

Program verification

Nelma Moreira

Departamento de Ciência de Computadores da FCUP

Program verification
Lecture 7

Verification Conditions for programs with arrays

Let `maxarray` be the following program:

```
max ← 0;  
i ← 1;  
while i < size do  
  if  $u[i] > u[\textit{max}]$  then  
    max ← i  
  else  
    skip;  
  i ← i + 1
```

We want to check that

$\{size \geq 1\}$

`maxarray`

$\{0 \leq \textit{max} < \textit{size} \wedge \forall a. 0 \leq a < \textit{size} \rightarrow u[a] \leq u[\textit{max}]\}$

Which is the invariant?

The annotated program is:

Require: $\{size \geq 1\}$

$max \leftarrow 0;$

$i \leftarrow 1;$

while $i < size$ **do** $\{\theta\}$

if $u[i] > u[max]$ **then**

$max \leftarrow i$

else

 skip;

$i \leftarrow i + 1$

Ensure: $\{0 \leq max < size \wedge \forall a. 0 \leq a < size \rightarrow u[a] \leq u[max]\}$

where the invariant is

$$\theta = 1 \leq i \leq size \wedge 0 \leq max < i \wedge \forall a. 0 \leq a < i \rightarrow u[a] \leq u[max]$$

The verification conditions can be calculated by applying the VCG for

$$\{size \geq 1\}$$

maxarray

$$\{0 \leq max < size \wedge \forall a. 0 \leq a < size \rightarrow u[a] \leq u[max]\}$$

We assume

$$\theta = 1 \leq i \leq size \wedge 0 \leq max < i \wedge \forall a. 0 \leq a < i \rightarrow u[a] \leq u[max]$$
$$C = \text{if } u[i] > u[max] \text{ then } max \leftarrow i \text{ else skip; } i \leftarrow i + 1;$$
$$\psi = 0 \leq max < size \wedge \forall a. 0 \leq a < size \rightarrow u[a] \leq u[max]$$

We have

$$\begin{aligned} VCG(\{size \geq 1\} \text{maxarray} \{\psi\}) &= \{size \geq 1 \rightarrow wp(\text{maxarray}, \psi)\} \\ &\cup VC(\text{maxarray}, \psi). \end{aligned}$$

$$\begin{aligned}
wp(C, \theta) &= (u[i] > u[max] \rightarrow (1 \leq i + 1 \leq size \\
&\wedge 0 \leq i < i + 1 \wedge \forall a. 0 \leq a < i + 1 \rightarrow u[a] \leq u[i]) \\
&\wedge (u[i] \leq u[max] \rightarrow (1 \leq i + 1 \leq size \\
&\wedge 0 \leq max < i + 1 \\
&\wedge \forall a. 0 \leq a < i + 1 \rightarrow u[a] \leq u[max]))
\end{aligned}$$

$$\begin{aligned}
wp(maxarray, \theta) &= (1 \leq 1 \leq size \\
&\wedge 0 \leq 0 < 1 \wedge \forall a. 0 \leq a < 1 \rightarrow u[a] \leq u[0])
\end{aligned}$$

$$\begin{aligned}
VC(maxarray, \psi) &= \{\theta \wedge i < size \rightarrow wp(C, \theta), \theta \wedge i \geq size \rightarrow \psi\} \\
&= \{(1 \leq i < size \wedge 0 \leq max < i \wedge \\
&\quad \forall a. 0 \leq a < i \rightarrow u[a] \leq u[max]) \rightarrow wp(C, \theta), \\
&\quad 1 \leq i = size \wedge 0 \leq max < i \wedge \\
&\quad \forall a. 0 \leq a < i \rightarrow u[a] \leq u[max]) \rightarrow \psi\}
\end{aligned}$$

We add the following rule to \mathcal{H}_g :

$$\frac{}{\{\varphi\} u[E] \leftarrow E' \{\psi\}} \text{ if } \models \varphi \rightarrow \psi[u[E \triangleright E'] / u]$$

we expand wp and VC in the following way:

$$\begin{aligned} wp(u[E] \leftarrow E', \psi) &= \psi[u[E \triangleright E'] / u] \\ VC(u[E] \leftarrow E', \psi) &= \emptyset \end{aligned}$$

For instance:

$$\begin{aligned} wp(u[i] \leftarrow 10, u[j] > 100) &= u[i \triangleright 10][j] > 100 \\ VC(u[i] \leftarrow 10, u[j] > 100) &= \emptyset \end{aligned}$$

Exerc.

