Einladung zum WIT-Kolloquium

Das Wissenschafterinnenkolleg Internettechnologien WIT in Zusammenarbeit mit der OCG lädt ein zu folgendem Vortrag...

How to give a great research talk? How to write a great research paper?
Simon Peyton Jones
Microsoft Research Cambridge

...einer der "Väter" von Haskell!!!

ZEIT: **Freitag, 15. Oktober 2004, 9:30 - 12:00 Uhr**
ORT: TU Wien, Neues Elektrotechnisches Institutsgebäude, HS EI 10, 1040 Wien, Gußhausstraße 27-29
MEHR INFO: `http://www.wit.at/events/` oder `http://inn.tuwien.ac.at/`

Das WIT erbittet Anmeldung unter `wit.tuwien.ac.at/events`!

Hinweise in eigener Sache...

- *Anmeldesystem ist freigeschaltet!*
  `http://www.complang.tuwien.ac.at/ivan/stud/`
  ...Anmeldungen sind noch bis zum 17.10.2004 möglich
- *Homepages der LVA sind umgezogen!*
  `http://www.complang.tuwien.ac.at/knoop/ak{1|2}_ws0405.html`
  ...sind die neuen Homepage-URLs

Weitere Literaturhinweise

1. Krzysztof R. Apt, Ernst-Rüdiger Olderog. *Programmverifikation*. Springer, 1994.
2. Ernst-Rüdiger Olderog, Bernhard Steffen. *Formale Semantik und Programmverifikation*. In "Handbuch der Informatik", Rechenberg und Pomberger (Hrsg.), Hanser Verlag, 143-163.
3. Hanne R. Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*, Wiley Professional Computing, Wiley, 1992.
   (Siehe `http://www.daimi.au.dk/~bra8130/Wiley_book/wiley.html` für eine frei verfügbare (überarbeitete) Version.)

Syntax von WHILE

Die Sprache der WHILE-Programme π

- Zuweisungen (einschließlich der leeren Anweisung)
- Fallunterscheidungen
- while-Schleife
- Sequentielle Komposition

Syntax

$\pi \ ::= \ x := t \mid skip \mid \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \ fi \mid \text{while } b \text{ do } \pi_1 \text{ od} \mid \pi_1; \pi_2$

Semantik
...Zustandstransformationen

$$[\![\ ]\!] : \pi \to (\Sigma \to \Sigma)$$

wobei $\Sigma =_{df} \{\sigma \mid \sigma : V \to D\}$

Semantik von WHILE

Hier: ... *strukturell operationelle Semantik* (*SOS*) i.S.v. G. Plotkin.

Zentral:

- *(Speicher-) Zustände:* $\Sigma =_{df} \{\sigma \mid \sigma : \mathbf{Var} \to \mathbb{Z}\}$
- *Semantik arithmetischer Ausdrücke:*
  $$[\![\,.\,]\!]_A : \mathbf{Aexpr} \to (\Sigma \to \mathbb{Z})$$
- *Semantik Boolescher Ausdrücke:*
  $$[\![\,.\,]\!]_B : \mathbf{Bexpr} \to (\Sigma \to \mathbb{B})$$
- *SO-Semantik von WHILE:*
  $$[\![\,.\,]\!] : \mathbf{Prg} \to (\Sigma \to \Sigma)$$

Semantik arithmetischer Ausdrücke

$[\![\,.\,]\!]_A : \mathbf{Aexpr} \to (\Sigma \to \mathbb{Z})$ induktiv definiert durch

- $[\![\, n \,]\!]_A(\sigma) =_{df} [\![\, n \,]\!]_N$
- $[\![\, x \,]\!]_A(\sigma) =_{df} \sigma(x)$
- $[\![\, a_1 + a_2 \,]\!]_A(\sigma) =_{df} plus([\![\, a_1 \,]\!]_A(\sigma), [\![\, a_2 \,]\!]_A(\sigma))$
- $[\![\, a_1 * a_2 \,]\!]_A(\sigma) =_{df} mult([\![\, a_1 \,]\!]_A(\sigma), [\![\, a_2 \,]\!]_A(\sigma))$
- ... (andere Operatoren analog)

wobei:

$[\![\,.\,]\!]_N : \mathbf{Num} \to \mathbb{Z}$ induktiv definiert durch

- $[\![\, 0 \,]\!]_N =_{df} 0$, ..., $[\![\, 9 \,]\!]_N =_{df} 9$
- $[\![\, ni \,]\!]_N =_{df} plus(mult(10, [\![\, n \,]\!]_A), [\![\, i \,]\!]_N)$, $i \in \{0, \ldots, 9\}$
- $[\![\, -n \,]\!]_N =_{df} minus([\![\, n \,]\!]_N)$

Semantik Boolescher Ausdrücke

$[\![\,.\,]\!]_B : \mathbf{Bexpr} \to (\Sigma \to \mathbb{B})$ induktiv definiert durch

- $[\![\, true \,]\!]_B(\sigma) =_{df} \mathtt{tt}$
- $[\![\, false \,]\!]_B(\sigma) =_{df} \mathtt{ff}$
- $[\![\, a_1 = a_2 \,]\!]_B(\sigma) =_{df} \begin{cases} \mathtt{tt} & \text{falls } equal([\![\, a_1 \,]\!]_A(\sigma), [\![\, a_2 \,]\!]_A(\sigma)) \\ \mathtt{ff} & \text{sonst} \end{cases}$
- ... (andere Relatoren (z.B. $<$, $\leq$, ...) analog)
- $[\![\, \neg b \,]\!]_B(\sigma) =_{df} neg([\![\, b \,]\!]_B(\sigma))$
- $[\![\, b_1 \wedge b_2 \,]\!]_B(\sigma) =_{df} conj([\![\, b_1 \,]\!]_B(\sigma), [\![\, b_2 \,]\!]_B(\sigma))$
- $[\![\, b_1 \vee b_2 \,]\!]_B(\sigma) =_{df} disj([\![\, b_1 \,]\!]_B(\sigma), [\![\, b_2 \,]\!]_B(\sigma))$

Strukturell operationelle Semantik von WHILE

... beschreibt den Berechnungsvorgang als Folge elementarer Speicherzustandsübergänge.

Zentral:

- *Konfigurationen*
  - *Nichtterminale* (oder *Zwischen-*) *Konfigurationen:* $\langle \Pi, \sigma \rangle$:
    ... Programmfragment Π ist auf den (Zwischen-) Zustand σ anzuwenden.
  - *Terminale* (oder *finale*) *Konfigurationen:* σ oder $undef$
    ... beschreiben das Resultat nach Ende der Berechnung.

SOS-Regeln von WHILE: Teil 1

[skip$_{sos}$] $\dfrac{}{\langle \textbf{skip},\sigma\rangle \Rightarrow \sigma}$

[abort$_{sos}$] $\dfrac{}{\langle \textbf{abort},\sigma\rangle \Rightarrow undef}$

[ass$_{sos}$] $\dfrac{}{\langle x{:=}t,\sigma\rangle \Rightarrow \sigma[\llbracket t \rrbracket_A(\sigma)/x]}$

[comp$^1_{sos}$] $\dfrac{\langle \Pi_1,\sigma\rangle \Rightarrow \langle \Pi_1',\sigma'\rangle}{\langle \Pi_1;\Pi_2,\sigma\rangle \Rightarrow \langle \Pi_1';\Pi_2,\sigma'\rangle}$

[comp$^2_{sos}$] $\dfrac{\langle \Pi_1,\sigma\rangle \Rightarrow \sigma'}{\langle \Pi_1;\Pi_2,\sigma\rangle \Rightarrow \langle \Pi_2,\sigma'\rangle}$

SOS-Regeln von WHILE: Teil 2

[if$^{tt}_{sos}$] $\dfrac{}{\langle \textbf{if } b \textbf{ then } \Pi_1 \textbf{ else } \Pi_2 \textbf{ fi},\sigma\rangle \Rightarrow \langle \Pi_1,\sigma\rangle}$ $\llbracket b \rrbracket_B(\sigma) = \mathtt{tt}$

[if$^{ff}_{sos}$] $\dfrac{}{\langle \textbf{if } b \textbf{ then } \Pi_1 \textbf{ else } \Pi_2 \textbf{ fi},\sigma\rangle \Rightarrow \langle \Pi_2,\sigma\rangle}$ $\llbracket b \rrbracket_B(\sigma) = \mathtt{ff}$

[while$_{sos}$] $\dfrac{}{\langle \textbf{while } b \textbf{ do } \Pi \textbf{ od},\sigma\rangle \Rightarrow \langle \textbf{if } b \textbf{ then } \Pi; \textbf{ while } b \textbf{ do } \Pi \textbf{ od else skip fi},\sigma\rangle}$

