```
consts merge :: "('a::linorder)list * 'a list => 'a list"
recdef merge "measure (%(xs,ys). size xs + size ys)"
  "merge(x#xs, y#ys) =
     (if x < y then x # merge(xs, y#ys)
               else y # merge(x#xs, ys))"
  "merge(xs,[]) = xs"
  "merge([],ys) = ys"

consts msort :: "('a::linorder) list => 'a list"
recdef msort "measure size"
    "msort [] = []"
    "msort [x] = [x]"
    "msort xs = merge(msort(take (size xs div 2) xs),
     msort(drop (size xs div 2) xs))"
```


Datentypen

- *Algebraische Datentypen*: auch hier ist *Rekursion* ein Fixpunkt.

  `datatype 'a list = Nil | Cons 'a "'a * 'a list"`

  `type 'a list = fix M. Nil | Cons 'a "'a * M"`
- *Kleinster* Fixpunkt: *endliche* Datentypen
  - Zum Beispiel Listen
- *Größter* Fixpunkt: *unendliche* Datentypen
  - Ströme, unendliche Listen:
    `type 'a stream = fix M. Cons 'a "'a * 'a M"`

Datentypen als Fixpunkte

- In HOL
  - Nur kovariante Rekursion
  - Lösung ist *kleinster* Fixpunkt
- Mit vollständigen Halbordnungen als Trägermengen (HOLCF):
  - *Jede* rekursive Typgleichung (*domain equation*) hat Lösung.
  - *Kleinster Fixpunkt* auch *größter Fixpunkt*
  - Damit auch *unendliche Datentypen* möglich (e.g. Listen)

Zusammenfassung

- Modellierung *rekursiver* Funktionen durch *Fixpunkte*
- *Problem*: Nicht jede Funktion hat einen *Fixpunkt*
- *Lösungen*:
  - *Erweiterung* der Logik: *stetige Funktionen* auf *vollständigen Halbordnungen*
  - *Vorteil:* beliebige Rekursion, *Nachteil:* neue Logik, umständlicheres Beweisen
  - *Beschränkung* der Rekursion: *wohlfundierte Rekursion*
  - *Vorteil:* gute Beweisunterstützung, *Nachteil:* nicht allgemein, Definition umständlich

Vorlesung vom 12.12.05: Modellierung imperativer Programme

Wo sind wir?

- Teil I: Grundlagen formaler Logik
- Teil II: Modellierung
  - Datentypen
  - Rekursive Programme
  - *Imperative Programme*
- Teil III: Spezifikationsformalismen

Fahrplan

- Nachtrag:
  - Datentypen als Fixpunkte
- *Imperative* Programme:
  - *Ausführung* ist *Zustandstransformation*
  - **IMP** — eine einfache imperative Sprache
  - *Denotationale* und *operationale* Semantik

Die Sprache IMP

- *Zahlen*: $\mathbf{N} = \{0, 1, 2, 3, \ldots\}$
- *Wahrheitswerte*: $\mathbf{T} = \{\mathtt{True}, \mathtt{False}\}$
- *Variablen*: **Loc**
- *Arithmetische* Ausdrücke: $a \in \mathbf{Aexp}$

  $a ::= n \mid x \mid a + a \mid a - a \mid a * a \qquad n \in \mathbf{N}, x \in \mathbf{Loc}$
- *Boolsche* Ausdrücke: $b \in \mathbf{Bexp}$

  $b ::= v \mid a = a \mid a \mathtt{<=} a \mid \mathtt{not}\ b \mid b\ \mathtt{and}\ b \mid b\ \mathtt{or}\ b \qquad v \in \mathbf{T}$
- *Anweisungen*:

  $\mathbf{Com}\, c ::= \mathtt{skip} \mid x := a \mid c;\ c \mid$
  $\mathtt{if}\ b\ \mathtt{then}\ c\ \mathtt{else}\ c\ \mathtt{fi} \mid \mathtt{while}\ b\ \mathtt{do}\ c\ \mathtt{od}$

Beispielprogramme in IMP

$Fac \equiv$
```
M := 1;
while 0 < N do
  M:= N* M;
  N:= N- 1
od
```

$Euclid \equiv$
```
while (not (M= N)) do
  if (M<= N)
  then N:= N- M
  else M:= M- N
  fi
od
```