Using the VCG algorithm calculate:

- 1 $VCG(\{u[j] > 100\}u[i] \leftarrow 10\{u[j] > 100\})$
- 2 $VCG(\{i \neq j \wedge u[j] > 100\}u[i] \leftarrow 10\{u[j] > 100\})$
- 3 $VCG(\{i = 70\}u[i] \leftarrow 10\{u[i] = 10\})$

◇

In the operational semantics we considered, every expression evaluates to a value and command execution would not produce any error.

We now consider some modifications that approximate the language to a real programming language:

- incorporating in the language semantics a special error value;
- modifying the evaluation relation to admit evaluation of commands to a special error state;

Error semantics for arithmetic expressions

$\mathcal{A} : \mathbf{Aexp} \rightarrow (\mathbf{State} \rightarrow (\mathbb{Z} \cup \{\text{error}\}))$

$$\mathcal{A}[[n]]s = n$$

$$\mathcal{A}[[x]]s = s(x)$$

$$\mathcal{A}[[E_1 \odot E_2]]s = \begin{cases} \mathcal{A}[[E_1]]s \odot \mathcal{A}[[E_2]]s & \text{if } \mathcal{A}[[E_1]]s \neq \text{error} \neq \mathcal{A}[[E_2]]s \\ \text{error} & \text{otherwise} \end{cases}$$

para $\odot \in \{+, -, \times\}$

$$\mathcal{A}[[E_1 \div E_2]]s = \begin{cases} \mathcal{A}[[E_1]]s \div \mathcal{A}[[E_2]]s & \text{if } \mathcal{A}[[E_1]]s \neq \text{error} \neq \mathcal{A}[[E_2]]s \\ & \text{and } \mathcal{A}[[E_2]]s \neq 0 \\ \text{error} & \text{otherwise} \end{cases}$$

Error semantics for Boolean expressions

$\mathbf{T} = \{\text{true}, \text{false}\}$, $\mathcal{B} : \mathbf{Bexp} \rightarrow (\mathbf{State} \rightarrow (\mathbf{T} \cup \{\text{error}\}))$

$$\mathcal{B}[\text{true}]s = \text{true}$$

$$\mathcal{B}[\text{false}]s = \text{false}$$

$$\mathcal{B}[\neg b]s = \begin{cases} \text{true} & \text{if } \mathcal{B}[b]s = \text{false} \\ \text{false} & \text{if } \mathcal{B}[b]s = \text{true} \\ \text{error} & \text{if } \mathcal{B}[b]s = \text{error} \end{cases}$$

$$\mathcal{B}[E_1 \odot E_2]s = \begin{cases} \mathcal{A}[E_1]s \odot \mathcal{A}[E_2]s & \text{if } \mathcal{A}[E_1]s \neq \text{error} \neq \mathcal{A}[E_2]s \\ \text{error} & \text{otherwise} \end{cases}$$

for $\odot \in \{=, <, \leq\}$.

$$\mathcal{B}[b_1 \wedge b_2]s = \begin{cases} \text{false} & \text{if } \mathcal{B}[b_1]s = \text{false} \\ \text{error} & \text{if } \mathcal{B}[b_1]s = \text{error} \\ \mathcal{B}[b_1]s & \text{otherwise} \end{cases}$$

Natural semantics with errors (*big-step*)

