Cálculo de Correção parcial $\mathcal{H}$

$[skip_p\ ]$

$$\{\varphi\}\,\texttt{skip}\,\{\varphi\}$$

$[ass_p\ ]$

$$\{\varphi[E/x]\}\,x \leftarrow E\,\{\varphi\}$$

$[comp_p\ ]$

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

$[if_p\ ]$

$$\frac{\{\varphi\ \wedge\ B\}\,C_1\,\{\psi\} \qquad \{\varphi\ \wedge\ \neg B\}\,C_2\,\{\psi\}}{\{\varphi\}\,\texttt{if}\,B\,\texttt{then}\,C_1\,\texttt{else}\,C_2\,\{\psi\}}$$

$[while_p\ ]$

$$\frac{\{\psi\ \wedge\ B\}\,C\,\{\psi\}}{\{\psi\}\,\texttt{while}\,B\,\texttt{do}\,C\,\{\psi\ \wedge\ \neg B\}}$$

$[cons_p\ ]$

$$\frac{\vdash \varphi' \rightarrow \varphi \qquad \{\varphi\}\,C\,\{\psi\} \qquad \vdash \psi \rightarrow \psi'}{\{\varphi'\}\,C\,\{\psi'\}}$$

Mecanização da construção de derivações na lógica de Hoare

De um modo geral, dado um triplo de Hoare ($\{\varphi\}C\{\psi\}$) aplicamos as regras a partir da conclusão, assumindo que as condições auxiliares se verificam.

- Se todas as condições auxiliares se verificarem então construímos uma demonstração;
- Se alguma das condições auxiliares não se verifica, a árvore construída não constitui uma dedução válida, mas será possível construir uma outra árvore que o seja?

Existe uma estratégia para construir as árvores de forma a poder concluir (caso algumas das condições auxiliares não se verifique) que não existe uma derivação para o triplo dado.

Mecanização da lógica de Hoare

A maior parte das regras do cálculo de Hoare têm a *propriedade de sub-fórmula:*

> *todas as asserções que ocorrem nas premissas de uma regra também ocorrem na sua conclusão.*

As excepções são:

- A regra *comp*, que requer uma condição intermédia;
- A regra *cons*, onde a pré-condição e a pós-condição têm que ser "adivinhadas".

Outra propriedade desejável é a falta de ambiguidade na escolha das regras:

- A regra *cons*, pode ser aplicada para qualquer triplo de Hoare.

Versão da lógica de Hoare sem *cons*: sistema $\mathcal{H}_g$

$$\frac{}{\{\varphi\}\,\mathsf{skip}\,\{\psi\}} \text{ se } \models \varphi \to \psi$$

$$\frac{}{\{\varphi\}\,x \leftarrow E\,\{\psi\}} \text{ se } \models \varphi \to \psi[E/x]$$

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

$$\frac{\{\varphi \ \wedge \ B\}\,C_1\,\{\psi\} \qquad \{\varphi \ \wedge \ \neg B\}\,C_2\,\{\psi\}}{\{\varphi\}\,\mathtt{if}\,B\,\mathtt{then}\,C_1\,\mathtt{else}\,C_2\,\{\psi\}}$$

$$\frac{\{\eta \ \wedge \ B\}\,C\,\{\eta\}}{\{\varphi\}\,\mathtt{while}\,B\,\mathtt{do}\,\{\eta\}C\,\{\psi\}} \text{ se } \models \varphi \to \eta \text{ e } \models \eta \ \wedge \ \neg B \to \psi$$

Sistema $\mathcal{H}_g$

É fácil de demonstrar que a regra *cons* é derivável em $\mathcal{H}_g$.

Lema 23.1. *Se* $\Gamma \vdash_{\mathcal{H}_g} \{\varphi\}C\{\psi\}$ *e* $\models \varphi' \to \varphi$, $\models \psi \to \psi'$, *então* $\Gamma \vdash_{\mathcal{H}_g} \{\varphi'\}C\{\psi'\}$.

Demonstração: Por indução sobre a derivação $\Gamma \vdash_{\mathcal{H}_g} \{\psi\}C\{\varphi\}$. Vamos ver os casos para o `skip` e para a sequência.