Semantik

- *Denotationale* Semantik:
  - Abbildung $\mathcal{D}$: **Com** $\rightarrow \mathcal{S}$ in *semantischen Bereich*
  - Für: *Programmentwicklung* und *Programmtransformation*
- *Operationale* Semantik:
  - *Syntaxgesteuerte* Ableitungsregeln (Zustandsübergang)
  - Für: *Sprachdefinition*, *Compilerverifikation*
- *Axiomatische* Semantik:
  - *Vor*- und *Nachbedingungen* für jede Anweisung
  - Für: *Programmverifikation*

Denotationale Semantik

- Letzte VL: denotationelle Semantik einer *funktionalen* Sprache
  - Programm als *Funktion* $\mathbf{cpo} \rightarrow \mathbf{cpo}$
  - *Flache* Einbettung
- Hier: denotationelle Semantik für **IMP**
  - Programm als *Funktion zwischen Systemzuständen*
  - *Tiefe* Einbettung
- *Systemzustand*: $\Sigma \stackrel{def}{=}$ **Loc** $\rightharpoonup \mathbb{N}$
- Semantische Funktionen:
  - Arithmetische Ausdrücke: $\mathcal{E}$: **Aexp** $\rightarrow (\Sigma \rightarrow \mathbb{N})$
  - Boolsche Ausdrücke: $\mathcal{B}$: **Bexp** $\rightarrow (\Sigma \rightarrow Bool)$
  - Anweisungen: $\mathcal{D}$: **Com** $\rightarrow (\Sigma \rightharpoonup \Sigma)$

Denotationale Semantik für IMP: Anweisungen

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

Strukturelle Operationale Semantik

- *Syntaxgesteuerte* Auswertungsregeln
- Relativ zu einem *Systemzustand*
- Relation $\Rightarrow$ auf *Konfigurationen*: $\langle a, \sigma\rangle$ mit $a \in$ **Aexp**, **Bexp**, **Com**
  - Eigentlich Familie von Relationen $\{\Rightarrow_i\}_{i \in \mathbf{Aexp},\mathbf{Bexp},\mathbf{Com}}$
- Regeln der Form

$$\frac{\langle a_1, \sigma\rangle \Rightarrow \langle a_1', \sigma_n'\rangle \quad \ldots \quad \langle a_n, \sigma\rangle \Rightarrow \langle a_n', \sigma_n'\rangle}{\langle a, \sigma\rangle \Rightarrow \langle a', \sigma'\rangle}\ C$$

ggf. mit Randbedingung C

Operationale Semantik

- Beispielregeln für **Aexp** und **Bexp**:

$$\frac{}{\langle \texttt{X}, \sigma\rangle \Rightarrow \sigma(\texttt{X})} \qquad \frac{\langle a_0, \sigma\rangle \Rightarrow n \quad \langle a_1, \sigma\rangle \Rightarrow m}{\langle a_0\ \texttt{+}\ a_1, \sigma\rangle \Rightarrow n + m}$$

$$\frac{\langle a_0, \sigma\rangle \Rightarrow n \quad \langle a_1, \sigma\rangle \Rightarrow m}{\langle a_0\ \texttt{=}\ a_1, \sigma\rangle \Rightarrow true}\ n = m \qquad \frac{\langle a_0, \sigma\rangle \Rightarrow n \quad \langle a_1, \sigma\rangle \Rightarrow m}{\langle a_0\ \texttt{=}\ a_1, \sigma\rangle \Rightarrow false}\ n \neq m$$

- Beispielregeln für Anweisungen:

$$\frac{\langle a, \sigma\rangle \Rightarrow n}{\langle X := a, \sigma\rangle \Rightarrow \sigma[n/X]} \qquad \frac{\langle b, \sigma\rangle \Rightarrow false}{\langle \texttt{while}\ \ b\ \ \texttt{do}\ \ c\ \texttt{od}, \sigma\rangle \Rightarrow \sigma}$$

$$\frac{\langle b, \sigma\rangle \Rightarrow true \quad \langle c, \sigma\rangle \Rightarrow \tau' \quad \langle \texttt{while}\ \ b\ \ \texttt{do}\ \ c\ \texttt{od}, \tau'\rangle \Rightarrow \tau}{\langle \texttt{while}\ \ b\ \ \texttt{do}\ \ c\ \texttt{od}, \sigma\rangle \Rightarrow \tau}$$