttt{in}\ P\}\texttt{let}\ x = e\ \texttt{in}\ s\{Q\}}$$

$$\frac{\{P\}s\{Q\}}{\{\texttt{let}\ \xi = e\ \texttt{in}\ P[x \leftarrow \xi]\}\texttt{let}\ x = \texttt{ref}\ e\ \texttt{in}\ s\{Q\}}$$

Local program variables: WP rules

Exercise: propose rules for

$$\texttt{let}\ x = e\ \texttt{in}\ s$$

and

$$\texttt{let}\ x = \texttt{ref}\ e\ \texttt{in}\ s$$

Local program variables: WP rules

$$\mathsf{WP}(\texttt{let}\ x = e\ \texttt{in}\ s, Q) = \mathsf{WP}(s, Q)[x \leftarrow e]$$

$$\mathsf{WP}(\texttt{let}\ x = \texttt{ref}\ e\ \texttt{in}\ s, Q) = \mathsf{WP}(s, Q)[x \leftarrow e]$$

Local program variables: WP rules

Alternative:

$$\mathsf{WP}(\texttt{let}\ x = e\ \texttt{in}\ s, Q) = \texttt{let}\ x = e\ \texttt{in}\ \mathsf{WP}(s, Q)$$

$$\mathsf{WP}(\texttt{let}\ x = \texttt{ref}\ e\ \texttt{in}\ s, Q) = \texttt{let}\ \xi = e\ \texttt{in}\ \mathsf{WP}(s, Q)[x \leftarrow \xi]$$

Outline

Procedures

Declaration of the form

procedure $p(x_1 : b_1, \ldots, x_n : b_n)$:
  contract
  body

- Contract:
  - pre-condition
  - post-condition (label *Old* available)
  - assigned variables: clause writes
- Body: sequence of statements

Parameters are not references
(call by reference addressed in another lecture)

Example: isqrt

```
val res: ref int

procedure isqrt(x:int):
contract:
  requires: x ≥ 0
  writes: res
  ensures: res ≥ 0 and sqr(res) ≤ x < sqr(res+1)
body:
  res := 0;
  let sum = ref 1 in
  while sum ≤ x do
    res := res + 1; sum := sum + 2 ∗ res + 1
  done
```

Example using Old label

```
val res: ref int

procedure incr(x:int):
contract:
  requires: true
  writes: res
  ensures: res = res@Old + x
body:
  res := res + x
```

Operational semantics

procedure $p(x_1 : \tau_1, \ldots, x_n : \tau_n)$:
  requires *Pre*
  writes $\vec{w}$
  ensures *Post*
  body *Body*

$$\frac{\Gamma' = \{x_i \leftarrow [\![e_i]\!]_{\Sigma,\Gamma}\} \qquad [\![pre]\!]_{\Sigma,\Gamma'} \text{ holds}}{\Sigma, \Gamma, p(e_1, \ldots, e_n) \rightsquigarrow \Sigma, \Gamma', (\texttt{Old} : \mathit{Body}; \mathit{return}(\mathit{post}, \Gamma))}$$

where *return* is a dummy statement introduced for this definition

$$\frac{[\![post]\!]_{\Sigma,\Gamma'} \text{ holds}}{\Sigma, \Gamma', \mathit{return}(\mathit{post}, \Gamma) \rightsquigarrow \Sigma, \Gamma, \texttt{skip}}$$

Hoare rule for procedure call (without Old)

procedure $p(x_1 : \tau_1, \ldots, x_n : \tau_n)$:
  requires Pre
  writes $\vec{w}$
  ensures $Post$
  body $Body$

$$\frac{}{\{Pre[x_i \leftarrow e_i]\}p(e_1, \ldots, e_n)\{Post[x_i \leftarrow e_i]\}}$$

- instance of precondition holds
- instance of postcondition holds

Example: isqrt(42)

Exercise: Prove that $\{true\} isqrt(42) \{res = 6\}$ holds using Hoare logic.

Hoare rule for procedure call (with Old)

procedure $p(x_1 : \tau_1, \ldots, x_n : \tau_n)$:
  requires Pre
  writes $\vec{w}$
  ensures $Post$
  body $Body$

$$\frac{}{\{Pre[x_i \leftarrow e_i] \wedge w = \xi\} f(e_1, \ldots, e_n) \{Post[x_i \leftarrow e_i][w@Old \leftarrow \xi]\}}$$

- instance of precondition holds
- instance of postcondition holds, where references to $w@Old$ are replaced by auxiliary variables

Example: incrementation

Exercise: Prove that $\{res = 6\} incr(36) \{res = 42\}$ holds using Hoare logic.

