Propriedades de segurança

Na semântica que consideramos todas as expressões avaliam para um determinado valor e os comandos executam sem provocarem erros. Vamos considerar algumas alterações que aproximem a nossa linguagem de uma linguagem real:

- introdução de um novo valor semântico de **erro**;
- alteração da relação de avaliação para considerar avaliações de comandos que terminem com o estado **erro**;

Semântica de expressões com erros

$\mathcal{A} : \mathbf{Aexp} \longrightarrow (\mathbf{State} \longrightarrow (Z \cup \{\mathbf{erro}\}))$

$$
\begin{aligned}
\mathcal{A}[\![n]\!]s &= n \\
\mathcal{A}[\![x]\!]s &= s(x) \\
\mathcal{A}[\![E_1 \odot E_2]\!]s &= \begin{cases} \mathcal{A}[\![E_1]\!]s \odot \mathcal{A}[\![E_2]\!]s & \text{se} \mathcal{A}[\![E_1]\!]s \neq \mathbf{erro} \neq \mathcal{A}[\![E_2]\!]s \\ \mathbf{erro} & \text{caso contrário} \end{cases} \\
& \text{para } \odot \in \{+, -, \times\} \\
\mathcal{A}[\![E_1 \div E_2]\!]s &= \begin{cases} \mathcal{A}[\![E_1]\!]s \div \mathcal{A}[\![E_2]\!]s & \text{se} \mathcal{A}[\![E_1]\!]s \neq \mathbf{erro} \neq \mathcal{A}[\![E_2]\!]s \\ & \text{e } \mathcal{A}[\![E_2]\!]s \neq 0 \\ \mathbf{erro} & \text{caso contrário} \end{cases}
\end{aligned}
$$

Semântica das expressões boleanas

$\mathbf{T} = \{\mathbf{V}, \mathrm{F}\}$, $\mathcal{B} : \mathbf{Bexp} \longrightarrow (\mathbf{State} \longrightarrow (\mathbf{T} \cup \{\mathbf{erro}\}))$

$$
\begin{aligned}
\mathcal{B}[\![\texttt{true}]\!]s &= \mathbf{V} \\
\mathcal{B}[\![\texttt{false}]\!]s &= \mathrm{F} \\
\mathcal{B}[\![\neg b]\!]s &= \begin{cases} \mathbf{V} & \text{se } \mathcal{B}[\![b]\!]s = \mathrm{F} \\ \mathrm{F} & \text{se } \mathcal{B}[\![b]\!]s = \mathbf{V} \\ \mathbf{erro} & \text{se } \mathcal{B}[\![b]\!]s = \mathbf{erro} \end{cases} \\
\mathcal{B}[\![E_1 \odot E_2]\!]s &= \begin{cases} \mathcal{A}[\![E_1]\!]s \odot \mathcal{A}[\![E_2]\!]s & \text{se } \mathcal{A}[\![E_1]\!]s \neq \mathbf{erro} \neq \mathcal{A}[\![E_2]\!]s \\ \mathbf{erro} & \text{se caso contrário} \end{cases} \\
& \text{para } \odot \in \{=, <, \leq\}. \\
\mathcal{B}[\![b_1 \ \wedge \ b_2]\!]s &= \begin{cases} \mathrm{F} & \text{se } \mathcal{B}[\![b_1]\!]s = \mathrm{F} \\ \mathbf{erro} & \text{se } \mathcal{B}[\![b_1]\!]s = \mathbf{erro} \\ \mathcal{B}[\![b_1]\!]s & \text{caso contrário} \end{cases}
\end{aligned}
$$

