Language Extensions

Arrays

We extend the language as follows

$$
\begin{array}{llllll}
\mathbf{Exp_{arr}} & \ni & a & ::= & u \\
\mathbf{Exp_{int}} & \ni & e & ::= & \ldots \mid a[e] \\
\mathbf{Comm} & \ni & C & ::= & \ldots \mid a[e] := e' \\
\mathbf{Term_{arr}} & \ni & t_a & ::= & u \mid t_a[t : t'] \\
\mathbf{Term} & \ni & t & ::= & \ldots \mid t_a[t]
\end{array}
$$

Semantics

$$\Sigma = (\mathcal{V} \to \mathbb{Z}) \times (\mathcal{A} \to \mathbb{Z} \to \mathbb{Z})$$

$$\begin{array}{rcl}
[\![\cdot]\!]_{\mathbf{Exp_{arr}}} & : & \mathbf{Exp_{arr}} \to \Sigma \to \mathbb{Z} \to \mathbb{Z} \\
[\![u]\!]_{\mathbf{Exp_{arr}}}(s) & = & s_a(u), \text{ where } s = (s_v, s_a)
\end{array}$$

$$\begin{array}{rcl}
[\![\cdot]\!]_{\mathbf{Term_{arr}}} & : & \mathbf{Term_{arr}} \to \Sigma \to \mathbb{Z} \to \mathbb{Z} \\
[\![u]\!]_{\mathbf{Term_{arr}}}(s) & = & s_a(u), \text{ where } s = (s_v, s_a) \\
[\![t_a[e : e']]\!]_{\mathbf{Term_{arr}}}(s) & = & [\![t_a]\!](s)([\![e]\!](s) : [\![e']\!](s))
\end{array}$$

$$[\![u[e]]\!]_{\mathbf{Exp_{int}}}(s) = [\![u]\!](s)([\![e]\!](s))$$

Semantics

The simple semantics of arrays poses problems:

- what is the value of an array position when the index is out of bounds?

- Assignment command to an out-of-bounds position of an array; how should it be evaluated?

We will add error-detection to the language.

Example

$$
\begin{array}{l}
\textbf{maxarray} \doteq \\
\quad max := 0\,; \\
\quad i := 1\,; \\
\quad \textbf{while}\ i < size\ \textbf{do}\,\{1 <= i <= size\ \&\&\ 0 <= max < i\ \&\& \\
\qquad\qquad\qquad \texttt{Forall}\ a.\ 0 <= a < i \rightarrow u[a] <= u[max]\} \\
\quad \{ \\
\qquad \textbf{if}\ u[i] > u[max]\ \textbf{then}\ max := i\ \textbf{else}\ \textbf{skip}\,; \\
\qquad i := i + 1 \\
\quad \}
\end{array}
$$

$$\{size >= 1\}$$

maxarray

$$\{0 <= max < size\ \&\&\ \texttt{Forall}\ a.\ 0 <= a < size \rightarrow u[a] <= u[max]\}$$

VCGen can be applied as before:

$$\mathsf{wp}\,(\mathbf{maxarray}, Q) = \quad 1 <= 1 <= size \;\&\&\; 0 <= 0 < 1 \;\&\&\; \texttt{Forall}\; a.\; 0 <= a < 1 \rightarrow u[a] <= u[0]$$

$$\begin{array}{l}
\mathsf{VC}_{\mathsf{aux}}(\mathbf{maxarray}, Q) = \{ \\
[\ (1 <= i < size \;\&\&\; 0 <= max < i \;\&\& \\
\qquad \texttt{Forall}\; a.\; 0 <= a < i \rightarrow u[a] <= u[max]) \rightarrow \mathsf{wp}\,(C, I)\ ], \\
[\ (1 <= i == size \;\&\&\; 0 <= max < i \;\&\& \\
\qquad \texttt{Forall}\; a.\; 0 <= a < i \rightarrow u[a] <= u[max]) \rightarrow Q\ ] \\
\}
\end{array}$$

$$\begin{array}{ll}
\mathsf{VCG}(\{size >= 1\}\, C\, \{Q\}) = & \{[\ size >= 1 \rightarrow \mathsf{wp}\,(\mathbf{maxarray}, Q)\ ]\} \\
& \cup\ \mathsf{VC}_{\mathsf{aux}}(\mathbf{maxarray}, Q)
\end{array}$$

