Denotationale Semantik I: Aexp

$$
\begin{aligned}
\mathcal{E}[\![n]\!] &\stackrel{def}{=} \lambda\,\sigma^{\Sigma}.n \\
\mathcal{E}[\![X]\!] &\stackrel{def}{=} \lambda\,\sigma^{\Sigma}.\sigma(X) \\
\mathcal{E}[\![a_0 \texttt{ + } a_1]\!] &\stackrel{def}{=} \lambda\,\sigma^{\Sigma}.(\mathcal{E}[\![a_0]\!]\sigma + \mathcal{E}[\![a_1]\!]\sigma) \\
\mathcal{E}[\![a_0 \texttt{ - } a_1]\!] &\stackrel{def}{=} \lambda\,\sigma^{\Sigma}.(\mathcal{E}[\![a_0]\!]\sigma - \mathcal{E}[\![a_1]\!]\sigma) \\
\mathcal{E}[\![a_0 \texttt{ * } a_1]\!] &\stackrel{def}{=} \lambda\,\sigma^{\Sigma}.(\mathcal{E}[\![a_0]\!]\sigma \cdot \mathcal{E}[\![a_1]\!]\sigma)
\end{aligned}
$$

Denotationale Semantik II: Bexp

$$
\begin{aligned}
\mathcal{B}[\![\texttt{True}]\!] &\stackrel{def}{=} \lambda\,\sigma^{\Sigma}.true \\
\mathcal{B}[\![\texttt{False}]\!] &\stackrel{def}{=} \lambda\,\sigma^{\Sigma}.false \\
\mathcal{B}[\![\texttt{not } b]\!] &\stackrel{def}{=} \lambda\,\sigma^{\Sigma}.\neg\mathcal{B}[\![b]\!]\sigma \\
\mathcal{B}[\![a_0 \texttt{ = } a_1]\!] &\stackrel{def}{=} \lambda\,\sigma^{\Sigma}.\begin{cases} true & \mathcal{E}[\![a_0]\!]\sigma = \mathcal{E}[\![a_1]\!]\sigma \\ false & \mathcal{E}[\![a_0]\!]\sigma \neq \mathcal{E}[\![a_1]\!]\sigma \end{cases} \\
\mathcal{B}[\![a_0 \texttt{ <= } a_1]\!] &\stackrel{def}{=} \lambda\,\sigma^{\Sigma}.\begin{cases} true & \mathcal{E}[\![a_0]\!]\sigma \leq \mathcal{E}[\![a_1]\!]\sigma \\ false & \mathcal{E}[\![a_0]\!]\sigma > \mathcal{E}[\![a_1]\!]\sigma \end{cases} \\
\mathcal{B}[\![b_0 \texttt{ and } b_1]\!] &\stackrel{def}{=} \lambda\,\sigma^{\Sigma}.\mathcal{B}[\![b_0]\!]\sigma \wedge \mathcal{B}[\![b_1]\!]\sigma \\
\mathcal{B}[\![b_0 \texttt{ or } b_1]\!] &\stackrel{def}{=} \lambda\,\sigma^{\Sigma}.\mathcal{B}[\![b_0]\!]\sigma \vee \mathcal{B}[\![b_1]\!]\sigma
\end{aligned}
$$

Denotationale Semantik für IMP: Anweisungen

$$
\begin{aligned}
\mathcal{D}[\![\texttt{skip}]\!] &\stackrel{def}{=} \lambda\,\sigma^{\Sigma}.\sigma \\
\mathcal{D}[\![X := a]\!] &\stackrel{def}{=} \lambda\,\sigma^{\Sigma}.\sigma[\mathcal{E}[\![a]\!]\sigma/X] \\
\mathcal{D}[\![c_0;\ c_1]\!] &\stackrel{def}{=} \mathcal{D}[\![c_1]\!] \circ \mathcal{D}[\![c_0]\!] \\
\mathcal{D}[\![\texttt{if } b \texttt{ then } c_0 \texttt{ else } c_1 \texttt{ fi}]\!] &\stackrel{def}{=} \lambda\,\sigma^{\Sigma}.\begin{cases} \mathcal{D}[\![c_0]\!]\sigma & \mathcal{B}[\![b]\!]\sigma = true \\ \mathcal{D}[\![c_1]\!]\sigma & \mathcal{B}[\![b]\!]\sigma = false \end{cases} \\
\mathcal{D}[\![\texttt{while } b \texttt{ do } c \texttt{ od}]\!] &\stackrel{def}{=} fix\ \Gamma \\
\text{mit } \Gamma(\phi) &\stackrel{def}{=} \lambda\,\sigma^{\Sigma}.\begin{cases} \phi \circ \mathcal{D}[\![c]\!]\sigma & \mathcal{B}[\![b]\!]\sigma = true \\ \sigma & \mathcal{B}[\![b]\!]\sigma = false \end{cases}
\end{aligned}
$$

