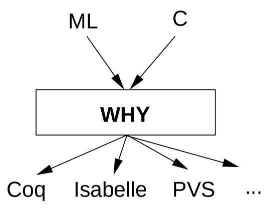
Vorlesung vom 27.06.06: Werkzeugunterstützte Verifikation

Wo sind wir?

- Hoare-Kalkül
- Jenseits von Hoare:
  - Einführung
  - Reasoning in JML — Ausnahmen
  - *Werkzeugunterstützung*
- Korrekte Programmentwicklung

WHY

- WHY: a *Multi-Language Multi-Prover Verification Condition Generator*



Wie funktioniert das?

- *Übersetzung* von ML/C/Java in *interne Sprache* WL
  - WL: Ausdrücke, Kommandos, lokale Variablen, Funktionen; Referenzen und Felder; Ausnahmen.
- *Verifikations-* und *Vorbedingungskalkül* für WL
- Theoretische Grundlage: Theorie der *Effekte*
- Funktionsweise: Generierung von *Beweisverpflichtungen* aus annotierten Programmen durch *Interpretation*

WL Konkrete Syntax: Terme und Programme

- Terme:
  $t ::= \mathit{constant} \mid x \mid x@L \mid f(t_1, \ldots, t_n)$
- Prädikate:
  $$\begin{aligned} p ::= {} & x \mid x(t_1,\ldots,t_n) \mid \texttt{true} \mid \texttt{false} \mid \texttt{not}\ p \mid \\ & p\ \texttt{and}\ p \mid p\ \texttt{or}\ p \mid \texttt{if}\ t\ \texttt{then}\ p\ \texttt{else}\ p \mid \\ & \texttt{forall}\ x:\beta.p \mid \texttt{exists}\ x:\beta.p \end{aligned}$$
- Programme:
  $$\begin{aligned} e ::= {} & \{p\}e\{p\} \mid t \mid !x \mid x := e \mid \texttt{ref}\ e \\ & x[e] \mid x[e] := e \mid e_1; e_2 \mid L : e \mid \\ & \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \mid \texttt{let}\ x = e\ \texttt{in}\ e \mid \\ & \texttt{fun}(x:\tau)\texttt{->}e \mid (ee) \mid \texttt{rec}\ x:\tau\{\texttt{variant}\ t\} = e \mid \\ & \texttt{while}\ e\ \texttt{do}\ \{\texttt{invariant}\ p\ \texttt{variant}\ t\}\ e\ \texttt{done} \end{aligned}$$

WL Konkrete Syntax: Effekte

- *Effekte*:
  $$\begin{aligned} \beta &::= \texttt{unit} \mid \texttt{bool} \mid \texttt{int} \mid \texttt{float} \\ \tau &::= \beta \mid \beta\ \texttt{ref} \mid \beta\ \texttt{array} \mid x:\tau\texttt{->}\kappa \\ \kappa &::= \{p\}\tau\epsilon\{p\} \\ \epsilon &::= \texttt{reads}\ x,\ldots,x\ \texttt{writes}\ x,\ldots,x \end{aligned}$$
- *Effekte* formalisieren *Seiteneffekte*
  - Typen und Terme wechselseitig rekursiv: *abhängige Typen* (dependent types)
  - Effekt enthält *Vor-*, *Nachbedingung* und *Referenzen*
  - Einschränkung: *kein Aliasing* für Referenzen

Vorbedingungen

- Für annotierte Tupel:
  $wp(\{p'\}e\{q'\}, q) = p' \wedge \forall \mathit{result}.\forall\omega.q' \Longrightarrow q$
- Für Ausdrücke (Auszug):
  $$\begin{aligned} wp(t,q) &= q\,[t/\mathit{result}] \\ wp(!x,q) &= q\,[t/x] \\ wp(x := e, q) &= wp(e, q\,[\texttt{void}/\mathit{result}]\,[\mathit{result}/x] \\ wp(e_1; e_2) &= wp(e_1, wp(e_2, q)) \\ wp(L : e, q) &= wp(e,q)\,[x/x@L] \\ wp(\texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3, q) & \\ &= wp(e_1, \texttt{if}\ \mathit{result}\ \texttt{then}\ wp(e_2,q)\ \texttt{else}\ wp(e_3,q)) \\ wp(\texttt{let}\ x = e_1\ \texttt{in}\ e_2, q) & \\ &= wp(e_1, wp(e_2,q)\,[\mathit{result}/x]) \end{aligned}$$

Erweiterung für Ausnahmen

- Effekte und Nachbedingungen:
  $$\begin{aligned} \epsilon &::= \texttt{reads}\ x,\ldots,x\ \texttt{writes}\ x,\ldots,x\ \texttt{raises}\ E \\ \kappa &::= \{p\}\tau\epsilon\{p \mid E \implies p\} \end{aligned}$$
- Kommandos für Ausnahmen (und Hoare-Tripel)
  $e ::= \ldots \mid \texttt{raise}\ (Ee) \mid \texttt{try}\ e\ \texttt{with}\ Ex\texttt{->}e\ \texttt{end}$
- Nachbedingungen: $wp(e,q,r)$ (Ausnahmenachbedingung r)
  $$\begin{aligned} wp(\texttt{raise}\ (Ee), q, r) &= wp(e, r, r) \\ wp(\texttt{try}\ e_1\ \texttt{with}\ Ex\texttt{->}e_2\ \texttt{end}, q, r) &= wp(e_1, q, wp(e_2, q, r)\,[\mathit{result}/x]) \end{aligned}$$

Verifikation von C-Programmen

- *Übersetzung* von C nach WL, dann Verifikation.
- Zwei Wege:
- *High-level-Übersetzung*:
  - *Einfachere* Beweisverpflichtungen
  - *Nicht den vollen* C-Sprachumfang
- *Low-level-Übersetzung* (Caduceus):
  - *Kompliziertere* Beweisverpflichtungen
  - *Voller* C-Sprachumfang

Zusammenfassung

- Why: *Multi-Language Multi-Prover Verification Condition*
- Grundlage: Typtheorie, *Effekte*
- Erzeugt *Beweise*
- Generiert *Beweisverpflichtungen* für Coq, PVS, Isabelle etc.
- Intern: Imperativ-funktionale Sprache WL
- Frontends für *C*, *Java* — Modellierung des *Speichers*
- Webseite: `http://why.lri.fr`