Semântica operacional natural com erros (*big-step*)

$$\langle \texttt{skip}, s\rangle \;\to\; s$$

$$\langle x := E, s\rangle \;\to\; \begin{cases} s[\mathcal{A}[\![E]\!]s/x] & \text{se } \mathcal{A}[\![E]\!]s \neq \textbf{erro} \\ \textbf{erro} & \text{caso contrário} \end{cases}$$

$$\frac{\langle C_1, s\rangle \;\to\; \textbf{erro}}{\langle C_1; C_2, s\rangle \;\to\; \textbf{erro}}$$

$$\frac{\langle C_1, s\rangle \;\to\; s' \;,\; \langle C_2, s'\rangle \;\to\; s''}{\langle C_1; C_2, s\rangle \;\to\; s''} \text{ se } s' \neq \textbf{erro}$$

$$\langle \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2, s\rangle \;\to\; \textbf{erro} \text{ se } \mathcal{B}[\![B]\!]s = \textbf{erro}$$

$$\frac{\langle C_1, s\rangle \;\to\; s'}{\langle \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2, s\rangle \;\to\; s'} \text{ se } \mathcal{B}[\![B]\!]s = \mathbf{V}$$

$$\frac{\langle C_2, s\rangle \;\to\; s'}{\langle \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2, s\rangle \;\to\; s'} \text{ se } \mathcal{B}[\![B]\!]s = \mathrm{F}$$

$$\langle \texttt{while } B \texttt{ do } C, s\rangle \;\to\; \textbf{erro} \text{ se } \mathcal{B}[\![B]\!]s = \textbf{erro}$$

$$\frac{\langle C, s\rangle \;\to\; \textbf{erro}}{\langle \texttt{while } B \texttt{ do } C, s\rangle \;\to\; \textbf{erro}} \text{ se } \mathcal{B}[\![B]\!]s = \mathbf{V}$$

$$\frac{\langle C, s\rangle \;\to\; s', \langle \texttt{while } B \texttt{ do } C, s'\rangle \;\to\; s''}{\langle \texttt{while } B \texttt{ do } C, s\rangle \;\to\; s''} \text{ se } \mathcal{B}[\![B]\!]s = \mathbf{V}, s' \neq \textbf{erro}$$

$$\langle \texttt{while } B \texttt{ do } C, s\rangle \;\to\; s \text{ se } \mathcal{B}[\![B]\!]s = \mathrm{F}$$

Lógica de Hoare com condições de segurança: sistema $\mathcal{H}_s$

$$\frac{}{\{\phi\}\,\texttt{skip}\,\{\psi\}} \text{ se } \phi \to \psi$$

$$\frac{}{\{\phi\}\,x := E\,\{\psi\}} \text{ se } \phi \to \texttt{safe}(E) \text{ e } \phi \to \psi[E/x]$$

$$\frac{\{\phi\}\,C_1\,\{\eta\} \qquad \{\eta\}\,C_2\,\{\psi\}}{\{\phi\}\,C_1; C_2\,\{\psi\}}$$

$$\frac{\{\phi \;\wedge\; B\}\,C_1\,\{\psi\} \qquad \{\phi \;\wedge\; \neg B\}\,C_2\,\{\psi\}}{\{\phi\}\,\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2\,\{\psi\}} \text{ se } \phi \to \texttt{safe}(B)$$

$$\frac{\{\eta \;\wedge\; B\}\,C\,\{\eta\}}{\{\psi\}\,\texttt{while } B \texttt{ do } \{\eta\} C\,\{\phi\}} \text{ se } \psi \to \eta,\ \eta \to \texttt{safe}(B) \text{ e } \eta \;\wedge\; \neg B \to \phi$$

Um algoritmo VCGen: cálculo das pré-condições mais fracas ($\mathbf{wp}^s$)

$$
\begin{array}{rcl}
wp^s(\texttt{skip}, \phi) & = & \phi \\
wp^s(x := E, \phi) & = & \texttt{safe}(E) \wedge \phi[E/x] \\
wp^s(C_1; C_2, \phi) & = & wp^s(C_1, wp(C_2, \phi)) \\
wp^s(\texttt{if}\, B\, \texttt{then}\, C_1\, \texttt{else}\, C_2, \phi) & = & \texttt{safe}(B) \wedge (B \rightarrow wp^s(C_1, \phi) \\
 & & \wedge\, (\neg B \rightarrow wp^s(C_2, \phi) \\
wp^s(\texttt{while}\, B\, \texttt{do}\, \{\eta\} C, \phi) & = & \eta
\end{array}
$$