Vorlesung vom 09.01.06: Algebraische Spezifikationen

Wo sind wir?

- Teil I: Grundlagen formaler Logik
- Teil II: Modellierung
- Teil III: Spezifikationsformalismen
  - *Algebraische Spezifikation* mit Isabelle und CASL
  - Modellbasierte Spezifikation mit Z

Algebraische Spezfikation

- Idee: Spezifikation ist *Signatur*, Programme sind *Algebren*
- Mathematische Grundlage: universelle *Algebra*
- *Geschichtliches*:
  - *Entstanden* um 1976 (ADJ-Gruppe)
  - In den 80ern *Vielzahl* von algebraischen Sprachen
  - Ende 90er Entwicklung der *Einheitssprache CASL*

Algebraische Spezifikation: die Grundidee

- *Deklaration* von Typen und Operationen in *Signatur*
- Gewünschte *Eigenschaften* als *Axiome*
- *Semantik*: *lose* (alle Algebren) vs. *initial* (Termalgebra)
  - Aber: *Semantik* ist etwas für verregnete Herbsttage
  - Hier: *Beweis* und (syntaktische) *Herleitung*
- *Implementationsbegriff* durch *Signatur*- und *Theoriemorphismen*

Ein klassisches Beispiel

- Ein *Stack* hat zwei *Sorten* und vier *Operationen*:

```
typedecl 'a stack
consts
  empty :: "'a stack"
  push  :: "'a stack => 'a => 'a stack"
  pop   :: "'a stack => 'a stack"
  top   :: "'a stack => 'a"
```

- Axiome: `pop` und `top` invers zu `push`

```
axioms a1: "top (push s x) = x"
       a2: "pop (push s x) = s"
```

  - `pop`, `top` partiell?
  - *Keine* Seiteneffekte

Typen

Def (Typdeklarationen): $\mathcal{T} = \langle T, tar \rangle$, wobei T Menge von *Typkonstanten*, mit $tar : T \to \mathbb{N}$ *Typaritäten*

Def: Aus *Typdeklaration* $\mathcal{T}$ generierte *Typen* $\mathcal{T}^*$ ist kleinste Menge

(i) $\alpha \in \mathcal{V}_{Type} \Longrightarrow \alpha \in \mathcal{T}^*$

(ii) $t_1, \ldots, t_n \in \mathcal{T}^*, c \in T, tar(t) = n \Longrightarrow c(t_1, \ldots, t_n) \in \mathcal{T}^*$

Vordefinierte Typkonstanten: $\Rightarrow$, $bool$
mit Arität $tar(\Rightarrow) = 2$, $tar(bool) = 0$.

Signatur und Terme

Def (Signatur): $\Sigma = \langle \mathcal{T}, \Omega, ar \rangle$ wobei $\mathcal{T}$ Typdeklaration, Ω Menge von *Operationen* mit $ar : \Omega \to \mathcal{T}^*$

Def: Gegeben Signatur Σ, Menge X von Variablen mit $type(x) \to \mathcal{T}^*$. *Terme* $\{T_{\Sigma,t}(X)\}_{t \in \mathcal{T}^*}$ sind kleinste Menge so dass

(i) $c \in \Sigma \Longrightarrow c \in T_{\Sigma, ar(c)}(X)$

(ii) $x \in X \Longrightarrow x \in T_{\Sigma, type(x)}(X)$

(iii) $r \in T_{\Sigma,t}(X), s \in T_{\Sigma, t \Rightarrow u}(X) \Longrightarrow r\, s \in T_{\Sigma,u}(X)$

(iv) $x \in X, s \in T_{\Sigma, type(x) \Rightarrow t}(X) \Longrightarrow \lambda\, x^{type(x)}.s \in T_{\Sigma,t}(X)$

Vordefinierte Operationen: siehe HOL.

