Diskussion:

- Die Code-Erzeugung muss schnell gehn :-)
- Anstelle für jeden Knoten neu zu überprüfen, wie die Regeln zusammen passen, kann die Berechnung auch in einen endlichen Automaten kompiliert werden :-))

Ein deterministischer endlicher Baumautomat (DTA) A besteht aus:

Q	$==$	endliche Menge von Zuständen
Σ	$==$	Operatoren und Konstanten
δ_a	$==$	Übergangsfunktion für $a \in \Sigma$
$F \subseteq Q$	$==$	akzeptierende Zustände

Dabei ist:

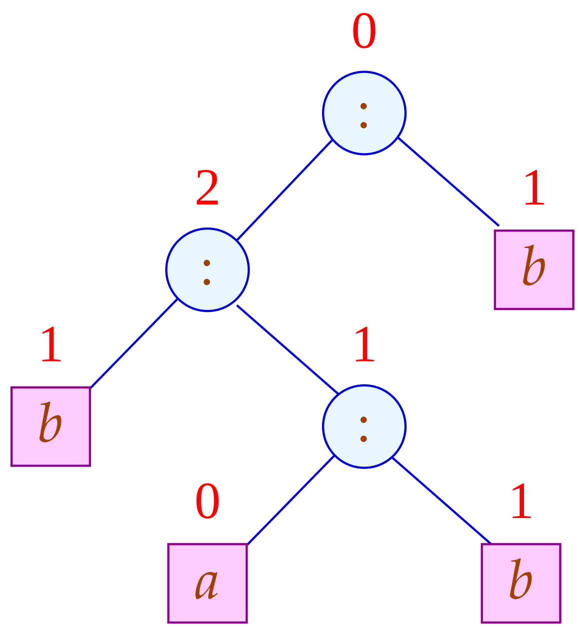
$$\begin{array}{llll} \delta_c : & Q & \text{falls} \quad c & \text{Konstante} \\ \delta_a : & Q^k \to Q & \text{falls} \quad a & k\text{-stellig} \end{array}$$

Beispiel:

$$\begin{array}{lllllll} Q & = & \{0,1,2\} & \quad & F & = & \{0\} \\ \Sigma & = & \{a, b, :\} & & & & \\ \delta_a & = & 0 & & \delta_b & = & 1 \end{array}$$

$$\delta_:(s_1, s_2) \;=\; (s_1 + s_2)\%3$$

// akzeptiert alle Bäume mit $3 \cdot k$ b-Blättern



Der Zustand an einem Knoten a ergibt sich aus den Zuständen der Kinder mittels δ_a (-:

$$\begin{array}{lll} Q(c) & = & \delta_c \\ Q(a(t_1, \ldots, t_k)) & = & \delta_a(Q(t_1), \ldots, Q(t_k)) \end{array}$$

Die von A definierte Sprache (oder: Menge von Bäumen) ist:

$$\mathcal{L}(A) = \{t \mid Q(t) \in F\}$$

... in unserer Anwendung:

$$\begin{array}{lcl}
Q & == & \text{Teilmengen von } \mathsf{Reg} \cup \mathsf{Term} \cup \{S\}
\end{array}$$

// I.a. werden nicht sämtiche Teilmengen benötigt :-)

$$\begin{array}{lcl}
F & == & \text{gewünschter Effekt} \\
\delta_R & == & \mathsf{Move}\,\{R\} \\
\delta_c & == & \mathsf{Move}\,\{c\} \\
\delta_a(Q_1, \ldots, Q_k) & == & \mathsf{Move}\,\{s = a(s_1, \ldots, s_k) \in \mathsf{Term} \mid s_i \in Q_i\}
\end{array}$$

... im Beispiel:

$$
\begin{array}{rclcl}
\delta_c & = & \{A, D\} & = & q_0 \\
 & = & \delta_A & & \\
 & = & \delta_D & &
\end{array}
$$

$$
\begin{array}{rclcl}
\delta_+(q_0, q_0) & = & \{A, D, A + A\} & = & q_1 \\
 & = & \delta_+(q_0, \_) & & \\
 & = & \delta_+(\_, q_0) & &
\end{array}
$$

$$
\begin{array}{rclcl}
\delta_M(q_0) & = & \{A, D\} & = & q_0 \\
 & = & \delta_M(q_1) & &
\end{array}
$$

Um die Anzahl der Zustände zu reduzieren, haben wir die vollständigen rechten Seiten, die keine echten Teilmuster sind, in den Zuständen weggelassen :-)

Integration der Kostenberechnung:

Problem:

Kosten können (im Prinzip) beliebig groß werden ;-(

Unser FTA besitzt aber nur endlich viele Zustände :-((

Idee:

Pelegri-Lopart 1988

Betrachte nicht absolute Kosten — sondern relative !!!