Soundness

Assuming that for each procedure declared as

procedure $p(x_1 : \tau_1, \ldots, x_n : \tau_n)$:
  requires Pre
  writes $\vec{w}$
  ensures $Post$
  body $Body$

- variables assigned in $Body$ belong to $\vec{w}$
- the triple $\{Pre\}Body\{Post\}$ is valid

then for any valid triple $\{P\}s\{Q\}$, s executes safely in any state satisfying P, and if it terminates the resulting state satisfies Q.

WLP for procedure call

procedure $p(x_1 : \tau_1, \ldots, x_n : \tau_n)$:
  requires Pre
  writes $\vec{w}$
  ensures $Post$
  body $Body$

$$\mathrm{WLP}(p(e_1, \ldots, e_n), Q) = Pre[x_i \leftarrow e_i] \wedge \\ \forall \vec{y}, (Post[x_i \leftarrow e_i][\vec{w} \leftarrow \vec{y}][\vec{w}@Old \leftarrow \vec{w}@Here] \Rightarrow Q[\vec{w} \leftarrow \vec{y}])$$

- instance of precondition holds
- instance of postcondition of p implies Q for any possible resulting values y of assigned variables $\vec{w}$.

WLP for procedure call: example

Exercise: Prove that $\{true\}isqrt(42)\{res = 6\}$ holds using WLP calculus

See file `isqrt_call.mlw`

Example: incrementation

Exercise: Prove that $\{res = 6\} incr(36) \{res = 42\}$ holds using WLP calculus

See file `incr.mlw`

Soundness

Assuming that for each procedure declared as

procedure $p(x_1 : \tau_1, \ldots, x_n : \tau_n)$:
  requires Pre
  writes $\vec{w}$
  ensures $Post$
  body $Body$

- variables assigned in $Body$ belong to $\vec{w}$
- The formula $\forall \vec{y}.Pre \Rightarrow \mathrm{WLP}(Body, Post)$ is valid

then for any triple $\{P\}s\{Q\}$ such that $P \Rightarrow \mathrm{WLP}(s, Q)$, s executes safely in any state satisfying P, and if it terminates the resulting state satisfies Q.

Procedures raising exceptions

A generalized contract has the form

requires Pre
writes $\vec{w}$
raises $E_1 \cdots E_k$
ensures $Post \mid E_1 \rightarrow Post_1 \mid \cdots \mid E_k \rightarrow Post_k$

In the rule for WLP, the implication $Post[\ldots] \Rightarrow Q$ must be replaced by a conjunction of implications

$$(Post[\ldots] \Rightarrow Q) \wedge \bigwedge_i (Post_i[\ldots] \Rightarrow Q_i)$$

Exceptions example: exact square root

```
val res: ref int
exception NotSquare

procedure isqrt(x:int):
contract:
  requires: true
  writes: res
  ensures: res ≥ 0 and sqr(res) = x
        | NotSquare → forall n:int. sqr(n) ≠ x
body:
  if x < 0 then raise NotSquare;
  res := 0;
  let sum = ref 1 in
  while sum ≤ x do
    res := res + 1; sum := sum + 2 ∗ res + 1
  done;
  if res ∗ res ≠ x then raise NotSquare
```

Exceptions example: exact square root

See file `isqrt_exc.mlw`

See file `isqrt_exc_call.mlw`

Recursive Procedures: Termination

If a procedure is recursive, the termination of call can be proved, provided that the procedure is annotated with a variant, similar to loop variant.

procedure $p(x_1 : \tau_1, \ldots, x_n : \tau_n)$:
  requires Pre
  variant var for $\prec$
  writes $\vec{w}$
  ensures $Post$
  body $Body$

$$\mathrm{WP}(p(e_1, \ldots, e_n), Q) = Pre[x_i \leftarrow e_i] \wedge var \prec var@Init$$
$$\forall \vec{y}, (Post[x_i \leftarrow e_i][\vec{w} \leftarrow \vec{y}][\vec{w}@Old \leftarrow \vec{w}@Here] \Rightarrow Q[\vec{w} \leftarrow \vec{y}])$$

where $Init$ is a label assumed to be present at the beginning of the Body

Example: McCarthy's 91 function

$$f91(n) = \texttt{if}\ n \leq 100\ \texttt{then}\ f91(f91(n+11))\ \texttt{else}\ n - 10$$

Exercise: Find adequate specifications

```
val res : ref int

procedure f91 (n:int) :
  requires ?
  variant ?
  writes ?
  ensures ?
body:
  if n ≤ 100 then f91 (n + 11); f91 !res
  else res := n − 10
```