Spezifikationen und Theoreme

Def (Spezifikation): $\mathcal{S}p = \langle \Sigma, \mathcal{A}x \rangle$ mit Signatur Σ und Menge von *Axiomen* (Terme $t \in T_{\Sigma, bool}(X)$

- Isabelle: Spezifikation $\equiv$ Theorie

Def (Theorem): $\phi \in T_{\Sigma, bool}(X)$, so dass $\mathcal{S}p \vdash \phi$ (ϕ in $\mathcal{S}p$ *herleitbar*)

- *Herleitbarkeit*: logischer Kalkül, z.B. natürliches Schließen in HOL
- *Gültigkeit* ($\mathcal{S}p \models \phi$): ϕ gilt in *allen Modellen*
- *Korrektheit* $\mathcal{S}p \vdash \phi \Longrightarrow \mathcal{S}p \models \phi$, *Vollständigkeit* $\mathcal{S}p \models \phi \Longrightarrow \mathcal{S}p \vdash \phi$
  - Eigenschaften des logischen *Kalküls*
  - HOL: *nicht vollständig*
  - FOL, Gleichungslogik: *vollständig*
  - *Gödel*: natürliche Zahlen + Konsistenz $\Longrightarrow$ *Unvollständigkeit*

Implementation

- *Implementation*: Axiome müssen *bewiesen* werden.
- *Beispiel*: Implementation von *Stack* durch *Listen*
- *Übersetzung* von *Typen* und *Operationen*:
  - *Abbildung* der Typen: $stack \mapsto list$
  - *Abbildung* der Operationen $empty \mapsto Nil$, $push \mapsto Cons, \ldots$
  - *Abbildung* muss *Aritäten* bewahren.
  - Damit *Übersetzung* von Termen möglich.

Signaturmorphismus

Def: Gegeben $\Sigma = \langle \mathcal{T}, \Omega \rangle, \Sigma' = \langle \mathcal{T}', \Omega' \rangle$. Ein *Signaturmorphismus* $\sigma : \Sigma \to \Sigma'$ ist $\sigma_T : T \to T', \sigma_\Omega : \Omega \to \Omega'$ so dass

(i) $tar(\sigma_T(t)) = tar(t)$, $\sigma_\Omega(\Rightarrow) = \Rightarrow$
Dann $\sigma_T^* : \mathcal{T}^* \to \mathcal{T}'^*$

$$\begin{aligned} \sigma_T^*(\alpha) &\stackrel{def}{=} \alpha \\ \sigma_T^*(c(t_1, \ldots, t_n)) &\stackrel{def}{=} \sigma_T(c)(\sigma_T^*(t_1), \ldots, \sigma_T^*(t_n)) \end{aligned}$$

(ii) $ar(\sigma_\Omega(c)) = \sigma_T^*(ar((c)))$
Dann $\sigma^* : T_\Sigma(X) \to T_{\Sigma'}(X)$

$$\begin{aligned} \sigma^*(x) &\stackrel{def}{=} x \\ \sigma^*(c) &\stackrel{def}{=} \sigma_\Omega(c) \\ \sigma^*(s\, t) &\stackrel{def}{=} \sigma^*(s)\, \sigma^*(t) \\ \sigma^*(\lambda\, x.t) &\stackrel{def}{=} \lambda\, x.\sigma^*(t) \end{aligned}$$

Lemma: Übersetzung *wohldefiniert* $s \in T_{\Sigma,t}(X) \Longrightarrow \sigma^*(s) \in T_{\Sigma, \sigma_T^*(t)}(X)$

Spezifikationsmorphismus

Def: Gegeben $\mathcal{S}p = \langle \Sigma, \mathcal{A}x \rangle, \mathcal{S}p' = \langle \Sigma', \mathcal{A}x' \rangle$ Ein *Spezifikationsmorphismus* (Theoriemorphismus) ist ein Signaturmorphismus $\sigma_\Sigma : \Sigma \to \Sigma'$ der Axiome, so dass $\phi \in \mathcal{A}x$, $\sigma^*(\phi) \in Thm(\mathcal{S}p')$.