- Para $C \equiv \texttt{skip}$, temos $\Gamma \vdash_{\mathcal{H}_g} \{\varphi\}\texttt{skip}\{\psi\}$, se $\models \varphi \to \psi$. Temos $\models \varphi' \to \varphi$, $\models \varphi \to \psi$ e $\models \psi \to \psi'$, logo $\models \varphi' \to \psi'$, o que significa que temos $\Gamma \vdash_{\mathcal{H}_g} \{\varphi'\}\texttt{skip}\{\psi'\}$.
- Para $C \equiv C_1; C_2$, temos $\Gamma \vdash_{\mathcal{H}_g} \{\varphi\}C_1; C_2\{\psi\}$, se $\Gamma \vdash_{\mathcal{H}_g} \{\varphi\}C_1\{\eta\}$ e $\Gamma \vdash_{\mathcal{H}_g} \{\eta\}C_2\{\psi\}$. Mas então por H.I. temos $\Gamma \vdash_{\mathcal{H}_g} \{\varphi'\}C_1\{\eta\}$ (uma vez que $\models \varphi' \to \varphi$ e $\models \eta \to \eta$) e $\Gamma \vdash_{\mathcal{H}_g} \{\eta\}C_2\{\psi'\}$ (uma vez que $\models \eta \to \eta$ e $\models \psi \to \psi'$), logo $\Gamma \vdash_{\mathcal{H}_g} \{\varphi'\}C_1; C_2\{\psi'\}$.

Exercício 23.1. *Completa a demonstração anterior.*

Equivalência $\mathcal{H}$ e $\mathcal{H}_g$

$\Gamma \vdash_{\mathcal{H}} \{\varphi\}C\{\psi\}$ se e só se $\Gamma \vdash_{\mathcal{H}_g} \{\varphi\}C\{\psi\}$

($\Rightarrow$) Por indução sobre a derivação $\Gamma \vdash_{\mathcal{H}} \{\psi\}C\{\varphi\}$, usando o lema anterior. Vamos ver os casos para atribuição e para a regra da consequência.

- Temos $\Gamma \vdash_{\mathcal{H}} \{\varphi[E/x]\}x \leftarrow E\{\varphi\}$ e $\models \varphi[E/x] \to \varphi[E/x]$, logo $\Gamma \vdash_{\mathcal{H}_g} \{\varphi[E/x]\}x \leftarrow E\{\varphi\}$
- Pela regra da consequência temos $\Gamma \vdash_{\mathcal{H}} \{\varphi\}C\{\psi\}$, se $\Gamma \vdash_{\mathcal{H}} \{\varphi'\}C\{\psi'\}$ e $\models \varphi \to \varphi'$, $\models \psi' \to \psi$. Por H.I. temos $\Gamma \vdash_{\mathcal{H}_g} \{\varphi'\}C\{\psi'\}$, logo pelo lema anterior temos $\Gamma \vdash_{\mathcal{H}_g} \{\varphi\}C\{\psi\}$.

($\Leftarrow$) Por indução sobre a derivação $\Gamma \vdash_{\mathcal{H}_g} \{\psi\}C\{\varphi\}$. Vamos ver os casos para a atribuição e para o condicional.

- Temos $\Gamma \vdash_{\mathcal{H}_g} \{\psi\}x \leftarrow E\{\varphi\}$ se $\models \psi \to \varphi[E/x]$. Como $\Gamma \vdash_{\mathcal{H}} \{\varphi[E/x]\}x \leftarrow E\{\varphi\}$ e $\models \psi \to \varphi[E/x]$ e $\models \psi \to \psi$, então pela regra da consequência, temos $\Gamma \vdash_{\mathcal{H}} \{\psi\}x \leftarrow E\{\varphi\}$.
- Temos $\Gamma \vdash_{\mathcal{H}_g} \{\psi\}\texttt{if}\, B\, \texttt{then}\, C_1\, \texttt{else}\, C_2\, \{\varphi\}$, se $\Gamma \vdash_{\mathcal{H}_g} \{\psi \wedge B\}C_1\{\varphi\}$ e $\Gamma \vdash_{\mathcal{H}_g} \{\psi \wedge \neg B\}C_2\{\varphi\}$. Por H.I. $\Gamma \vdash_{\mathcal{H}} \{\psi \wedge B\}C_1\{\varphi\}$ e $\Gamma \vdash_{\mathcal{H}} \{\psi \wedge \neg B\}C_2\{\varphi\}$, logo $\Gamma \vdash_{\mathcal{H}} \{\psi\}\texttt{if}\, B\, \texttt{then}\, C_1\, \texttt{else}\, C_2\, \{\varphi\}$

Exercício 23.2. *Completa a demonstração anterior.*

Pós e Contras

Vantagens de $\mathcal{H}_g$:

- Eliminamos a ambiguidade provocada pela regra *cons*.
- Eliminamos uma das regras sem a propriedade de sub-fórmula.

No entanto, ainda é necessário "adivinhar" pré-condições intermédias para *comp*.

Desvantagens de $\mathcal{H}_g$:

- Perdemos alguma capacidade de re-utilizar resultados de correcção (veremos mais adiante como resolver esta questão).

A estratégia de pré-condição mais fraca

Queremos construir uma derivação para um triplo de Hoare $\{\varphi\}C\{\psi\}$, onde φ pode ou não ser conhecido (nesse caso escrevemos $\{?\}C\{\psi\}$).