Algoritmo VCGen

Calcula as VC não considerando as pré-condições

$$
\begin{array}{rcl}
VC^s(\texttt{skip}, \phi) & = & \emptyset \\
VC^s(x := E, \phi) & = & \emptyset \\
VC^s(C_1; C_2, \phi) & = & VC^s(C_1, wp^s(C_2, \phi)) \cup \\
 & & VC^s(C_2, \phi) \\
VC^s(\texttt{if}\, B\, \texttt{then}\, C_1\, \texttt{else}\, C_2, \phi) & = & VC^s(C_1, \phi) \cup VC^s(C_2, \phi) \\
VC^s(\texttt{while}\, B\, \texttt{do}\, \{\eta\} C, \phi) & = & \{\theta \rightarrow \texttt{safe}(B)\} \cup \\
 & & \{(\eta \wedge B) \rightarrow wp^s(C, \eta)\} \cup \\
 & & \{(\eta \wedge \neg B) \rightarrow \phi\} \cup VC^s(C, \eta)
\end{array}
$$

Definimos VCG^s como:

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

A função `safe` para a linguagem While$^{\texttt{int}}$

$$
\begin{array}{rcl}
\texttt{safe}(n) & = & \top \\
\texttt{safe}(x) & = & \top \\
\texttt{safe}(-E) & = & \texttt{safe}(E) \\
\texttt{safe}(E_1 \odot E_2) & = & \texttt{safe}(E_1) \wedge \texttt{safe}(E_2) \\
 & & \text{com } \odot \in \{+, -, \times, =, <, \leq\} \\
\texttt{safe}(E_1 \div E_2) & = & \texttt{safe}(E_1) \wedge \texttt{safe}(E_2) \wedge E_2 \neq 0 \\
\texttt{safe}(\neg B) & = & \texttt{safe}(B) \\
\texttt{safe}(B_1 \wedge B_2) & = & \texttt{safe}(B_1) \wedge (B_1 \rightarrow \texttt{safe}(B_2)) \\
\texttt{safe}(B_1 \ \vee \ B_2) & = & \texttt{safe}(B_1) \wedge (\neg B_1 \rightarrow \texttt{safe}(B_2))
\end{array}
$$

Temos que

$$\mathcal{A}[\![E]\!]s \neq \texttt{erro} \text{ se e só se } [\![\texttt{safe}(E)]\!]s = \top.$$

Adequação de VCGen^s

Seja $\{\phi\}C\{\psi\}$ um triplo de Hoare e Γ um conjunto de asserções.

$$\Gamma \models VCG^s(\{\phi\}C\{\psi\}) \text{ se e só se } \Gamma \vdash_{\mathcal{H}_s} \{\phi\}C\{\psi\}.$$

Demonstração:

($\Rightarrow$) Por indução sobre a estrutura de C.

($\Leftarrow$) Por indução sobre a derivação $\Gamma \vdash_{\mathcal{H}_s} \{\phi\}C\{\psi\}$.

Exemplos:

$$
\begin{array}{rcl}
\texttt{safe}((x \div y) > 2) & = & \texttt{safe}(x) \wedge \texttt{safe}(y) \wedge y \neq 0 \wedge \texttt{safe}(2) \\
 & = & \top \wedge \top \wedge y \neq 0 \wedge \top \\
 & \equiv & y \neq 0
\end{array}
$$

$$
\begin{array}{rcl}
\texttt{safe}(7 > x \wedge (x \div y) > 2) & = & \texttt{safe}(7 > x) \wedge \\
 & & (\texttt{safe}(7 > x) \rightarrow \texttt{safe}((x \div y) > 2) \\
 & = & \top \wedge \top \wedge (7 > x \rightarrow (\top \wedge \top \wedge y \neq 0 \wedge \top)) \\
 & \equiv & 7 > x \rightarrow y \neq 0
\end{array}
$$

Arrays com índices limitados: linguagem While$^{\texttt{array}[N]}$

Os arrays que vimos até ao momento têm tamanho potencialmente infinito, o que não é realista numa linguagem de programação. Vamos considerar expressões do tipo $\texttt{array}[N]$, como arrays de tamanho N, que admitem índices não negativos inferiores a N. O que fazer com operações que envolvam índices fora dos limites do array? Vamos tratar estas situações como erros de execução.