Def: Eine *Implementation* für $\mathcal{S}p = \langle \Sigma, \mathcal{A}x \rangle$ ist eine Theorie $\mathcal{T}h = \langle \Sigma', \mathcal{A}x' \rangle$ und ein Spezifikationsmorphismus $\sigma : \mathcal{S}p \to \mathcal{T}h$.

- *Signatur*- und *Theoriemorphismen strukturieren* die Entwicklung

Der Entwicklungszyklus

- *Anforderungsspezifikation* (*requirement specification*)
  - Axiomatisch
  - Unvollständig
- *Entwurfsspezifikation* (*design specification*)
  - Konsistent (konservativ, in HOL)
  - Vollständig
- *Ausführbare Spezifikation* (*executable specification*)
  - Ausführbar (funktionale Untermenge von HOL), testbar
  - Strukturiert

Warteschlangen

- Eine Sorte, fünf Operationen (vgl. Stack)

```
typedecl 'a q
consts
  mtQ     :: "'a q"
  isEmpty :: "'a q => bool"
  enq     :: "'a => 'a q => 'a q"
  front   :: "'a q => 'a"
  deq     :: "'a q => 'a q"
```

Warteschlangen

- Wann ist eine Schlange leer:

```
axioms
  empty: "isEmpty mtQ"
  notEmpty: "~ (isEmpty (enq a q))"
```

- Verhalten von `front`:

```
  front: "front (enq a q) = (if isEmpty q then a
                             else front q)"
```

- Verhalten von `deq`:

```
  deq:  "deq (enq a q) = (if isEmpty q then mtQ
                          else enq a (deq q))"
```


Warteschlangen — Entwurfsspezfikation

- Eine Warteschlange hat einen *Inhalt*
- Selbe Signatur wie oben, plus:

```
consts
  cont    :: "'a q => 'a list"
axioms
  contEq:    "cont q= cont p ==> q= p"
```

- Spezifikation des Verhaltens über den Inhalt

Warteschlangen — Entwurfsspezfikation

- Axiome oben ersetzt durch

```
axioms
  mt:        "cont (mtQ) = []"
  isMtCont:  "isEmpty q = (cont q = [])"
  enqCont:   "cont (enq a q) = cont q @ [a]"
  frontCont: "front q = hd (cont q)"
  deqCont:   "cont (deq q) =  tl (cont q)"
```

- Zusätzliches Axiom

```
  contEq:    "cont q= cont p ==> q= p"
```

  ◦ Vorherige Axiome können *bewiesen* werden

Warteschlangen — Implementation

- Warteschlangen *implementiert* durch Listen

```
datatype 'a q = Q "'a list"
  ...
defs
  mt_def: "mtQ == Q []"
primrec
 "isEmpty (Q q)= null q"
primrec
 "enq x (Q q)= Q (x#q)"
primrec
 "front (Q q)= hd (rev q)"
primrec
 "deq (Q q)= Q (tl (rev q))"
```


Warteschlangen — Struktur der Spezifikationen

$$\begin{tikzcd}
\mathbf{QSpec} \arrow[d, "\sigma"] & \\
\mathbf{QDes} \arrow[d, "\tau"'] \arrow[dr, bend left, "\rho"] & \\
\mathbf{QImp} & \mathbf{QImp2}
\end{tikzcd}$$

- `QSpec` — Anforderungsspezifikation
- `QDes` — Entwurfsspezifikation
- `QImp` — Implementation
- `QImp2` — effizientere Implementation:

```
datatype 'a q =
   Q "'a list" "'a list"
```

Spezifikationsmorphismen können *komponiert* werden:
`QImp`, `QImp2` Implementationen von `QSpec`

Abschließendes Beispiel

- Datenbank mit eindeutigen, generierten Schlüsseln:

```
consts
 init    :: db
 new_key :: db-> (db, key)
 lookup  :: db -> key -> data option
 upd     :: db -> key -> data-> db
 rem     :: db -> key -> db
```

  ◦ Alternative:

```
  lookup   :: db -> key -> data
  contains :: db-> key-> bool
```

  ◦ . . . aber mit `option` kürzer, Axiome lesbarer