Hoare logic rule for arrays

Previous example did not include array assignment.
What should be the HL rule to deal with assignment?

$$\frac{}{\{P\}\, u[e] := e' \,\{Q\}} \quad \text{if } \models P \rightarrow Q[u[e] \mapsto e']$$

This handles *aliasing* inadequately. It would derive for instance the incorrect triple

$$\{u[j] > 100\}\ u[i] := 10\ \{u[j] > 100\}$$

Correct rule would use an array update operation

$$\frac{}{\{P\}\, u[e] := e' \,\{Q\}} \quad \text{if } \models P \rightarrow Q[u \mapsto u[e : e']]$$

This would derive

$$\{u\ [i{:}10]\ [j] > 100\}\ u[i] := 10\ \{u[j] > 100\}$$

Theory of applicative arrays

$$\texttt{Forall } u, x, e.\ u[x : e][x] = e$$

$$\texttt{Forall } u, x, e, y.\ x! = y \rightarrow u[x : e][y] = u[y]$$

$$\texttt{Forall } u, x, e.\ length(u[x : e]) = length(u)$$

VCGen for Arrays

Our VCGen can be extended as follows

$$\mathsf{wp}\,(u[e] := e', Q) \quad = \quad Q[u \mapsto u[e : e']]$$

$$\mathsf{VC_{aux}}(u[e] := e', Q) \quad = \quad \emptyset$$

In the absence of the theory of arrays, conditions can always be translated as follows

$$u[i : 10][j] > 100 \quad \mapsto \quad (i == j \rightarrow 10 > 100)\ \&\&\ (i\,!= j \rightarrow u[j] > 100)$$

Example

$$
\begin{array}{l}
\textbf{factab} \doteq \\
\quad k := 0; \\
\quad \textbf{while}\ k < size\ \textbf{do}\ \{0 <= k <= size\ \&\& \\
\qquad\qquad\qquad\qquad \texttt{Forall}\ a.\ 0 <= a < k \rightarrow out[a] == fact(in[a])\} \\
\quad \{ \\
\qquad f := 1;\ i := 1;\ n := in[k]; \\
\quad \textbf{while}\ i <= n\ \textbf{do}\ \{i <= n + 1\ \&\&\ f == fact(i-1)\}\ \{ \\
\qquad\quad f := f * i; \\
\qquad\quad i := i + 1 \\
\quad \} \\
\quad out[k] := f; \\
\quad k := k + 1 \\
\}
\end{array}
$$

$$
\begin{array}{l}
\{size >= 0\ \&\&\ \texttt{Forall}\ a.\ 0 <= a < size \rightarrow in[a] >= 0\} \\
\textbf{factab} \\
\{\texttt{Forall}\ a.\ 0 <= a < size \rightarrow out[a] == fact(in[a])\}
\end{array}
$$

Program Errors

It is easy to adapt the language semantics to make it more realistic, including an error value and an error state such that, for instance,

$$[\![x \ \mathbf{div}\ y > 2]\!]_{\mathbf{Exp_{bool}}}\{x \mapsto 10, y \mapsto 0\} = \mathbf{error}$$

$$(\mathbf{if}\ x \ \mathbf{div}\ y > 2\ \mathbf{then}\ C_t\ \mathbf{else}\ C_f, \{x \mapsto 10, y \mapsto 0\}) \Downarrow \mathbf{error}$$

Hoare logic and Errors

The definition of Hoare triple can now be changed to include the requirement that execution does not go wrong:

$$[\![\{P\}\, C\, \{Q\}]\!] = \forall s, s' \in \Sigma.\ [\![P]\!](s) \wedge (C, s) \Downarrow s' \Longrightarrow s' \neq \mathbf{error} \wedge [\![Q]\!](s')$$

We will modify the system of Hoare logic by including additional side conditions in the rules, so that it will derive only valid triples with respect to the above definition