1. Se φ for conhecido, então aplicamos a única regra possível de $\mathcal{H}_g$. Se C for $C_1; C_2$, então construímos uma sub-derivação da forma $\{?\}C_2\{\psi\}$. Eventualmente quando concluírmos esta derivação podemos prosseguir com $\{\varphi\}C_1\{\theta\}$, com θ obtido da sub-derivação anterior.
2. Se φ é desconhecido, a construção procede da mesma forma, excepto que no caso das regras skip, atribuição e ciclos, com uma condição auxiliar $\varphi \to \theta$, tomamos a pré-condição φ como θ.

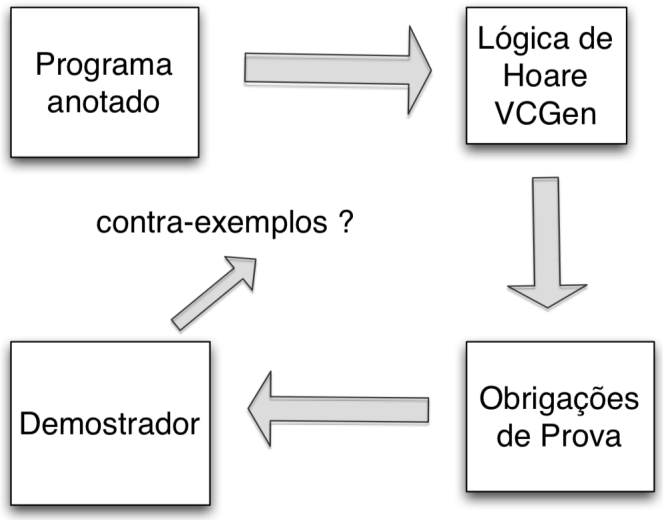
Uma Arquitectura para Verificação de Programas

Dado um triplo de Hoare $\{\varphi\}C\{\psi\}$ e uma teoria $\mathcal{T}$:

1. Aplicamos os princípios apresentados anteriormente para construir uma derivação com conclusão $\{\varphi\}C\{\psi\}$, assumindo que todas as condições auxiliares geradas no processo se verificam.
2. Cada fórmula de primeira ordem gerada como condição auxiliar (chamada neste contexto de *condição de verificação* (VC)) tem que ser verificada numa ferramenta de prova.
3. Se todas as condições de verificação são classificadas como $\mathcal{T}$-válidas, então $\mathcal{T} \vdash_{\mathcal{H}_g} \{\varphi\}C\{\psi\}$.

Nota: como não existe ambiguidade na construção das árvores, podemos eliminar essa parte do processo e simplesmente gerar as VC usando um *Gerador de condições de verificação*(VCGen).

Duas fases para a verificação



Um algoritmo VCGen: cálculo das pré-condições mais fracas (wp)

Dado um programa C e uma pós-condição ψ, podemos calcular $wp(C,\psi)$ tal que $\{wp(C,\psi)\}C\{\psi\}$ é válida e se $\{\varphi\}C\{\psi\}$ é válida para algum φ então $\varphi \rightarrow wp(C,\psi)$.

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

Algoritmo VCGen

Primeiro calcula as VC não considerando as pré-condições

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

A pré-condição é tomada em consideração:

$$VCG(\{\varphi\}C\{\psi\}) = \{\varphi \to wp(C,\psi)\} \cup VC(C,\psi)$$

Propriedades de wp e VCG

Dado um comando C e uma asserção ψ se $\Gamma \vdash_{\mathcal{H}_g} \{\varphi\}C\{\psi\}$, para alguma pré-condição φ, então

1. $\Gamma \vdash_{\mathcal{H}_g} \{wp(C,\psi)\}C\{\psi\}$
2. $\Gamma \models \varphi \to wp(C,\psi)$

Demonstração: Por indução sobre C. Vamos ver os casos de `skip` e `while`.