$$\langle \text{skip}, s \rangle \longrightarrow s$$
$$\langle x \leftarrow E, s \rangle \longrightarrow \begin{cases} s[A[[E]]s/x] & \text{if } A[[E]]s \neq \text{error} \\ \text{error} & \text{otherwise} \end{cases}$$
$$\frac{\langle C_1, s \rangle \longrightarrow \text{error}}{\langle C_1; C_2, s \rangle \longrightarrow \text{error}}$$
$$\frac{\langle C_1, s \rangle \longrightarrow s', \langle C_2, s' \rangle \longrightarrow s''}{\langle C_1; C_2, s \rangle \longrightarrow s''} \text{ if } s' \neq \text{error}$$
$$\langle \text{if } B \text{ then } C_1 \text{ else } C_2, s \rangle \longrightarrow \text{error if } B[[B]]s = \text{error}$$
$$\frac{\langle C_1, s \rangle \longrightarrow s'}{\langle \text{if } B \text{ then } C_1 \text{ else } C_2, s \rangle \longrightarrow s'} \text{ if } B[[B]]s = \text{true}$$
$$\frac{\langle C_2, s \rangle \longrightarrow s'}{\langle \text{if } B \text{ then } C_1 \text{ else } C_2, s \rangle \longrightarrow s'} \text{ if } B[[B]]s = \text{false}$$
$$\langle \text{while } B \text{ do } C, s \rangle \longrightarrow \text{error if } B[[B]]s = \text{error}$$
$$\frac{\langle C, s \rangle \longrightarrow \text{error}}{\langle \text{while } B \text{ do } C, s \rangle \longrightarrow \text{error}} \text{ if } B[[B]]s = \text{true}$$
$$\frac{\langle C, s \rangle \longrightarrow s', \langle \text{while } B \text{ do } C, s' \rangle \longrightarrow s''}{\langle \text{while } B \text{ do } C, s \rangle \longrightarrow s''} \text{ if } B[[B]]s = \text{true}, s' \neq \text{error}$$
$$\langle \text{while } B \text{ do } C, s \rangle \longrightarrow s \text{ if } B[[B]]s = \text{F}$$

To extend the deductive systems of Hoare logic

- consider the structure of each command
- consider the possible values of the expressions that occur
- associate *safety side conditions* to each expression E which we denote by $\text{safe}(E)$ (which is an assertion).

Hoare logic with safety conditions: system \mathcal{H}_s

$$\frac{}{\{\varphi\} \text{ skip } \{\psi\}} \text{ if } \varphi \rightarrow \psi$$

$$\frac{}{\{\varphi\} x \leftarrow E \{\psi\}} \text{ if } \varphi \rightarrow \text{safe}(E) \text{ and } \varphi \rightarrow \psi[E/x]$$

$$\frac{\{\varphi\} C_1 \{\eta\} \quad \{\eta\} C_2 \{\psi\}}{\{\varphi\} C_1; C_2 \{\psi\}}$$

$$\frac{\{\varphi \wedge B\} C_1 \{\psi\} \quad \{\varphi \wedge \neg B\} C_2 \{\psi\}}{\{\varphi\} \text{ if } B \text{ then } C_1 \text{ else } C_2 \{\psi\}} \text{ if } \varphi \rightarrow \text{safe}(B)$$

$$\frac{\{\eta \wedge B\} C \{\eta\}}{\{\psi\} \text{ while } B \text{ do } \{\eta\} C \{\varphi\}} \text{ if } \psi \rightarrow \eta, \eta \rightarrow \text{safe}(B) \text{ and } \eta \wedge \neg B \rightarrow \varphi$$

VCG algorithm: calculation of the weakest preconditions (wp^s)

$$wp^s(\text{skip}, \psi) = \psi$$

$$wp^s(x \leftarrow E, \psi) = \text{safe}(E) \wedge \psi[E/x]$$

$$wp^s(C_1; C_2, \psi) = wp^s(C_1, wp^s(C_2, \psi))$$

$$wp^s(\text{if } B \text{ then } C_1 \text{ else } C_2, \psi) = \text{safe}(B) \wedge (B \rightarrow wp^s(C_1, \psi)) \\ \wedge (\neg B \rightarrow wp^s(C_2, \psi))$$

$$wp^s(\text{while } B \text{ do } \{\eta\} C, \psi) = \eta$$

VCG algorithm: Compute VC without preconditions

$$VC^s(\text{skip}, \psi) = \emptyset$$

$$VC^s(x \leftarrow E, \psi) = \emptyset$$

$$VC^s(C_1; C_2, \psi) = VC^s(C_1, wp^s(C_2, \psi)) \cup \\ VC^s(C_2, \psi)$$

$$VC^s(\text{if } B \text{ then } C_1 \text{ else } C_2, \psi) = VC^s(C_1, \psi) \cup VC^s(C_2, \psi)$$

$$VC^s(\text{while } B \text{ do } \{\eta\} C, \psi) = \{\eta \rightarrow \text{safe}(B)\} \cup \\ \{(\eta \wedge B) \rightarrow wp^s(C, \eta)\} \cup \\ \{(\eta \wedge \neg B) \rightarrow \psi\} \cup VC^s(C, \eta)$$