Safe Expression Evaluation

$$
\begin{array}{rcl}
safe(x) & \doteq & \textbf{true for } x \in \mathcal{V} \cup \mathcal{A} \\
safe(c) & \doteq & \textbf{true for } c \in \{\textbf{true}, \textbf{false}\} \cup \{\ldots, -1, 0, 1, \ldots\} \\
safe(e_1 \Box e_2) & \doteq & safe(e_1)\ \&\&\ safe(e_2) \\
 & & \text{where } \Box \in \{+, -, *, ==, <, <=, >, >=, !=\} \\
safe(e_1 \Box e_2) & \doteq & safe(e_1)\ \&\&\ safe(e_2)\ \&\&\ e_2 \neq 0 \\
 & & \text{where } \Box \in \{\ \textbf{div}\ ,\ \textbf{mod}\ \} \\
safe(-e) & \doteq & safe(e) \\
safe(u[e']) & \doteq & safe(e')\ \&\&\ valid\_index(u, e') \\
safe(b_1 \Box b_2) & \doteq & safe(b_1)\ \&\&\ safe(b_2) \text{ where } \Box \in \{\ \&\&\ ,\ \|\ \} \\
safe(\ !b) & \doteq & safe(b)
\end{array}
$$

$$valid\_index(u, i) \doteq 0 <= i < length(u)$$

Example

$$
\begin{aligned}
safe(x \ \mathbf{div}\ y > 2) &= safe(x)\ \&\&\ safe(y)\ \&\&\ y \neq 0\ \&\&\ safe(2) \\
&= y \neq 0
\end{aligned}
$$

We also introduce the following macro:

$$valid\_range(u, i, j) \doteq 0 <= i <= j < length(u) \parallel i > j$$

Hoare Logic with Safety

$$\frac{}{\{P\}\,\mathbf{skip}\,\{Q\}} \quad \text{if } \models P \to Q$$

$$\frac{}{\{P\}\,x := e\,\{Q\}} \quad \text{if } \models P \to \mathit{safe}(e) \text{ and } \models P \to Q[x \mapsto e]$$

$$\frac{}{\{P\}\,u[e] := e'\,\{Q\}} \quad \text{if } \begin{array}{l} \models P \to \mathit{safe}(u[e]) \text{ and } \models P \to \mathit{safe}(e') \\ \text{and } \models P \to Q[u \mapsto u[e : e']] \end{array}$$

$$\frac{\{P\}\, C_1\, \{R\} \qquad\qquad \{R\}\, C_2\, \{Q\}}{\{P\}\, C_1\, ;\, C_2\, \{Q\}}$$

$$\frac{\{I\ \&\&\ b\}\, C\, \{I\}}{\{P\}\, \textbf{while}\ b\ \textbf{do}\, \{I\}\, C\, \{Q\}} \quad \text{if} \quad \models P \rightarrow I \ \text{ and } \models I \rightarrow \mathit{safe}(b) \text{ and } \models I\ \&\&\ !b \rightarrow Q$$

$$\frac{\{P\ \&\&\ b\}\, C_t\, \{Q\} \qquad\qquad \{P\ \&\&\ !b\}\, C_f\, \{Q\}}{\{P\}\, \textbf{if}\ b\ \textbf{then}\ C_t\ \textbf{else}\ C_f\, \{Q\}} \quad \text{if} \quad \models P \rightarrow \mathit{safe}(b)$$

Safety-aware VCGen

Just replace the weakest prec. function by:

$$
\begin{aligned}
\mathsf{wp}^{\mathsf{s}}\left(\mathbf{skip}, Q\right) &= Q \\
\mathsf{wp}^{\mathsf{s}}\left(x := e, Q\right) &= safe(e)\ \&\&\ Q[x \mapsto e] \\
\mathsf{wp}^{\mathsf{s}}\left(u[e] := e', Q\right) &= safe(u[e])\ \&\&\ safe(e')\ \&\&\ Q[u \mapsto u[e : e']] \\
\mathsf{wp}^{\mathsf{s}}\left(C_1; C_2, Q\right) &= \mathsf{wp}^{\mathsf{s}}\left(C_1, \mathsf{wp}^{\mathsf{s}}\left(C_2, Q\right)\right) \\
\mathsf{wp}^{\mathsf{s}}\left(\mathbf{if}\ b\ \mathbf{then}\ C_t\ \mathbf{else}\ C_f, Q\right) &= safe(b)\ \&\&\ (b \rightarrow \mathsf{wp}^{\mathsf{s}}\left(C_t, Q\right))\ \&\&\ (\ !b \rightarrow \mathsf{wp}^{\mathsf{s}}\left(C_f, Q\right)) \\
\mathsf{wp}^{\mathsf{s}}\left(\mathbf{while}\ b\ \mathbf{do}\ \{I\}\, C, Q\right) &= safe(b)\ \&\&\ I
\end{aligned}
$$