- Para $C \equiv$ `skip`, temos $\Gamma \vdash_{\mathcal{H}_g} \{\varphi\}\texttt{skip}\{\psi\}$ se $\models \varphi \to \psi$. Note-se que $wp(\texttt{skip},\psi) = \psi$.
  1. Trivialmente temos $\Gamma \vdash_{\mathcal{H}_g} \{\psi\}\texttt{skip}\{\psi\}$, uma vez que $\models \psi \to \psi$.
  2. Por hipótese temos $\Gamma \models \varphi \to \psi = wp(\texttt{skip},\psi)$.
- $C \equiv$ `while`, temos $\Gamma \vdash_{\mathcal{H}_g} \{\psi\}\,\texttt{while}\,B\,\texttt{do}\,\{\eta\}C\,\{\psi\}$ se $\Gamma \vdash_{\mathcal{H}_g} \{\eta \wedge B\}C\{\eta\}$ e $\models \psi \to \eta$, $\models \eta \wedge \neg B \to \psi$. Note-se que $wp(\texttt{while}\,B\,\texttt{do}\,\{\eta\}C,\psi) = \eta$
  1. Como $\models \eta \to \eta$, e por hipótese $\models \eta \wedge \neg B \to \psi$ e $\Gamma \vdash_{\mathcal{H}_g} \{\eta \wedge B\}C\{\eta\}$, então $\Gamma \vdash_{\mathcal{H}_g} \{\eta\}\,\texttt{while}\,B\,\texttt{do}\,\{\eta\}C\,\{\psi\}$
  2. Por hipótese temos $\Gamma \models \varphi \to \eta = wp(\texttt{while}\,B\,\texttt{do}\,\{\eta\}C\,\psi)$.

Exercício 23.3. *Completa a demonstração anterior.*

Teorema 23.1 (Adequação de VCGen)**.** *Seja* $\{\varphi\}C\{\psi\}$ *um triplo de Hoare e* Γ *um conjunto de asserções.*

$$\Gamma \models VCG(\{\varphi\}C\{\psi\}) \text{ se e só se } \Gamma \vdash_{\mathcal{H}_g} \{\varphi\}C\{\psi\}.$$

($\Rightarrow$) Por indução sobre a derivação C. Vamos ver os casos para atribuição e para a regra da sequência.

- Para $C \equiv x \leftarrow E$, temos $VCG(\{\varphi\}X \leftarrow E\{\psi\}) = \{\varphi \to wp(X \leftarrow E,\psi)\} \cup VC(x \leftarrow E,\psi) = \{\varphi \to \psi[E/x]\}$. Se $\Gamma \models \varphi \to \psi[E/x]$, então pela regra da atribuição $\Gamma \vdash_{\mathcal{H}_g} \{\varphi\}C\{\psi\}$.

Adequação do VCG

- Para $C \equiv C_1; C_2$, temos

$$\begin{array}{rcl} VCG(\{\varphi\}C_1; C_2\{\psi\}) & = & \{\varphi \to wp(C_1; C_2, \psi)\} \cup VC(C_1; C_2, \psi) \\ & = & \{\varphi \to wp(C_1, wp(C_2, \psi))\} \\ & & \cup\, VC(C_1, wp(C_2, \psi)) \cup VC(C_2, \psi). \end{array}$$

Seja $\eta = wp(C_2, \psi)$. Como

$$\Gamma \models \varphi \to wp(C_1, \eta) \cup VC(C_1, \eta) = VCG(\{\varphi\}C_1\{\eta\},$$

por I.H. $\Gamma \vdash_{\mathcal{H}_g} \{\varphi\}C_1\{\eta\}$.
Também $\Gamma \models \eta \to \eta \cup VC(C_2, \psi) = VCG(\{\eta\}C_2\{\psi\})$, por I.H. $\Gamma \vdash_{\mathcal{H}_g} \{\eta\}C_2\{\psi\}$, logo $\Gamma \vdash_{\mathcal{H}_g} \{\varphi\}C_1; C_2\{\psi\}$

Adequação do VCG

($\Leftarrow$) Por indução sobre a derivação $\Gamma \vdash_{\mathcal{H}_g} \{\psi\}C\{\varphi\}$. Vamos ver os casos para o `skip` e para o condicional.

- $\Gamma \vdash_{\mathcal{H}_g} \{\varphi\}\texttt{skip}\{\psi\}$, se $\Gamma \models \varphi \to \psi = VCG(\{\varphi\}\texttt{skip}\{\psi\})$.

Adequação do VCG

- $\Gamma \vdash_{\mathcal{H}_g} \{\varphi\}\texttt{if}\, B\, \texttt{then}\, C_1\, \texttt{else}\, C_2, \{\psi\}$ se $\Gamma \vdash_{\mathcal{H}_g} \{\varphi \wedge B\}C_1\{\psi\}$ e $\Gamma \vdash_{\mathcal{H}_g} \{\varphi \wedge \neg B\}C_2\{\psi\}$. Por H.I.

$$\Gamma \models VCG(\{\varphi \wedge B\}C_1\{\psi\}) = \{(\varphi \wedge B) \to wp(C_1, \psi)\} \cup VC(C_1, \psi)$$

e

$$\Gamma \models VCG(\{\varphi \wedge \neg B\}C_2\{\psi\}) = \{(\varphi \wedge \neg B) \to wp(C_2, \psi)\} \cup VC(C_2, \psi).$$

Note-se que,

$$wp(\texttt{if}\, B\, \texttt{then}\, C_1\, \texttt{else}\, C_2, \psi) = B \to wp(C_1, \psi) \wedge \neg B \to wp(C_2, \psi)\},$$

logo

$$\Gamma \models \{\varphi \to wp(\texttt{if}\, B\, \texttt{then}\, C_1\, \texttt{else}\, C_2, \psi)\}.$$

Logo $\Gamma \models \{\varphi \to wp(\texttt{if}\, B\, \texttt{then}\, C_1\, \texttt{else}\, C_2, \psi)\} \cup VC(C_1, \psi) \cup VC(C_2, \psi) = VCG(\{\varphi\}\texttt{if}\, B\, \texttt{then}\, C_1\, \texttt{else}\, C_2\{\psi\})$.