Sintaxe da linguagem While$^{\texttt{array}[N]}$

Para n em **Num** e x em **Var**

$$
\begin{array}{rrl}
Exp_{\texttt{array}[N]} & A ::= & u \mid A[E \leftarrow E] \\
\\
Exp_{\texttt{int}} & E ::= & n \mid x \mid -E \mid E + E \mid E - E \\
& & \mid E \times E \mid E \div E \\
& & \mid A[E] \mid \texttt{len}(A) \\
\\
Exp_{\texttt{bool}} & B ::= & \texttt{true} \mid \texttt{false} \mid \neg B \mid E = E \\
& & \mid B < E \mid B \leq B \mid B \ \wedge \ B \mid B \ \vee \ B
\end{array}
$$

Extensão de VCGen para array[N]

Adicionamos a seguinte regra a $\mathcal{H}_g$:

$$\frac{}{\{\phi\}\, u[E] := E' \,\{\psi\}} \text{ se } \phi \rightarrow \texttt{safe}(u[E \leftarrow E']) \text{ e } \phi \rightarrow \psi[u[E \leftarrow E']/u]$$

extendemos wp^s e VC^s e `safe` da seguinte forma:

$$
\begin{array}{rcl}
wp^s(u[E] := E', \psi) & = & \texttt{safe}(A[E \leftarrow E']) \wedge \psi[u[E \leftarrow E']/u] \\
VC^s(u[E] := E', \psi) & = & \emptyset \\
\\
\texttt{safe}(u) & = & \top \\
\texttt{safe}(\texttt{len}(A)) & = & \top \\
\texttt{safe}(A[E]) & = & \texttt{safe}(A) \wedge \texttt{safe}(E) \wedge 0 \leq E \leq len(A) \\
\texttt{safe}(A[E \leftarrow E']) & = & \texttt{safe}(A) \wedge \texttt{safe}(E) \wedge \\
& & 0 \leq E \leq len(A) \wedge \texttt{safe}(E')
\end{array}
$$

Exemplos:

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

$$\begin{aligned}
\texttt{safe}(u[3 \leftarrow 10]) &= \texttt{safe}(u) \wedge \texttt{safe}(3) 0 \leq 3 < \texttt{len}(u) \wedge \texttt{safe}(10) \\
&= \top \wedge \top \wedge 0 \leq 3 < \texttt{len}(u) \wedge \top \\
&\equiv 0 \leq 3 < \texttt{len}(u)
\end{aligned}$$

Sub-rotinas

- Os programas que consideramos até agora, constituem sequências de comandos
- Um aspecto importante e desafiante em termos de verificação é o tratamento de sub-rotinas (procedimentos ou funções)
- O tratamento de sub-rotinas inclui aspectos como:
  - tratamento de chamadas recursivas (que podem levar a não terminação);
  - tratamento de passagem de parâmetros;

Procedimentos e Recursão

- `proc p` $= \texttt{C}_\texttt{p}$ é a definição de um procedimento `p`;
- o comando $C_\texttt{p}$ é o corpo do procedimento `p`;
- um novo comando `call p` invoca o procedimento, transferindo a execução para o corpo de `p`;
- uma regra de semântica definida como:

$$\frac{\langle \texttt{body(p)}, s\rangle \;\rightarrow\; s'}{\langle \texttt{call p}, s\rangle \;\rightarrow\; s'}$$

- para procedimentos não recursivos uma regra

$$\frac{\{\varphi\}\texttt{body(p)}\{\psi\}}{\{\varphi\}\texttt{call p}\{\psi\}}$$