The VCGen thus obtained is correct w.r.t. the system of Hoare logic with safety

Exercise

Calculate safety conditions for maxarray and modify the specification in order for the program to be successfully verified.

Procedures

$$\begin{array}{rcl} \mathbf{Comm} \ni C & ::= & \ldots \mid \mathbf{call}\,\mathbf{f} \\ \mathbf{Proc} \ni \Phi & ::= & \mathbf{pre}\ A\ \mathbf{post}\ A\ \mathbf{proc}\ \mathbf{f} = C \\ \mathbf{Prog} \ni \Pi & ::= & \Phi \mid \Phi\ \Lambda \end{array}$$

Dado um procedimento $\mathbf{pre}\ P\ \mathbf{post}\ Q\ \mathbf{proc}\ \mathbf{f} = C$
os operadores pre, post e body devolvem o que o nome indica:

$$\begin{array}{rcl} \mathbf{pre}(\mathbf{f}) & = & P \\ \mathbf{post}(\mathbf{f}) & = & Q \\ \mathbf{body}(\mathbf{f}) & = & C \end{array}$$

Correct Program

A program is thus a collection of procedures.

A program π is correct if all its procedures are correct with respect to their corresponding specications, i.e. for all f defined in π, the following Hoare triple is valid.

$$\{\mathbf{pre}(f)\}\ \mathbf{body}(f)\ \{\mathbf{post}(f)\}$$

i.e. f obeys its *contract*.

Hoare Logic for Procedures

$$\frac{}{\{P\}\,\mathbf{call\,f}\,\{Q\}} \quad \text{if } \models P \rightarrow \mathbf{pre(f)} \text{ and } \models \mathbf{post(f)} \rightarrow Q$$

In reasoning about calls to f , we assume that the correctness of f has been proved independently. Each procedure is proved correct with respect to its specification, and reasoning about calls to f does not require knowledge of how f is implemented.

Design by contract!

Example

$$\textbf{pre}\ n >= 0$$

$$\textbf{post}\ f == fact(n)$$

$$\textbf{proc fact} =$$

$$f := 1\,;\ i := 1\,;$$

$$\textbf{while}\ i <= n\ \textbf{do}\ \{f == fact(i-1)\ \&\&\ i <= n+1\}$$

$$f := f * i\,;$$

$$i := i + 1$$

We already proved $\{n >= 0\}\,\textbf{body}(\textbf{fact})\,\{f == fact(n)\}$
and can now derive for instance

$$\{x >= -10\ \&\&\ n == x + 20\}\,\textbf{call fact}\,\{f == fact(n)\}$$

Exercise: VCs for Procedures

How should the VCGen deal with procedures?

Contracts

```
/*@ requires P
  @ ensures Q
  @*/
... C (...) {
      ...
}
```

a procedure / method corresponds to a Hoare triple

{P} C {Q}

Verifying a program / class implies verifying the set of Hoare triples generated by its procedures / methods

Contracts

What about procedure / function / method *calls*?

$$\mathsf{wp}\,(\mathbf{call}\,\mathbf{f}(e_1,\ldots,e_n),Q) = \quad \mathbf{pre}(\mathbf{f})[\overline{x} \mapsto \overline{e}]$$

where $\overline{x} = \mathbf{param}(\mathbf{f}) = x_1,\ldots,x_n$ and $\overline{e} = e_1,\ldots,e_n$.

$$\mathsf{VC}_{\mathsf{aux}}(\mathbf{call}\,\mathbf{f}(x_1,\ldots,x_n),Q) = \quad \{[\,\mathbf{post}(\mathbf{f})[\overline{x} \mapsto \overline{e}] \;\rightarrow\; Q\,]\}$$