Exemplo

Seja fact o seguinte programa:

$f \leftarrow 1; i \leftarrow 1;$
while $i \leq n$ **do**
Ensure: $f = (i-1)! \wedge i \leq n+1$
$f \leftarrow f * i;$
$i \leftarrow i + 1;$

Vamos calcular

$$\mathrm{VCG}(\{n \geq 0\}\mathsf{fact}\{f = n!\})$$

com $\theta = f = (i-1)! \wedge i \leq n+1$ e $C_w = f \leftarrow f * i; i \leftarrow i + 1$

$$\begin{array}{ll}
 & VC(\texttt{fact}, f = n!) \\
= & VC(f \leftarrow 1; i \leftarrow 1, wp(\textbf{while } i \leq n \textbf{ do}\{\theta\}C_w, f = n!)) \\
 & \cup VC(\textbf{while } i \leq n \texttt{ do}\{\theta\}C_w, f = n!) \\
= & VC(f \leftarrow 1; i \leftarrow 1, \theta) \cup \{\theta \wedge i \leq n \rightarrow wp(C_w, \theta)\} \\
 & \cup \{\theta \wedge i > n \rightarrow f = n!\} \cup VC(C_w, \theta) \\
= & VC(f \leftarrow 1, wp(i \leftarrow 1, \theta)) \cup VC(i \leftarrow 1, \theta) \\
 & \cup \{f = (i-1)! \wedge i \leq n+1 \wedge i \leq n \rightarrow wp(f \leftarrow f * i; i \leftarrow i+1, \theta)\} \\
 & \cup \{f = (i-1)! \wedge i \leq n+1 \wedge i > n \rightarrow f = n!\} \\
 & \cup VC(f = f * i, wp(i \leftarrow i+1, \theta)) \cup VC(i \leftarrow i+1, \theta) \\
= & \emptyset \cup \emptyset \cup \{f = (i-1)! \wedge i \leq n+1 \wedge i \leq n \\
 & \qquad \rightarrow wp(f \leftarrow f * i, f = (i+1-1)! \wedge i + 1 \leq n+1)\} \\
 & \cup \{f = (i-1)! \wedge i \leq n+1 \wedge i \leq n \rightarrow f = n!\} \cup \emptyset \cup \emptyset \\
= & \{f = (i-1)! \wedge i \leq n+1 \wedge i \leq n \rightarrow f * i = (i+1-1)! \wedge i + 1 \leq n+1, \\
 & f = (i-1)! \wedge i \leq n+1 \wedge i \leq n \rightarrow f = n!\}
\end{array}$$

$$\begin{array}{ll}
 & VCG(\{n \geq 0\}\mathsf{fact}\{f = n!\}) \\
= & \{n \geq 0 \rightarrow wp(\mathsf{fact}, f = n!)\} \cup VC(\mathsf{fact}, f = n!) \\
= & \{n \geq 0 \rightarrow wp(f \leftarrow 1; i \leftarrow 1; wp(\textbf{while } i \leq n \textbf{ do}\{\theta\}C_w, f = n!), \\
 & f = (i-1)! \wedge i \leq n+1 \wedge i \leq n \rightarrow f * i = (i+1-1)! \wedge i + 1 \leq n+1, \\
 & f = (i-1)! \wedge i \leq n+1 \wedge i \leq n \rightarrow f = n!\} \\
= & \{n \geq 0 \rightarrow wp(f \leftarrow 1; i \leftarrow 1; \theta), \\
 & f = (i-1)! \wedge i \leq n+1 \wedge i \leq n \rightarrow f * i = (i+1-1)! \wedge i + 1 \leq n+1, \\
 & f = (i-1)! \wedge i \leq n+1 \wedge i \leq n \rightarrow f = n!\}
\end{array}$$

Chegamos às seguintes obrigações de prova:

1. $n \geq 0 \rightarrow 1 = (1-1)! \wedge 1 \leq n+1$
2. $f = (i-1)! \wedge i \leq n+1 \wedge i \leq n \rightarrow f * i = (i+1-1)! \wedge i+1 \leq n+1)$
3. $f = (i-1)! \wedge i \leq n+1 \wedge i \leq n \rightarrow f = n!$

Arrays (*aliases*)

Se tivermos uma variável indexada $a[]$ a regra da atribuição não pode ser directamente aplicada a um elemento:

$$\{\varphi[E_2/a[E_1]]\}a[E_1] := E_2\{\varphi\}$$

porque as modificações em $a[E_1]$ podem alterar outras referências a a que ocorram em φ ou em E_2.