Berechnungsschritte, Berechnungsfolgen

- Ein *Berechnungsschritt* ... ist von der Form

$$\langle \Pi,\sigma\rangle \Rightarrow \gamma \quad \text{mit} \quad \gamma \in (\mathbf{Prg} \times \Sigma) \cup \Sigma \cup \{undef\}$$

- Eine *Berechnungsfolge* zu einem Programm Π angesetzt auf einen (Start-) Zustand $\sigma \in \Sigma$ ist
  - eine endliche Folge $\gamma_0, \ldots, \gamma_k$ von Konfigurationen mit $\gamma_0 = \langle \Pi,\sigma\rangle$ und $\gamma_i \Rightarrow \gamma_{i+1}$ für alle $i \in \{0,\ldots,k-1\}$,
  - eine unendliche Folge von Konfigurationen mit $\gamma_0 = \langle \Pi,\sigma\rangle$ und $\gamma_i \Rightarrow \gamma_{i+1}$ für alle $i \in \mathbb{N}$.

Terminierende und divergierende Berechnungsfolgen

- Eine maximale (d.h. nicht mehr verlängerbare) Berechnungsfolge heißt
  - *regulär terminierend*, wenn sie endlich ist und die letzte Konfiguration aus Σ ist,
  - *irregulär terminierend*, wenn sie endlich ist und die letzte Konfiguration $undef$ ist,
  - *divergierend*, falls sie unendlich ist.

Beachte: WHILE-Programme sind *deterministisch*.

Axiomatische Semantik von WHILE

Hier: ... Korrektheit und Vollständigkeit der axiomatischen Semantik

Erinnerung:

- *Hoare-Tripel* (syntaktische Sicht) bzw. *Korrektheitsformeln* (semantische Sicht) der Form

$$\{p\}\ \pi\ \{q\}$$

- Gültigkeit einer Korrektheitsformel im Sinne
  - *partieller* Korrektheit
  - *totaler* Korrektheit

Partielle und totale Korrektheit

- Die Zustandsmenge

$$Ch(p) =_{df} \{\sigma \in \Sigma \mid \llbracket p \rrbracket_B(\sigma) = \mathtt{tt}\}$$

heißt *Charakterisierung von* $p \in \mathbf{Bexp}$.

- *Semantik von Korrektheitsformeln:*
Eine Korrektheitsformel $\{p\}\ \pi\ \{q\}$ heißt
  - *partiell korrekt*, falls $\llbracket \pi \rrbracket(Ch(p)) \subseteq Ch(q)$
  - *total korrekt*, falls $\{p\}\ \pi\ \{q\}$ partiell korrekt ist und $Def(\llbracket \pi \rrbracket) \supseteq Ch(p)$ gilt.

Konvention: $\llbracket \pi \rrbracket(Ch(p)) =_{df} \{\llbracket \pi \rrbracket(\sigma) \mid \sigma \in Ch(p)\}$

Hoare-Kalkül HK_{PK} für partielle Korrektheit

[skip] $\dfrac{}{\{p\}\ \textbf{skip}\ \{p\}}$

[abort] $\dfrac{}{\{p\}\ \textbf{abort}\ \{q\}}$

[ass] $\dfrac{}{\{p[t\backslash x]\}\ x{:=}t\ \{p\}}$

[comp] $\dfrac{\{p\}\ \pi_1\ \{r\},\ \{r\}\ \pi_2\ \{q\}}{\{p\}\ \pi_1;\pi_2\ \{q\}}$

[ite] $\dfrac{\{p \wedge b\}\ \pi_1\ \{q\},\ \{p \wedge \neg b\}\ \pi_2\ \{q\}}{\{p\}\ \textbf{if } b \textbf{ then } \pi_1 \textbf{ else } \pi_2 \textbf{ fi}\ \{q\}}$

[while] $\dfrac{\{I \wedge b\}\ \pi\ \{I\}}{\{I\}\ \textbf{while } b \textbf{ do } \pi \textbf{ od}\ \{I \wedge \neg b\}}$

[cons] $\dfrac{p \Rightarrow p_1,\ \{p_1\}\ \pi\ \{q_1\},\ q_1 \Rightarrow q}{\{p\}\ \pi\ \{q\}}$

Hoare-Kalkül HK_{TK} für totale Korrektheit

... identisch mit HK_{PK}, wobei aber Regel [$while$] ersetzt ist durch:

[while$_{TK}$] $\dfrac{I \wedge b \Rightarrow u[t/v],\ \{I \wedge b \wedge t{=}w\}\ \pi\ \{I \wedge t{<}w\}}{\{I\}\ \textbf{while } b \textbf{ do } \pi \textbf{ od}\ \{I \wedge \neg b\}}$

wobei

- u Boolescher Ausdruck über der Variablen v; t Term,
- w Variable, die in I, b, π und t nicht frei vorkommt,
- $M =_{df} \{\sigma(v) \mid \sigma \in \Sigma \wedge \llbracket u \rrbracket_B(\sigma) = \mathtt{tt}\}$ noethersch geordnete Menge (sog. noethersche Halbordnung).

Korrektheit und Vollständigkeit von HK_{PK} und HK_{TK}

Sei K ein Kalkül für partielle bzw. totale Korrektheit

Frage der ...

- *Korrektheit:* ... ist jede mithilfe von K ableitbare Korrektheitsformel partiell bzw. total korrekt?
- *Vollständigkeit:* ... ist jede partiell bzw. total korrekte Korrektheitsformel mithilfe von K ableitbar?

Speziell:

- Sind HK_{PK} und HK_{TK} *korrekt* und *vollständig*?

Erster Beweis

```
                                                        ____                                                     ____
                                ---------------------------------------------- Ass   ---------------------------------------------- Ass
                                {(y*x)*(x-1)!=a! & (x-1)>0} y:=y*x {y*(x-1)!=a! & (x-1)>0}   {y*(x-1)!=a! & (x-1)>0} x:=x-1 {y*x!=a! & x>0}
                                ---------------------------------------------------------------------------------------------------- Comp
y*x!=a! & x>0 & x>1 ==> (y*x)*(x-1)!=a! & (x-1)>0   {(y*x)*(x-1)!=a! & (x-1)>0} y:=y*x; x:=x-1 {y*x!=a! & x>0}
---------------------------------------------------------------------------------------------------- Cons
                        {y*x!=a! & x>0 & x>1} y:=y*x; x:=x-1 {y*x!=a! & x>0}
                        ---------------------------------------------------------------------------- While
         T              {y*x!=a! & x>0} WHILE x>1 DO y:=y*x; x:= x-1 OD {y*x!=a! & x>0 & ~(x>1)}   y*x!=a! & x>0 & ~(x>1) ==> y=a!
------------------------------ Cons   ---------------------------------------------------------------------------------------- Cons
{a>0} x:=a; y:=1 {y*x!=a! & x>0}        {y*x!=a! & x>0} WHILE x>1 DO y:=y*x; x:= x-1 OD {y=a!}
---------------------------------------------------------------------------------------------------- Comp
                {a>0} x:=a; y:=1; WHILE x>1 DO y:=y*x; x:=x-1 OD {y=a!}
```

wobei

```
           ____                                                  ____
     ----------------------------------------- Ass   ----------------------------------------- Ass
     {1*a!=a! & a>0 & a=a} x:=a {1*x!=a! & a>0 & x=a}   {1*x!=a! & a>0 & x=a} y:=1 {y*x!=a! & a>0 & x=a}
     ---------------------------------------------------------------------------------------------- Comp
T ≡  a>0 ==> 1*a!=a! & a>0 & a=a   {1*a!=a! & a>0 & a=a} x:=a; y:=1 {y*x!=a! & a>0 & x=a}   y*x!=a! & a>0 & x=a ==> y*x!=a! & x>0
     -------------------------------------------------------------------------------------------------------------------- Cons
                            {a>0} x:=a; y:=1 {y*x!=a! & x>0}
```

