Domain Specific Embedded Languages:

Observable Sharing mit MonadFix
oder
Der Unterschied zwischen Konstruktion und Wert

H.-G. Zirnstein

5. Dezember 2006

Parser combinators

- Parser combinators – Grammatiken in HASKELL spezifizieren

$$
\begin{aligned}
zahl &:: Parser \\
zahl &= (empty \mid\mid\mid zahl) \mathbin{\&\&\&} ziffer \\
ziffer &:: Parser \\
ziffer &= symbol\,'0' \mid\mid\mid \ldots \mid\mid\mid symbol\,'9'
\end{aligned}
$$

- $zahl$ kontextfreie Sprache, besteht aus Zahlen wie "12"
$empty$	leere Sprache,
$symbol\ c$	ein Buchstabe,
$p \mid\mid\mid q$	Vereinigung der Wortmengen,
$p \mathbin{\&\&\&} q$	alle Wortkombinationen uv mit $u \in p, v \in q$

Linkrekursive Grammatiken?

- altbekannte Implementierung: Backtracking Parser combinators

$$
\begin{array}{lll}
\textbf{type}\ Parser & = String \rightarrow [String] \\
empty & = \lambda\, s \rightarrow [[] \mid [] \leftarrow [s]] \\
symbol\ c' & = \lambda\, s \rightarrow [cs \mid (c : cs) \leftarrow [s],\ c == c'] \\
p \mid\mid\mid q & = \lambda\, s \rightarrow p\ s +\!\!+ q\ s \\
p\ \&\&\&\ q & = \lambda\, s \rightarrow [u \mid t \leftarrow p\ s,\ u \leftarrow q\ t]
\end{array}
$$

- $n :: Parser =$ Wort $w \mapsto$ [Suffix von w]
  Fehlende Präfixe $=$ Wörter $\in n$ die eben Präfixe von w sind
- Problem: nicht für **linksrekursive** Grammatiken wie $zahl$

$$
\begin{array}{rl}
zahl\ "12" & \Longrightarrow [u \mid t \leftarrow (empty \mid\mid\mid zahl)\ "12",\ u \leftarrow ziffer\ t] \\
& \Longrightarrow [u \mid t \leftarrow (zahl\ "12"),\ u \leftarrow ziffer\ t] \\
& \Longrightarrow [u \mid t \leftarrow (\ldots ??\ \ldots
\end{array}
$$

Abstrakte Grammatik

- Idee: mit abstrakter Beschreibung der Grammatik könnten wir Linksrekursion in Rechtsrekursion umschreiben

$$\begin{aligned}
&\textbf{data}\ Parser = Empty \mid Symbol\ Char \\
&\qquad\qquad \mid\ Or\ Parser\ Parser \mid Cat\ Parser\ Parser \\
&\&\&\& \qquad = Cat \quad \text{— usw.}
\end{aligned}$$

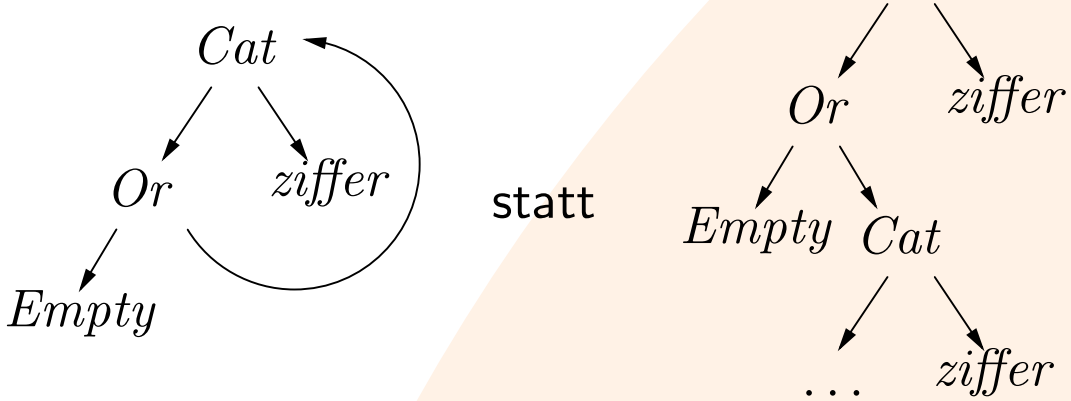
- aber

$$\begin{aligned}
zahl\ &=\ (empty \mid\mid\mid zahl)\ \&\&\&\ ziffer \\
&\Longrightarrow Cat\ (Or\ Empty\ zahl)\ ziffer \\
&\Longrightarrow Cat\ (Or\ Empty\ (Cat\ (Or\ Empty\ (\ldots))\ ziffer)\ ziffer
\end{aligned}$$

- $zahl$ ist eine **unendliche** Datenstruktur!
- und das auch bei Rechtsrekursion!

Sharing, Observable Sharing

- $zahl$ wird intern als Graph gespeichert



- "Sharing" := im linken Graph ist $zahl$ eine Refrenz auf genau ein Objekt im Speicher, im rechten nicht
- "Observable Sharing" := erkennen, dass linker und rechter Graph verschieden sind, z.B. indem Speicheradressen verglichen werden

$$\textbf{??}\ pointerEqual :: a \to a \to Bool\ \textbf{??}$$

- **Nicht** in Haskell. Wehe euch!