Por exemplo, o triplo $\{a[j] > 100\}a[i] := 10\{a[j] > 100\}$ seria derivado pelo axioma acima, mas não é válido: por exemplo se for avaliado num estado em que $i = j$.

A solução de T. Hoare foi considerar os *Arrays* como monolíticos, e uma atribuição

$$a := a[E_1 \leftarrow E_2]$$

que significa que a passou a ser um *array* igual ao anterior mas em que a posição E_1 passou a valer E_2.

Assim no caso anterior o valor de $a[i]$ e de $a[j]$ mudam os dois...porque muda o próprio array...

Arrays

Lógica de Hoare

$[array_p\ ]$

$$\{\psi[a[E_1 \leftarrow E_2]/a]\}\, a[E_1] := E_2\, \{\psi\}$$

onde E_1 é um inteiro.

E onde

$$\begin{array}{lcl} a[E_1 \leftarrow E_2][E_1] & = & E_2 \\ a[E_1 \leftarrow E_2][E_3] & = & a[E_3] \text{ se } E_3 \neq E_1. \end{array}$$

Exemplo

$$
\begin{array}{l}
\vdash_p \{a[x] = x \ \wedge \ a[y] = y\} \\
\quad r := a[x]; \\
\quad a[x] := a[y]; \\
\quad a[y] := r \\
\quad \{a[x] = y \ \wedge \ a[y] = x\}
\end{array}
$$

Nota: Actualmente nas implementações não se usa esta técnica porque é computacionalmente muito ineficiente!...

Condições de verificação para programas com arrays

Seja maxarray o seguinte programa anotado:

$max \leftarrow 0;$
$i \leftarrow 1;$
while $i < size$ **do** $\{1 \leq i \leq size \wedge 0 \leq max < i \wedge \forall a.0 \leq a < i \rightarrow u[a] \leq u[max]\}$
  if $u[i] > u[max]$ **then**
    $max \leftarrow i$
  else
    skip;
  $i \leftarrow i + 1$

Condições de verificação para programas com arrays

Queremos verificar que

$\{size \geq 1\}$ maxarray $\{0 \leq max < size \wedge \forall a.0 \leq a < size \rightarrow u[a] \leq u[max]\}$

Assumimos

$$
\begin{array}{rcl}
\eta & = & 1 \leq i \leq size \wedge 0 \leq max < i \wedge \forall a.0 \leq a < i \rightarrow u[a] \leq u[max] \\
C & = & \textbf{if } u[i] \ > \ u[max] \textbf{ then } max \leftarrow i \textbf{ else skip}; i \leftarrow i + 1 \\
\psi & = & 0 \leq max < size \wedge \forall a.0 \leq a < i \rightarrow u[a] \leq u[max]
\end{array}
$$

Extensão de VCGen para arrays

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

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

extendemos wp e VC da seguinte forma:

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

por exemplo:

$$wp(u[i] \leftarrow 10, u[j] > 100) = u[i \leftarrow 10][j] > 100$$

Exercícios

Usando o algoritmo VCGen calcula:

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. $VCG(\{i = 70\}u[i] \leftarrow 10\{u[i] = 10\})$

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

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

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

Semântica das expressões boleanas

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

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

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

$$
\begin{aligned}
&\langle \texttt{skip}, s\rangle \ \longrightarrow \ s \\
&\langle x \leftarrow E, s\rangle \ \longrightarrow \ \begin{cases} s[\mathcal{A}[\![E]\!]s/x] & \text{se } \mathcal{A}[\![E]\!]s \neq \mathbf{erro} \\ \mathbf{erro} & \text{caso contrário} \end{cases} \\
&\frac{\langle C_1, s\rangle \ \longrightarrow \ \mathbf{erro}}{\langle C_1; C_2, s\rangle \ \longrightarrow \ \mathbf{erro}} \\
&\frac{\langle C_1, s\rangle \ \longrightarrow \ s' \ , \ \langle C_2, s'\rangle \ \longrightarrow \ s''}{\langle C_1; C_2, s\rangle \ \longrightarrow \ s''} \text{ se } s' \neq \mathbf{erro} \\
&\langle \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2, s\rangle \ \longrightarrow \ \mathbf{erro} \text{ se } \mathcal{B}[\![B]\!]s = \mathbf{erro}
\end{aligned}
$$