& : Logisches und

~ : Logisches nicht

Zweiter Beweis

```
                                                        ____                                                     ____
                                ---------------------------------------------- Ass   ---------------------------------------------- Ass
                                {(y*x)*(x-1)!=a! & (x-1)>0} y:=y*x {y*(x-1)!=a! & (x-1)>0}   {y*(x-1)!=a! & (x-1)>0} x:=x-1 {y*x!=a! & x>0}
                                ---------------------------------------------------------------------------------------------------- Comp
y*x!=a! & x>0 & x>1 ==> (y*x)*(x-1)!=a! & (x-1)>0   {(y*x)*(x-1)!=a! & (x-1)>0} y:=y*x; x:=x-1 {y*x!=a! & x>0}
---------------------------------------------------------------------------------------------------- Cons
                        {y*x!=a! & x>0 & x>1} y:=y*x; x:=x-1 {y*x!=a! & x>0}
                        ---------------------------------------------------------------------------- While
         T|             {y*x!=a! & x>0} WHILE x>1 DO y:=y*x; x:= x-1 OD {y*x!=a! & x>0 & ~(x>1)}   y*x!=a! & x>0 & ~(x>1) ==> y=a!
------------------------------ Cons   ---------------------------------------------------------------------------------------- Cons
{a>0} x:=a; y:=1 {y*x!=a! & x>0}        {y*x!=a! & x>0} WHILE x>1 DO y:=y*x; x:= x-1 OD {y=a!}
---------------------------------------------------------------------------------------------------- Comp
                {a>0} x:=a; y:=1; WHILE x>1 DO y:=y*x; x:=x-1 OD {y=a!}
```

wobei

```
            ____                                           ____
      ---------------------------------- Ass   ---------------------------------- Ass
      {1*a!=a! & a>0} x:=a {1*x!=a! & x>0}   {1*x!=a! & x>0} y:=1 {y*x!=a! & x>0}
      ---------------------------------------------------------------------- Comp
T| ≡  a>0 ==> 1*a!=a! & a>0   {1*a!=a! & a>0} x:=a; y:=1 {y*x!=a! & x>0}
      ------------------------------------------------------------------------------ Cons
                       {a>0} x:=a; y:=1 {y*x!=a! & x>0}
```

& : Logisches und

~ : Logisches nicht


```
                      ____                                                  ____
Ass   ---------------------------------------------------   --------------------------------------------- Ass
      {x=(q+1)*y+r-y & r-y>=0} q:=q+1 {x=q*y+r-y & r-y>=0}   {x=q*y+r-y & r-y>=0} r:=r-y {x=q*y+r & r>=0}
Comp  ---------------------------------------------------------------------------------------------------
      (x=q*y+r & r>=0 & r>=y) => (x=(q+1)*y+r-y & r-y>=0)   {x=(q+1)*y+r-y & r-y>=0} q:=q+1; r:=r-y {x=q*y+r & r>=0}
Cons  ---------------------------------------------------------------------------------------------------
                  {x=q*y+r & r>=0 & r>=y} q:=q+1; r:=r-y {x=q*y+r & r>=0}
While ---------------------------------------------------------------------------------------------------
      {x=q*y+r & r>=0} WHILE r>=y DO q:=q+1; r:=r-y OD {x=q*y+r & r>=0 & ~(r>=y)}   (x=q*y+r & r>=0 & ~(r>=y)) => (x=q*y+r & r>=0 & r<y)
Cons  -------------------------------------------------------------------------------------------------------------------------------

             ____                                ____
Ass   -------------------------------   ------------------------------- Ass
      {x=0*y+x & x>=0} q:=0 {x=q*y+x & x>=0}   {x=q*y+x & x>=0} r:=x {x=q*y+r & r>=0}
      ------------------------------------------------------------------------------ Comp
      (x>=0 & y>0) => (x=0*y+x & x>=0) {x=0*y+x & x>=0} q:=0; r:=x {x=q*y+r & r>=0}
      ------------------------------------------------------------------------------ Cons
      {x>=0 & y>0} q:=0; r:=x {x=q*y+r & r>=0}        {x=q*y+r & r>=0} WHILE r>=y DO q:=q+1; r:=r-y OD {x=q*y+r & r>=0 & r<y}
      ------------------------------------------------------------------------------------------------------------------------ Comp
              {x>=0 & y>0} q:=0; r:=x; WHILE r>=y DO q:=q+1; r:=r-y OD {x=q*y+r & r>=0 & r<y}
```

& : Logisches und

~ : Logisches nicht

Hauptresultate

Zur Korrektheit:

Theorem [Korrektheit von HK_{PK} und HK_{TK}]

1. HK_{PK} ist korrekt, d.h. jede mit HK_{PK} ableitbare Korrektheitsformel ist gültig im Sinne partieller Korrektheit.
2. HK_{TK} ist korrekt, d.h. jede mit HK_{TK} ableitbare Korrektheitsformel ist gültig im Sinne totaler Korrektheit.

Beweis ... durch Induktion über die Anzahl der Regelanwendungen im Beweisbaum zur Ableitung der Korrektheitsformel.

Zur Vollständigkeit:

i.a. nur *relative* Vollständigkeit von Korrektheitskalkülen beweisbar.