Eduardo Pelegri-Llopart,
Sun Microsystems, Inc.

Beobachtung:

- In gängigen Prozessoren kann man Werte von jedem Register in jedes andere schieben $\Longrightarrow$

  Die Kosten zwischen Registern differieren nur um eine Konstante :-)

- Komplexe rechte Seiten lassen sich i.a. mittels elementarerer Instruktionen simulieren $\Longrightarrow$

  Die Kosten zwischen Teilausdrücken und Registern differieren nur um eine Konstante :-))

- Die Kostenberechnung ist additiv $\Longrightarrow$

  Wir können statt mit absoluten Kosten-Angaben auch mit Kosten-Differenzen rechnen !!!

  Von diesen gibt es nur endlich viele :-)

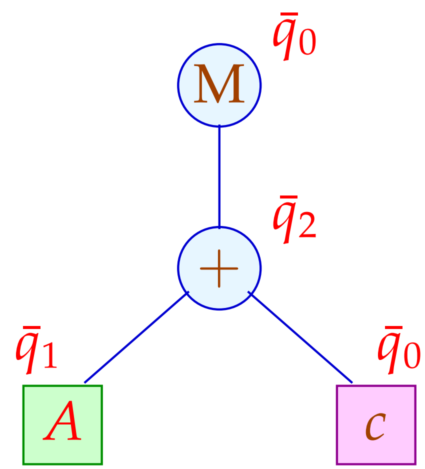
... im Beispiel:

$$\begin{array}{rcl} \delta_c &=& \{A \mapsto 1, D \mapsto 0\} \quad = \quad \bar{q}_0 \\ &=& \delta_D \\ \delta_A &=& \{A \mapsto 0, D \mapsto 1\} \quad = \quad \bar{q}_1 \end{array}$$

$$\begin{array}{rcl} \delta_+(\bar{q}_1, \bar{q}_0) &=& \{A \mapsto 2, D \mapsto 1, A + A \mapsto 0\} \quad = \quad \bar{q}_2 \\ \delta_+(\bar{q}_0, \bar{q}_0) &=& \{A \mapsto 1, D \mapsto 0, A + A \mapsto 1\} \quad = \quad \bar{q}_3 \\ \delta_+(\bar{q}_1, \bar{q}_1) &=& \{A \mapsto 4, D \mapsto 3, A + A \mapsto 0\} \quad = \quad \bar{q}_4 \\ && \ldots \end{array}$$

$$\begin{array}{rcl} \delta_M(\bar{q}_2) &=& \{A \mapsto 1, D \mapsto 0\} \quad = \quad \bar{q}_0 \\ &=& \delta_M(\bar{q}_i) \qquad , \quad i = 0, \ldots, 4 \end{array}$$

... das liefert die folgende Berechnung:



Für jede Konstanten-Klasse c und jedes Register R in δ_c tabellieren wir die zu wählende billigste Berechnung:

$$c \; : \quad \{A \mapsto 5, 3, D \mapsto 3\}$$

Analog tabellieren wir für jeden Operator a , jedes $\tau \in \bar{Q}^k$ und jedes R in $\delta_a(\tau)$:

M	select_M
$\bar{q}_0$	$\{A \mapsto 5, 1, D \mapsto 1\}$
$\bar{q}_1$	$\{A \mapsto 5, 1, D \mapsto 1\}$
$\bar{q}_2$	$\{A \mapsto 5, 0, D \mapsto 0\}$
$\bar{q}_3$	$\{A \mapsto 5, 1, D \mapsto 1\}$
$\bar{q}_4$	$\{A \mapsto 5, 0, D \mapsto 0\}$

Für “$+$” ist die Tabelle besonders einfach:

$+$	$\bar{q}_j$
$\bar{q}_i$	$\{A \mapsto 5, 3, D \mapsto 3\}$

Problem:

- Für reale Instruktionssätze benötigt man leicht um die 1000 Zustände.
- Die Tabellen für mehrstellige Operatoren werden riesig :-(

$\Longrightarrow$ Wir benötigen Verfahren der Tabellen-Komprimierung ...