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

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

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

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

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

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

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

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

$$\frac{}{\{\varphi\}\,\mathsf{skip}\,\{\psi\}} \text{ se } \varphi \to \psi$$

$$\frac{}{\{\varphi\}\, x \leftarrow E\,\{\psi\}} \text{ se } \varphi \to \mathsf{safe}(E) \text{ e } \varphi \to \psi[E/x]$$

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

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

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

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

$$\begin{aligned}
wp^s(\texttt{skip}, \varphi) &= \varphi \\
wp^s(x \leftarrow E, \varphi) &= \mathsf{safe}(E) \wedge \varphi[E/x] \\
wp^s(C_1; C_2, \varphi) &= wp^s(C_1, wp(C_2, \varphi)) \\
wp^s(\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2, \varphi) &= \mathsf{safe}(B) \wedge (B \to wp^s(C_1, \varphi) \\
&\quad \wedge (\neg B \to wp^s(C_2, \varphi) \\
wp^s(\texttt{while } B \texttt{ do } \{\eta\} C, \varphi) &= \eta
\end{aligned}$$

Algoritmo VCGen

Calcula as VC não considerando as pré-condições

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

Definimos VCG^s como:

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

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

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

Temos que

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

Adequação de VCGen^s

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

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

Demonstração:

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

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

Exemplos:

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

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

Ferramentas dee verificação dedutiva de programas

- Anotação de programas
- Geração automática de obrigações de prova
- Utilização de demonstradores automáticos ou interactivos para a prova: resolutores SMT ou assistentes de demonstração (Coq)

Why: VCGen multi-linguagem, multi-demonstrador

why(3) é uma ferramenta que produz condições de verificação a partir de programas anotados:

- podem ser em Java, ML ou C
- os demonstradores podem ser

  automáticos: Simplify, Yices, Alt-ergo, CVC3, Z3 etc.

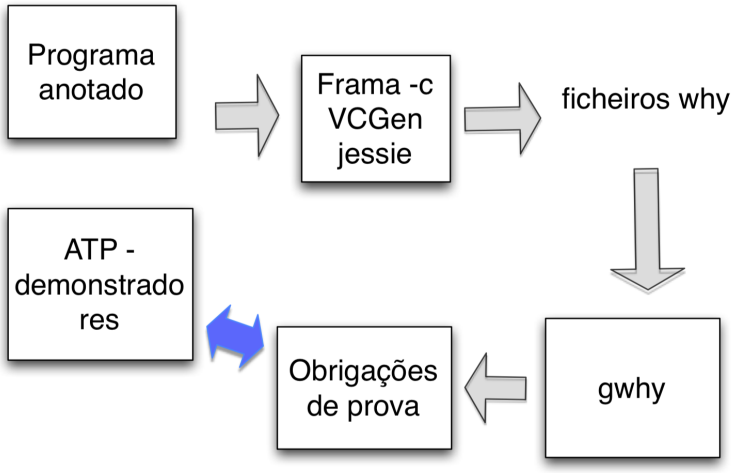
  interactivos: Coq, Isabelle, Mizar, etc.
- Satisfaz o critério de Bruijn: uma vez demonstradas as obrigações de prova, uma demonstração de que o programa satisfaz as especificações é construída e verificado o seu tipo, automaticamente.

- why traduz o programa anotado num programa funcional anotado que incluí o modelo de memória da linguagem imperativa/objectos.
- `why.lri.fr` e `why3.lri.fr`

Frama-C

- Ambiente integrado para a análise de programas fonte em C. `frama-c.cea.fr`
- Inclui vários *plugins* entre eles
  - *Jessie* o VCGen para C baseado no *why*.
  - *WP* um VCGen nativo com o demonstrador integrado `Alt-ergo`
- Os programas são anotados com a linguagem ACSL (inspirada no JML) mas orientada para:
  - Verificação estática
  - Verificação dedutiva

Frama-C com ATP



Um linguagem de especificação

- ACSL (ANSI/ISO **C** Specification Language).
- Uma linguagem de anotação para programas em **C**.
  - Cada função em **C** é anotada com uma especificação ACSL - o contracto da função.

- A verificação de um programa consistindo num conjunto de funções é modular: assumindo que o resto das funções é correcto verificamos a correcção de uma função relativamente ao seu contracto.
- Um programa é correcto se todas as suas funções o forem.
- As funções podem ser anotadas com condições *frame* e os ciclos com variantes e invariantes.

ACSL:linguagem de especificação para o ANSI C

Anotações entre /*@ */ ou //@

Expressões lógicas: expressões C com alguns constructores novos: com um *backslash* antes: `forall, exists`, `c?e1:e2`

Tipos: de C ou lógicos: `integer`, `real` and `boolean`. Há coersão para os tipos de C.