We define VCG^s as:

$$VCG^s(\{\varphi\} C \{\psi\}) = \{\varphi \rightarrow wp^s(C, \psi)\} \cup VC^s(C, \psi)$$

The function safe for the While^{int} language

$$\text{safe}(n) = \text{true}$$

$$\text{safe}(x) = \text{true}$$

$$\text{safe}(-E) = \text{safe}(E)$$

$$\text{safe}(E_1 \odot E_2) = \text{safe}(E_1) \wedge \text{safe}(E_2)$$

with $\odot \in \{+, -, \times, =, <, \leq\}$

$$\text{safe}(E_1 \div E_2) = \text{safe}(E_1) \wedge \text{safe}(E_2) \wedge E_2 \neq 0$$

$$\text{safe}(\neg B) = \text{safe}(B)$$

$$\text{safe}(B_1 \wedge B_2) = \text{safe}(B_1) \wedge (B_1 \rightarrow \text{safe}(B_2))$$

$$\text{safe}(B_1 \vee B_2) = \text{safe}(B_1) \wedge (\neg B_1 \rightarrow \text{safe}(B_2))$$

We have

$$\mathcal{A}[[E]]s \neq \text{error} \text{ iff } [[\text{safe}(E)]]s = \text{true}.$$

Let $\{\varphi\}C\{\psi\}$ be a Hoare triple and Γ a set of assertions. Then

$$\Gamma \models VCG^s(\{\varphi\}C\{\psi\}) \text{ iff } \Gamma \vdash_{\mathcal{H}_s} \{\varphi\}C\{\psi\}.$$

Proof.

(\Rightarrow) By induction on the structure of C .

(\Leftarrow) By induction on the derivation of $\Gamma \vdash_{\mathcal{H}_s} \{\varphi\}C\{\psi\}$.

□

Ex.

$$\begin{aligned}\text{safe}((x \div y) > 2) &= \text{safe}(x) \wedge \text{safe}(y) \wedge y \neq 0 \wedge \text{safe}(2) \\ &= \text{true} \wedge \text{true} \wedge y \neq 0 \wedge \text{true} \\ &\equiv y \neq 0\end{aligned}$$

$$\begin{aligned}\text{safe}(7 > x \wedge (x \div y) > 2) &= \text{safe}(7 > x) \wedge \\ &\quad ((7 > x) \rightarrow \text{safe}((x \div y) > 2)) \\ &= \text{true} \wedge \text{true} \wedge (7 > x \rightarrow (y \neq 0)) \\ &\equiv 7 > x \rightarrow y \neq 0\end{aligned}$$

- The notion of array introduced before is unrealistic since arrays are virtually infinite.
- we will consider expressions of the form $\text{array}[N]$, representing arrays of size N , that admit as valid indexes nonnegative integers below N .
- What to do if the operations refer indexes out of the array limits?
- We consider as error situations.
- We introduce $\text{len}(A)$ that given a array A returns its length.

- The notion of array introduced before is unrealistic since arrays are virtually infinite.
- we will consider expressions of the form `array[N]`, representing arrays of size N , that admit as valid indexes nonnegative integers below N .
- What to do if the operations refer indexes out of the array limits?
- We consider as error situations.
- We introduce `len(A)` that given a array A returns its length.

- The notion of array introduced before is unrealistic since arrays are virtually infinite.
- we will consider expressions of the form $\text{array}[N]$, representing arrays of size N , that admit as valid indexes nonnegative integers below N .
- What to do if the operations refer indexes out of the array limits?
- We consider as error situations.
- We introduce $\text{len}(A)$ that given a array A returns its length.

- The notion of array introduced before is unrealistic since arrays are virtually infinite.
- we will consider expressions of the form $\text{for array}[N]$, representing arrays of size N , that admit as valid indexes nonnegative integers below N .
- What to do if the operations refer indexes out of the array limits?
- We consider as error situations.
- We introduce $\text{len}(A)$ that given a array A returns its length.