Tabellen-Kompression:

Die Tabelle für "$+$" sieht im Beispiel so aus:

$+$	$\bar{q}_0$	$\bar{q}_1$	$\bar{q}_2$	$\bar{q}_3$	$\bar{q}_4$
$\bar{q}_0$	$\bar{q}_3$	$\bar{q}_2$	$\bar{q}_3$	$\bar{q}_3$	$\bar{q}_3$
$\bar{q}_1$	$\bar{q}_2$	$\bar{q}_4$	$\bar{q}_2$	$\bar{q}_2$	$\bar{q}_2$
$\bar{q}_2$	$\bar{q}_3$	$\bar{q}_2$	$\bar{q}_3$	$\bar{q}_3$	$\bar{q}_3$
$\bar{q}_3$	$\bar{q}_3$	$\bar{q}_2$	$\bar{q}_3$	$\bar{q}_3$	$\bar{q}_3$
$\bar{q}_4$	$\bar{q}_3$	$\bar{q}_2$	$\bar{q}_3$	$\bar{q}_3$	$\bar{q}_3$

Die meisten Zeilen / Spalten sind offenbar ganz ähnlich ;-)

Idee 1: Äquivalenzklassen

Wir setzen $q \equiv_a q'$, genau dann wenn

$$\forall\, p: \quad \delta_a(q,p) = \delta_a(q',p) \quad \wedge \quad \delta_a(p,q) = \delta_a(p,q')$$
$$\wedge \quad \mathsf{select}_a(q,p) = \mathsf{select}_a(q',p) \quad \wedge \quad \mathsf{select}_a(p,q) = \mathsf{select}_a(p,q')$$

Im Beispiel:

$$Q_1 = \{\bar{q}_0, \bar{q}_2, \bar{q}_3, \bar{q}_4\}$$
$$Q_2 = \{\bar{q}_1\}$$

mit:

$+$	Q_1	Q_2
Q_1	$\bar{q}_3$	$\bar{q}_2$
Q_2	$\bar{q}_2$	$\bar{q}_4$

Idee 2: Zeilenverschiebung

Sind viele Einträge gleich (im Beispiel etwa $\mathsf{default} = \bar{q}_3$), genügt es, die übrigen Einträge zu speichern ;-)

Im Beispiel:

$+$	$\bar{q}_0$	$\bar{q}_1$	$\bar{q}_2$	$\bar{q}_3$	$\bar{q}_4$
$\bar{q}_0$		$\bar{q}_2$			
$\bar{q}_1$	$\bar{q}_2$	$\bar{q}_4$	$\bar{q}_2$	$\bar{q}_2$	$\bar{q}_2$
$\bar{q}_2$		$\bar{q}_2$			
$\bar{q}_3$		$\bar{q}_2$			
$\bar{q}_4$		$\bar{q}_2$			

Dann legen wir:

(1) gleiche Zeilen übereinander;

(2) verschiedene (Klassen von) Zeilen auf Lücke verschoben übereinander:

	$\bar{q}_0$	$\bar{q}_1$	$\bar{q}_2$	$\bar{q}_3$	$\bar{q}_4$
class	0	1	0	0	0

	0	1
disp	0	2

	0	1	2	3	4	5	6
A	$\bar{q}_2$	$\bar{q}_2$	$\bar{q}_4$	$\bar{q}_2$	$\bar{q}_2$	$\bar{q}_2$	$\bar{q}_2$
valid	0	0	1	1	1	1	1

Für jeden Eintrag im ein-dimensionalen Feld A vermerken wir in valid , zu welcher Zeile der Eintrag gehört ...