Contractos funcionais: `requires`,`ensures`,`assigns`, `assumes`,`behavior`,`decreases`, `terminates`

Etiquetas lógicas: com um *backslash* antes: `old`,`result`, `at(e,id)`

Anotações de comandos: `loop invariant`, `assert`, `loop variant`

WP ou Jessie

Permitem a verificação de

- Propriedades de segurança: limites de valores (overflows, etc); integridade das referências a variáveis indexadas e apontadores
- Propriedades funcionais dos programas

- `frama-c -jessie max.c`
- (* `frama-c -jessie -jessie-atp alt-ergo max.c` *)
- (* `frama-c-gui -jessie max.c` *)

ou

- `frama-c -wp max.c`
- `frama-c-gui -wp max.c`

Linguagem de especificação ACSL

As condições de verificação são agrupadas em

Segurança

Funcional Comportamento por omissão

Comportamentos normais `behavior`

Comportamentos definidos pelo utilizador

Contratos funcionais

```
  /*@ requires P ;
@ behavior b1 :
@ assumes A1 ;
@ requires R1 ;
@ assigns L1 ;
@ ensures E1 ;
@ behavior b2 :
@ assumes A2 ;
@ requires R2 ;
@ assigns L2 ;
@ ensures E2 ;
@*/
```

Contratos funcionais

A semântica é:

- A chamada da função tem de ser feita num estado tal que P & & $(A_1 \Rightarrow R_1)$ & & $(A_2 \Rightarrow R_2)$ se verifica.
- A função retorna um estado tal que $\backslash old(A_i) \Rightarrow E_i$ para cada i.
- para cada i, se a função é chamada num estado em que A_i se verifica, toda a memória alocada nesse estado e que não está em L_i fica alocada e com os valores que tinha no estado final.

Verificação de Segurança

Memória validade dos acessos a memória alocada

Inteiros verifica a não existência de *overflows* e que as operações aritméticas são executadas correctamente (não existência de divisão por zero).

Terminação

`pragma JessieIntegerModel(math)` (ou `exact`) com este pragma só é necessário verificar a memória porque supõe-se que os inteiros têm precisão infinita.

Outros valores:

- `strict` com precisão limitada e para cada operação é nec. garantir que não há overflow
- `modulo` modela os processadores reais com operações modulo 2^n.

Exemplo: pesquisa binária

```
//@ requires n >= 0 && \valid(t+(0..n-1));
int binary_search(int* t, int n, int v) {
int l = 0, u = n-1, p = -1;
//@ loop invariant 0 <= l && u <= n-1;
```

É necessário (pré-condição) que `t` seja válido entre 0 e $n-1$ e que $n \geq 0$.No invariante de ciclo `l` e `u` devem ser estar entre 0 e $n-1$.

Segurança de *overflow* de inteiros

Sem o pragma anterior (corresponde a `strict`)

é necessário garantir que $l+u$ não ultrapassa os limites de um inteiro de 32 bits.

Neste caso basta mudar a instrução:

`int m = (l + u) / 2;`

para

`int m = l + (u - l) / 2;`

Verificação funcional

Exemplo: pesquisa binária

O resultado tem de ser um índice entre os definidos ou -1:

`@  ensures -1 <= \result <= n-1;`

Para demonstrar isto é necessário fortalecer o invariante de ciclo:

`@ 0 <= l && u <= n -1 &&  -1 <= p <= n-1;` Neste caso deve-se ainda dividir o comportamento em dois: um quando há sucesso (v está em t) outro quando há falhanço (v não está em t).

`maxarray` em C com anotações ACSL

```
int size, u[],max;

/*@ requires size >=1 && \valid(u+(0..size-1));
  @ ensures 0 <= max < size &&
  @    (\forall int a; 0 <= a < size ==> u[a] <= u [max]);
  @*/
void maxarray() {
  int i = 1;
  max = 0;

  /*@ loop invariant
    @    1 <= i <= size && 0 <= max < i &&
    @    (\forall int a; 0 <= a < i ==> u[a] <= u[max]);
    @ loop variant
    @    size-i;
    @*/
  while (i < size) {
    if (u[i] > u[max]) max = i;
    i = i+1;
  }
}
```

Exemplo - maxarray

- Tanto o input `(size,u)` como o output `(max)` são variáveis globais.
- As linhas (comentadas) que antecedem a função, contêm:
  - a pré-condição da função assinalada por `require`
  - a pós-condição da função assinalada por `ensures`
- As linhas (comentadas) que antecedem o ciclo while, contêm o variante e invariante do ciclo.
- A condição `valid_range` é usada para definir uma pré-condição de segurança.

Casos de estudo industriais com Frama-C

- Dassault Aviation: `https://www.dassault-aviation.com/fr/`
- ETCS: European Train Control System: `http://openetcs.org`
- e em geral na certificação de veículos autónomos...no futuro...