Explizit Namen vergeben

- Lösung: wir nennen die Dinge beim Namen, bzw. geben ihnen eine Id

$$
\begin{aligned}
&\textbf{type } Id &&= Integer \\
&\textbf{data } NT &&= NT\ Id\ Regel \quad \text{— Nichtterminale} \\
&\textbf{data } Regel &&= Empty \mid Symbol\ Char \mid Or\ Id\ Id \mid Cat\ Id\ Id \\
&\textbf{data } Parser &&= [NT]
\end{aligned}
$$

- Id macht sozusagen die Speicheradressen explizit.
- Beispielgrammatik sehr **unschön** wiefolgt

$$
\begin{aligned}
empty &= NT\ (-1)\ Empty \\
z0 &= NT\ 0\ (Symbol\ '0') \ \ldots\ z9 = NT\ 9\ (Symbol\ '9') \\
z0bis1 &= NT\ 10\ (Or\ 1\ 0) \ \ldots\ z0bis9 = NT\ 19\ (Or\ 9\ 18) \\
maybezahl &= NT\ 20\ (Or\ 21\ (-1))\ \ zahl = NT\ 21\ (Cat\ 20\ 19) \\
grammatik &= [z0, \ldots, z9, z0bis0, \ldots, z0bis9, maybezahl, zahl]
\end{aligned}
$$

Monade verteilt Namen

- Probleme: doppelte Bennnung, Eindeutigkeit der Namen
- Lösung: Namen über eine Monade verteilen

$$
\begin{array}{l}
\textbf{do} \\
\quad z0 \leftarrow neueId\ \$\ Symbol\ '0' \quad \ldots \\
\quad z0bis1 \leftarrow neueId\ \$\ Or\ z0\ z1 \quad \ldots \\
\quad ziffer \leftarrow neueId\ \$\ Or\ z0bis8\ z9
\end{array}
$$

- "$\leftarrow\ neueId\ \$$" einfach als "$=$" lesen
- Übersetzung der **do**-Syntax:

  "$var \leftarrow\ action$" wird "$action\ \gg\!=\ (\lambda\, var\ \rightarrow \ldots)$"

  d.h. Ids werden mit dem ultimativen Namensgeber λ benannt!
- Namenserzeugung per Monade immer noch first-class

$$
\begin{array}{l}
zks \leftarrow mapM\ (neueId\ .\ Symbol)\ ['0'..'9'] \\
z0bisks \leftarrow foldM\ ((neueId\ .)\ .\ Or)\ empty\ zks
\end{array}
$$

MonadFix

- Rekursion? **mdo**!

  mdo
  $$maybezahl \leftarrow neueId\ \$\ Or\ empty\ zahl$$
  $$zahl \leftarrow neueId\ \$\ Cat\ maybezahl\ ziffer$$

- Wie **do**, aber Variablen vor ihrer Deklaration verwendbar
- Magie steckt in
  $$mfix :: MonadFix\ m \Rightarrow (a \rightarrow m\ a) \rightarrow m\ a$$
  Trick: $mfix\ f$ steckt den **Wert** hinten wieder vorne rein, ohne die **Konstruktion** f mehrmals auszuführen
  $$mfix\ f \text{ ist } \textbf{nicht } fix\ (f\ \gg\!=) = (f\ \gg\!=\ f\ \gg\!=\ f\ \gg\!=\ \ldots)$$

- **mdo** oben wird übersetzt in
  $$mfix\ (\lambda\ (zahl, maybezahl)\ \rightarrow\ \ldots\ \gg\ return\ (zahl', maybezahl')\ )$$

Real Life - Frisby

- **Frisby** `http://repetae.net/computer/frisby/`.
- Compositional Parsers für kontextfrei-ähnliche Grammatiken
- Beispielgrammatik für arithmetische Ausdrücke mit $+$ und $*$

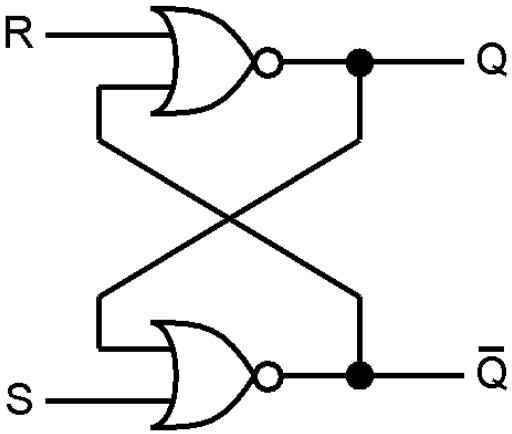
$$
\begin{aligned}
&additive = \mathbf{mdo} \\
&\quad additive \leftarrow newRule\ \$\ multitive \diamond char\ '+' \succcurlyeq additive\ \#\#\ \ldots \\
&\quad multitive \leftarrow newRule\ \$\ primary \diamond char\ '*' \succcurlyeq multitive\ \#\#\ \ldots \\
&\quad primary \leftarrow newRule\ \$\ char\ '(' \succcurlyeq additive \preccurlyeq char\ ')' \mathbin{//} decimal \\
&\quad decimal \leftarrow newRule\ \$\ many1\ (oneOf\ ['0' .. '9'])\ \#\#\ read \\
&\quad return\ additive
\end{aligned}
$$

- Linksrekursion geht hier übrigens nicht
- Observable Sharing wegen Memoization

Real Life - Lava

- **Lava** = DSEL zu Beschreibung von Schaltkreisen
- Beispiel (früher)

$$
\begin{aligned}
&srLatch\ s\ r = \mathbf{mdo} \\
&\quad q \leftarrow nor\ r\ q' \\
&\quad q' \leftarrow nor\ s\ q \\
&\quad return\ (q, q')
\end{aligned}
$$



- Lava heute: `http://raintown.org/lava/`, `http://www.cs.chalmers.se/~koen/Lava/`
- heute: etwas schwächere Version von Zeiger vergleichen, macht referentielle Transparenz trotzdem kaputt
- für Observable Sharing sind Monaden zuviel $\Longrightarrow$ besseren *Syntactic Sugar* einführen?