- The notion of array introduced before is unrealistic since arrays are virtually infinite.
- we will consider expressions of the form $\text{array}[N]$, representing arrays of size N , that admit as valid indexes nonnegative integers below N .
- What to do if the operations refer indexes out of the array limits?
- We consider as error situations.
- We introduce $\text{len}(A)$ that given a array A returns its length.

For $n \in \mathbf{Num}$, $x \in \mathbf{Var}$, $u \in \mathbf{Array}$

$$\text{Exp}_{\text{array}[N]} \quad A ::= u \mid A[E \triangleright E]$$

$$\begin{aligned} \text{Exp}_{\text{int}} \quad E ::= & n \mid x \mid -E \mid E + E \mid E - E \\ & \mid E \times E \mid E \div E \\ & \mid A[E] \mid \text{len}(A) \end{aligned}$$

$$\begin{aligned} \text{Exp}_{\text{bool}} \quad B ::= & \text{true} \mid \text{false} \mid \neg B \mid E = E \\ & \mid B < E \mid B \leq E \mid B \wedge B \mid B \vee B \end{aligned}$$

Semantics of the arithmetic expressions for $\text{While}_{\text{array}[N]}$

We only need to define the semantics for $\text{Exp}_{\text{array}[N]}$.

$$\mathcal{A}[[u]]s = s(u)$$

$$\mathcal{A}[[A[E \triangleright E']]]s = \begin{cases} \mathcal{A}[[A]]s[\mathcal{A}[[E']]s / \mathcal{A}[[E]]s] & \text{if} \\ \quad \mathcal{A}[[A]]s \neq \text{error} \\ \quad \mathcal{A}[[E]]s \neq \text{error} \\ \quad 0 \leq \mathcal{A}[[E]]s < \mathcal{A}[[\text{len}(A)]]s \\ \quad \mathcal{A}[[E']]s \neq \text{error} \\ \text{error} & \text{otherwise.} \end{cases}$$

e

$$\mathcal{A}[[\text{len}(A)]]s = N$$

$$\mathcal{A}[[A[E]]]s = \begin{cases} \mathcal{A}[[A]]s(\mathcal{A}[[E]]s) & \text{if} \\ \quad \mathcal{A}[[A]]s \neq \text{error} \\ \quad \mathcal{A}[[E]]s \neq \text{error} \\ \quad 0 \leq \mathcal{A}[[E]]s < \mathcal{A}[[\text{len}(A)]]s \\ \text{error} & \text{otherwise.} \end{cases}$$

We add the following rule to \mathcal{H}_g :

$$\frac{}{\{\varphi\} u[E] \leftarrow E' \{\psi\}} \text{ if } \varphi \rightarrow \text{safe}(u[E \triangleright E']) \text{ and } \varphi \rightarrow \psi[u[E \triangleright E'] / u]$$

we extend wp^s and VC^s and safe as follows:

$$wp^s(u[E] \leftarrow E', \psi) = \text{safe}(u[E \triangleright E']) \wedge \psi[u[E \triangleright E'] / u]$$

$$VC^s(u[E] \leftarrow E', \psi) = \emptyset$$

$$\text{safe}(u) = \text{true}$$

$$\text{safe}(\text{len}(A)) = \text{true}$$

$$\text{safe}(A[E]) = \text{safe}(A) \wedge \text{safe}(E) \wedge 0 \leq E \leq \text{len}(A)$$

$$\begin{aligned} \text{safe}(A[E \triangleright E']) &= \text{safe}(A) \wedge \text{safe}(E) \wedge \\ &0 \leq E \leq \text{len}(A) \wedge \text{safe}(E') \end{aligned}$$

Ex.

$$\begin{aligned}\text{safe}(u[x \div 2]) &= \text{safe}(u) \wedge \text{safe}(x \div 2) \wedge 0 \leq x \div 2 < \text{len}(u) \\ &= \text{true} \wedge \text{safe}(x) \wedge \text{safe}(2) \wedge 2 \neq 0 \\ &\quad \wedge 0 \leq x \div 2 < \text{len}(u) \\ &= \text{true} \wedge \text{true} \wedge \text{true} \wedge 2 \neq 0 \wedge 0 \leq x \div 2 < \text{len}(u) \\ &\equiv 2 \neq 0 \wedge 0 \leq x \div 2 < \text{len}(u)\end{aligned}$$

$$\begin{aligned}\text{safe}(u[3 \triangleright 10]) &= \text{safe}(u) \wedge \text{safe}(3) \wedge 0 \leq 3 < \text{len}(u) \wedge \text{safe}(10) \\ &= \text{true} \wedge \text{true} \wedge 0 \leq 3 < \text{len}(u) \wedge \text{true} \\ &\equiv 0 \leq 3 < \text{len}(u)\end{aligned}$$