Ein Feld-Zugriff $\delta_+(\bar{q}_i, \bar{q}_j)$ wird dann so realisiert:

$$\begin{array}{lll} \delta_+(\bar{q}_i, \bar{q}_j) & = & \mathsf{let} \quad c \;=\; \mathsf{class}[\bar{q}_i]; \\ & & \qquad\;\; d \;=\; \mathsf{disp}[c]; \\ & & \mathsf{in}\;\; \mathsf{if}\;\; (\mathsf{valid}[d + j] \equiv c) \\ & & \qquad \mathsf{then}\;\; \mathsf{A}[d + j] \\ & & \qquad \mathsf{else}\;\; \mathsf{default} \\ & & \mathsf{end} \end{array}$$

Reinhard Wilhelm, Saarbrücken

Diskussion:

- Die Tabellen werden i.a. erheblich kleiner.
- Dafür werden Tabellenzugriffe etwas teurer.
- Das Verfahren versagt in einigen (theoretischen) Fällen.
- Dann bleibt immer noch das dynamische Verfahren ...

  möglicherweise mit Caching der einmal berechneten Werte, um unnötige Mehrfachberechnungen zu vermeiden :-)

7 Instruction Level Parallelität

Moderne Prozessoren führen nicht eine Instruktion nach der anderen aus.

Wir betrachten hier zwei Ansätze:

(1) VLIW (Very Large Instruction Words)
(2) Pipelining

VLIW:

Eine Instruktion führt simultan bis zu k (etwa 4:-) elementare Instruktionen aus.

Pipelining:

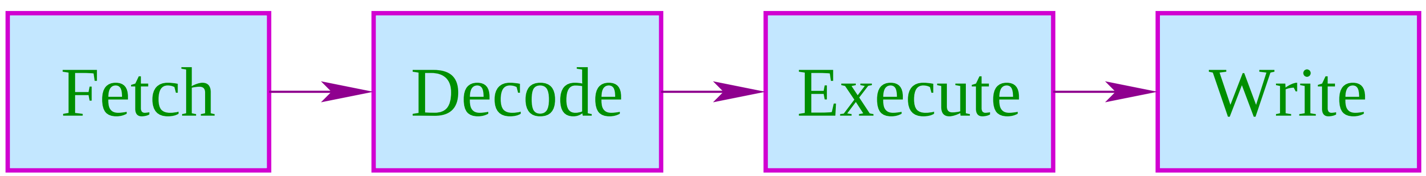
Instruktionsausführungen können zeitlich überlappen.

Beispiel:

$$w = (R_1 = R_2 + R_3 \mid D = D_1 * D_2 \mid R_3 = M[R_4])$$

Achtung:

- Instruktionen belegen Hardware-Einrichtungen.
- Instruktionen greifen auf die gleichen Register zu $\Longrightarrow$ Hazards
- Ergebnisse einer Insgtruktion liegen erst nach einiger Zeit vor.
- Während dieser Zeit wechselt i.a. die benutzte Hardware:



- Während Execute bzw. Write werden evt. unterschiedliche interne Register/Busse/Alus benutzt.

Wir schließen:

Aufteilung der Instruktionsfolge in Wörter und ihre Aufeinanderfolge ist Restriktionen unterworfen ...

Im folgenden ignorieren wir die Phasen Fetch und Decode :-)

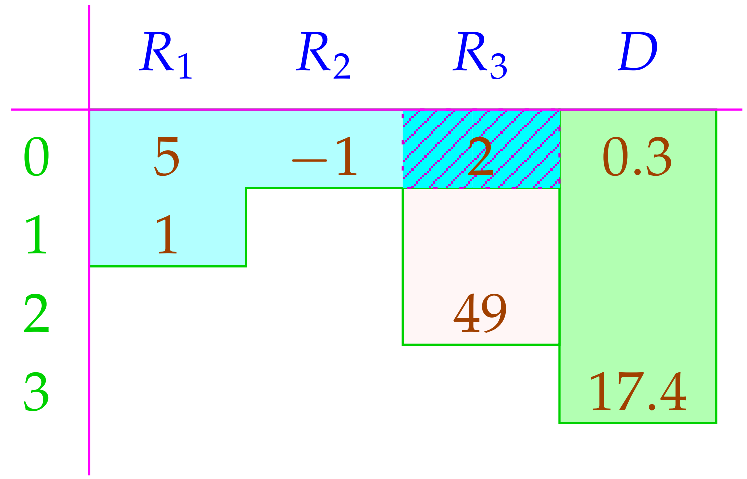
Beispiele für Restriktionen:

(1) maximal ein Load/Store pro Wort;
(2) maximal ein Jump;
(3) maximal ein Write in das selbe Register.

Timing:

Gleitkomma-Operation	3
Laden/Speichern	2
Integer-Arithmetik	1

Timing-Diagramm:



R_3 wird überschrieben, nachdem die Addition 2 abgeholte :-)