Exemplo

Consideremos o procedimento

```
proc fact =
     f:= 1; i:= 1;
     while i <= n do {f = fact(i-1) and i <= n+1} {
       f:= f*i;
       i:= i+1;
     }
```

Pela correcção do corpo do procedimento temos:

$$\{n \geq 0 \wedge n = n_0\}\texttt{body(fact)}\{f = fact(n) \wedge n = n_0\}$$

Aplicando a regra acima temos:

$$\{n \geq 0 \wedge n = n_0\}\texttt{call fact}\{f = fact(n) \wedge n = n_0\}$$

Modularidade

- A modularidade é um aspecto importante em programação;
- Em verificação é desejável podermos reutilizar resultados de correcção;
- Por exemplo, tendo

  $$\{n \geq 0\}\texttt{fact}\{f = fact(n)\}$$

  seria desejável poder demonstrar um resultado mais fraco:

  $$\{n = 10\}\texttt{fact}\{f = fact(n)\}$$

  O que é trivialmente deduzido utilizando a regra da consequência. No entanto tendo,

  $$\{n \geq 0 \wedge n = n_0\}\texttt{fact}\{f = fact(n) \wedge n = n_0\}$$

  o mesmo já não se verifica.

O problema da adaptação

(Especificação satisfazível) Dizemos que uma especificação (ϕ, ψ) é satisfazível se existe um programa C tal que $\models \{\phi\}C\{\psi\}$.

(Completude de adaptação) Seja (ϕ, ψ) satisfazível e para qualquer programa C temos $\models \{\phi'\}C\{\psi'\}$ sempre que $\models \{\phi\}C\{\psi\}$. Um sistema de inferência de triplos de Hoare diz-se completo em relação à adaptação se e só se, para qualquer programa C a seguinte regra é derivável.

$$\frac{\{\phi\}C\{\psi\}}{\{\phi'\}C\{\psi'\}}$$

A lógica de Hoare não é completa em relação à adaptação, devido à presença de variáveis auxiliares.

Adaptação - continuação

- Podemos ver as variáveis auxiliares como variáveis quantificadas universalsemente sobre triplos de Hoare, relacionado pré e pós condições.
- Um solução foi proposta por Kleymann, considerando uma regra mais forte de consequência que distingue variáveis de programa e variáveis auxiliares
- Consideremos a regra de consequência

$$\frac{\{\phi\}C\{\psi\}}{\{\phi'\}C\{\psi'\}} \quad \text{se} \quad \phi' \to \phi \wedge \psi \to \psi'$$

- As variáveis em ψ têm que ser interpretadas no pré-estado ϕ' e devem ser quantificadas existencialmente: $n = 0 \to n \geq 0 \wedge n = n_0$, não se verifica mas sim $n = 0 \to \exists n_0.n \geq 0 \wedge n = n_0$

Adaptação - continuação

A condição mais adequada sugerida por Kleymann tem a seguinte forma

$$\phi' \to (\phi \wedge (\psi \to \psi'))$$

Sendo $\overline{y}$ as variáveis auxiliares de $\{\phi\}C\{\psi\}$, introduzimos esta quantificação existencial da seguinte forma:

$$\phi' \to \exists \overline{y_f}.(\phi[\overline{y_f}/\overline{y}] \wedge (\psi[\overline{y_f}/\overline{y}] \to \psi'))$$

Interpretamos as variáveis auxiliares em ϕ' e ψ' introduzindo novas variáveis quantificadas universalmente

$$\frac{\{\phi\}C\{\psi\}}{\{\phi'\}C\{\psi'\}} \quad \text{se} \quad \phi' \to \forall \overline{x_f}.\exists \overline{y_f}.(\phi[\overline{y_f}/\overline{y}] \wedge (\psi[\overline{y_f}/\overline{y}, \overline{x_f}/\overline{x}] \to \psi'[\overline{x_f}/\overline{x}]))$$

onde $\overline{y}$ são as variáveis auxiliares de $\{\phi\}C\{\psi\}$, $\overline{x}$ são as variáveis de programa de C, $\overline{y_f}$, $\overline{x_f}$ novas.