Now one needs consider the range of valid indexes of an array u

$$\text{valid_range}(u, i, j) = 0 \leq i \leq j < \text{len}(u) \vee i > j$$

Require: $\{size \geq 1 \wedge \text{valid_range}(u, 0, size - 1)\}$

$max \leftarrow 0;$

$i \leftarrow 0;$

while $i < size$ **do** $\{\theta\}$

if $u[i] > u[max]$ **then**

$max \leftarrow i$

else

 skip;

$i \leftarrow i + 1$

Ensure: $\{0 \leq max < size \wedge \forall a. 0 \leq a < size \rightarrow u[a] \leq u[max]\}$

$$\begin{aligned}
\varphi &= \text{size} \geq 1 \wedge \text{valid_range}(u, 0, \text{size} - 1) \\
\theta &= \text{valid_range}(u, 0, \text{size} - 1) \wedge 1 \leq i \leq \text{size} \\
&\quad \wedge 0 \leq \text{max} < i \wedge \forall a. 0 \leq a < i \rightarrow u[a] \leq u[\text{max}] \\
C &= \text{if } u[i] > u[\text{max}] \text{ then } \text{max} \leftarrow i \text{ else skip; } i \leftarrow i + 1; \\
\psi &= 0 \leq \text{max} < \text{size} \wedge \forall a. 0 \leq a < \text{size} \rightarrow u[a] \leq u[\text{max}]
\end{aligned}$$

We have

$$\begin{aligned}
\text{VCG}^s(\{\varphi\} \text{maxarray} \{\psi\}) &= \{\varphi \rightarrow \text{wp}^s(\text{maxarray}, \psi)\} \\
&\quad \cup \text{VC}^s(\text{maxarray}, \psi).
\end{aligned}$$

$$\begin{aligned}
wp^s(C, \theta) = & \text{safe}(u[i] > u[\text{max}]) \\
& \wedge (u[i] > u[\text{max}] \rightarrow (\text{safe}(i) \wedge \text{safe}(i + 1) \\
& \wedge 1 \leq i + 1 \leq \text{size} \wedge 0 \leq i < i + 1 \\
& \wedge \forall a. 0 \leq a < i + 1 \rightarrow u[a] \leq u[i])) \\
& \wedge (\neg(u[i] \leq u[\text{max}]) \rightarrow (\text{safe}(i + 1) \\
& \wedge 1 \leq i + 1 \leq \text{size} \wedge 0 \leq \text{max} < i + 1 \\
& \wedge \forall a. 0 \leq a < i + 1 \rightarrow u[a] \leq u[\text{max}]))
\end{aligned}$$

$$\begin{aligned}
wp^s(\text{maxarray}, \theta) = & (\text{safe}(0) \wedge \text{safe}(1) \wedge 1 \leq 1 \leq \text{size} \\
& \wedge 0 \leq 0 < 1 \wedge \forall a. 0 \leq a < 1 \rightarrow u[a] \leq u[0])
\end{aligned}$$

$$\begin{aligned}
VC^s(\text{maxarray}, \psi) = & \{ \theta \rightarrow \text{safe}(i < \text{size}), \\
& (1 \leq i < \text{size} \wedge 0 \leq \text{max} < i \wedge \\
& \forall a. 0 \leq a < i \rightarrow u[a] \leq u[\text{max}]) \rightarrow wp^s(C, \theta), \\
& (1 \leq i = \text{size} \wedge 0 \leq \text{max} < i \wedge \\
& \forall a. 0 \leq a < i \rightarrow u[a] \leq u[\text{max}]) \rightarrow \psi \}
\end{aligned}$$

See [AFPMdS11] Chap. 6.5, 7.



José Bacelar Almeida, Maria João Frade, Jorge Sousa Pinto,
and Simão Melo de Sousa.

*Rigorous Software Development: An Introduction to Program
Verification.*

Springer, 2011.