Adaptação - continuação

- A regra anterior destina-se a triplos de correcção total.
- Temos uma condição mais fraca para correcção parcial

$$\phi' \to (\phi \to \psi) \to \psi')$$

As variáveis auxiliares são agora quantificadas universalmente

$$\phi' \to \forall \overline{y}.(\phi \to \psi) \to \psi')$$

A regra resultante é então

$$\frac{\{\phi\}C\{\psi\}}{\{\phi'\}C\{\psi'\}} \quad \text{se} \quad \phi' \to \forall \overline{x_f}.(\forall \overline{y_f}.\phi[\overline{y_f}/\overline{y}] \to \psi[\overline{y_f}/\overline{y}, \overline{x_f}/\overline{x}]) \to \psi'[\overline{x_f}/\overline{x}])$$

onde $\overline{y}$ são as variáveis auxiliares de $\{\phi\}C\{\psi\}$, $\overline{x}$ são as variáveis de programa de C, $\overline{y_f}$, $\overline{x_f}$ novas.

Exemplo

Consideremos a seguinte asserção:

$$\{n \geq 0 \wedge n = n_0\}\texttt{fact}\{f = fact(n) \wedge n = n_0\}$$

Para derivarmos a asserção mais fraca:

$$\{n = 10\}\texttt{fact}\{f = fact(n)\}$$

Obtemos a condição auxiliar

$$\begin{aligned} n = \quad & 10 \to \forall n_f, f_f.(\forall n_{0f}.n \geq 0 \wedge n = n_{0f} \to f_f = fact(n_f) \wedge n_f = n_{0f}) \\ & \to f_f = fact(10) \end{aligned}$$

Procedimentos recursivos

- Em procedimentos recursivos, `body(p)` pode conter chamadas a `call p`
- A aplicação da regra apresentada anteriormente, poderia levar à construção de derivações infinitas.
- Consideremos a seguinte regra proposta por Hoare

$$\frac{\begin{array}{c}[\{\varphi\}]\texttt{call } p\{\psi\}] \\ \vdots \\ \{\varphi\}\texttt{body}(p)\{\psi\}\end{array}}{\{\varphi\}\texttt{call p}\{\psi\}}$$

- Consiste na interpretação axiomática da noção de ponto-fixo.

Exemplo

Consideremos o procedimento

```
proc factr =
     if n=0 then f:=1
      else {
        n:=n-1;
        call factr;
        n:=n+1;
        f:=n*f;
     }
```

$$\{n \geq 0 \wedge n = n_0\}\texttt{call factr}\{f = fact(n) \wedge n = n_0\}$$

Podemos derivar esta asserção com base na regra da página anterior e considerando a regra da consequência com variáveis auxiliares como apresentado no problema de adaptação.

Chamadas a procedimentos em $\mathcal{H}_g$

Temos de considerar o problema de adaptação na ausência da regra da consequência

$$\frac{\{\phi\}\texttt{body(p)}\{\psi\}}{\{\phi'\}\texttt{call p}\{\psi'\}} \text{ se } \phi' \rightarrow \forall \overline{x_f}.(\forall \overline{y_f}.\phi[\overline{y_f}/\overline{y}] \rightarrow \psi[\overline{y_f}/\overline{y}, \overline{x_f}/\overline{x}]) \rightarrow \psi'[\overline{x_f}/\overline{x}])$$

onde $\overline{y}$ são as variáveis auxiliares de $\{\phi\}\texttt{body(p)}\{\psi\}$, $\overline{x}$ são as variáveis de programa de `body(p)`, $\overline{y_f}$, $\overline{x_f}$ novas. A ideia é de que provando que o corpo do parâmetro p é correcto relativamente a (ϕ, ψ), então esta especificação deve ser suficientemente forte para deduzir especificações mais fracas.