3.2 Instruktionen

Problem:

- unregelmäßige Instruktionssätze ...
- mehrere Adressierungsarten, die evt. mit arithmetischen Operationen kombiniert werden können;
- Register für unterschiedliche Verwendungen ...

Beispiel: Motorola MC68000

Dieser einfachste Prozessor der 680x0-Reihe besitzt

- 8 Daten- und 8 Adressregister;
- eine Vielzahl von Adressierungsarten ...

Notation	Beschreibung	Semantik
D_n	Datenregister direkt	D_n
A_n	Adressregister direkt	A_n
(A_n)	Adressregister indirekt	$M[A_n]$
$d(A_n)$	Adressregister indirekt mit Displacement	$M[A_n + d]$
$d(A_n, D_m)$	Adressregister indirekt mit Index und Displacement	$M[A_n + D_m + d]$
x	Absolut kurz	$M[x]$
x	Absolut lang	$M[x]$
$\#x$	Unmittelbar	x

- Der MC68000 ist eine 2-Adress-Maschine, d.h. ein Befehl darf maximal 2 Adressierungen enthalten. Die Instruktion:

  $$\mathsf{add}\;\; D_1 \;\; D_2$$

  addiert die Inhalte von D_1 und D_2 und speichert das Ergebnis nach und D_2 :-)

- Die meisten Befehle lassen sich auf Bytes, Wörter (2 Bytes) oder Doppelwörter (4 Bytes) anwenden.

  Das unterscheiden wir durch Anhängen von .B, .W, .D (Default: .W)

- Die Ausführungszeit eines Befehls ergibt sich (i.a.) aus den Kosten der Operation plus den Kosten für die Adressierung der Operanden ...

	Adressierungsart	Byte / Wort	Doppelwort
D_n	Datenregister direkt	0	0
A_n	Adressregister direkt	0	0
(A_n)	Adressregister indirekt	4	8
$d(A_n)$	Adressregister indirekt mit Displacement	8	12
$d(A_n, D_m)$	Adressregister indirekt mit Index und Displacement	10	14
x	Absolut kurz	8	12
x	Absolut lang	12	16
$\#x$	Unmittelbar	4	8

Beispiel:

Die Instruktion:	move.B $8(A_1, D_1.\text{W}), D_5$
benötigt:	$4 + 10 + 0 = 14$ Zyklen

Alternativ könnten wir erzeugen:

adda	$\#8, A_1$	Kosten:	$8 + 8 + 0 = 16$
adda	$D_1.\text{W}, A_1$	Kosten:	$8 + 0 + 0 = 8$
move.B	$(A_1), D_5$	Kosten:	$4 + 4 + 0 = 8$

mit Gesamtkosten 32 oder:

adda	$D_1.\text{W}, A_1$	Kosten:	$8 + 0 + 0 = 8$
move.B	$8(A_1), D_5$	Kosten:	$4 + 8 + 0 = 12$

mit Gesamtkosten 20 :-)

Achtung:

- Die verschieden Code-Sequenzen sind im Hinblick auf den Speicher und das Ergebnis äquivalent !
- Sie unterscheiden sich im Hinblick auf den Wert des Registers A_1 sowie die gesetzten Bedingungs-Codes !!
- Ein schlauer Instruktions-Selektor muss solche Randbedingungen berücksichtigen :-)

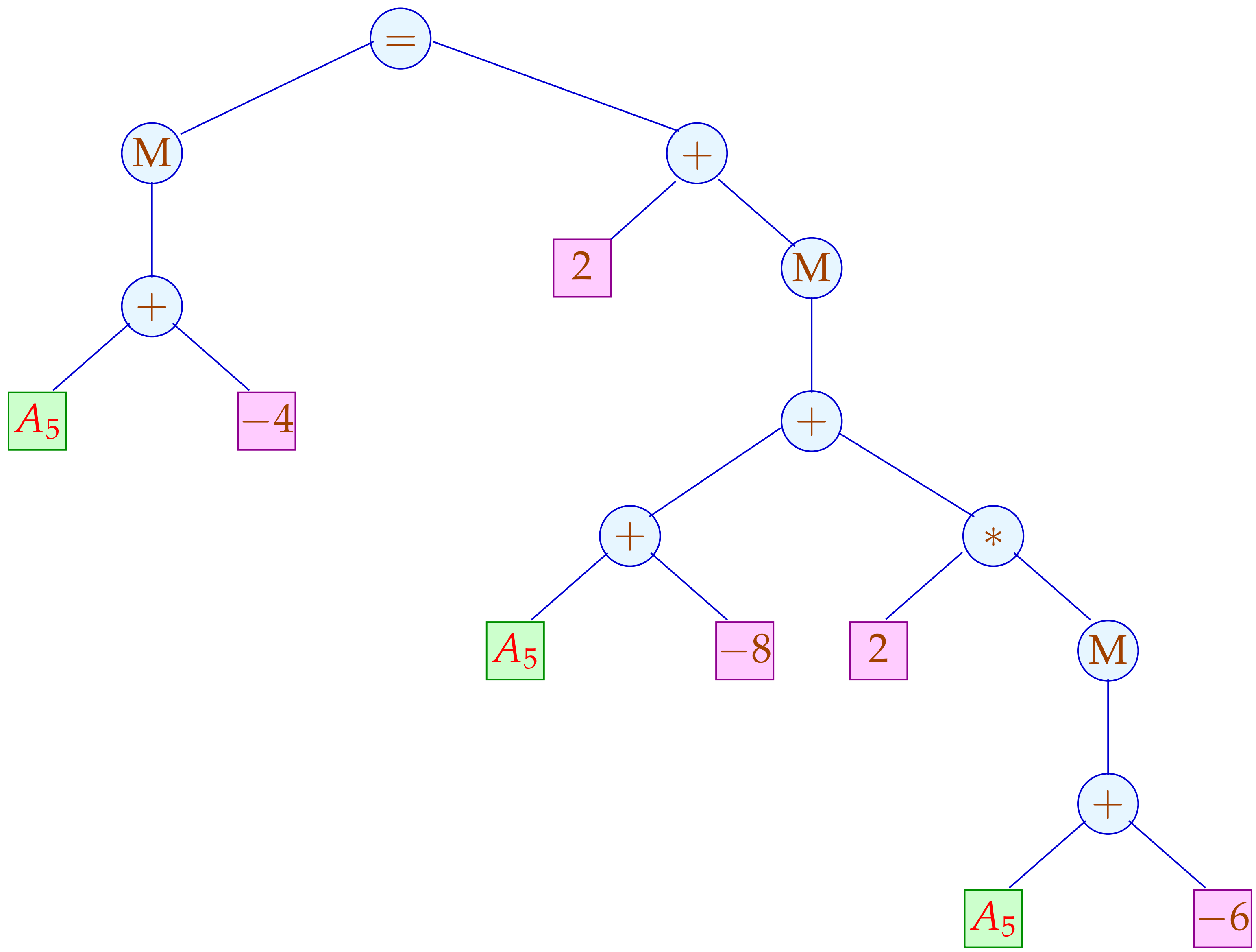
Etwas größeres Beispiel:

$$\text{int } b,\ i,\ a[100];$$
$$b = 2 + a[i];$$

Nehmen wir an, die Variablen werden relativ zu einem Framepointer A_5 mit den Adressen $-4, -6, -8$ adressiert. Dann entspricht der Zuweisung das Stück Zwischen-Code:

$$M[A_5 - 4] = 2 + M[A_5 - 8 + 2 \cdot M[A_5 - 6]];$$

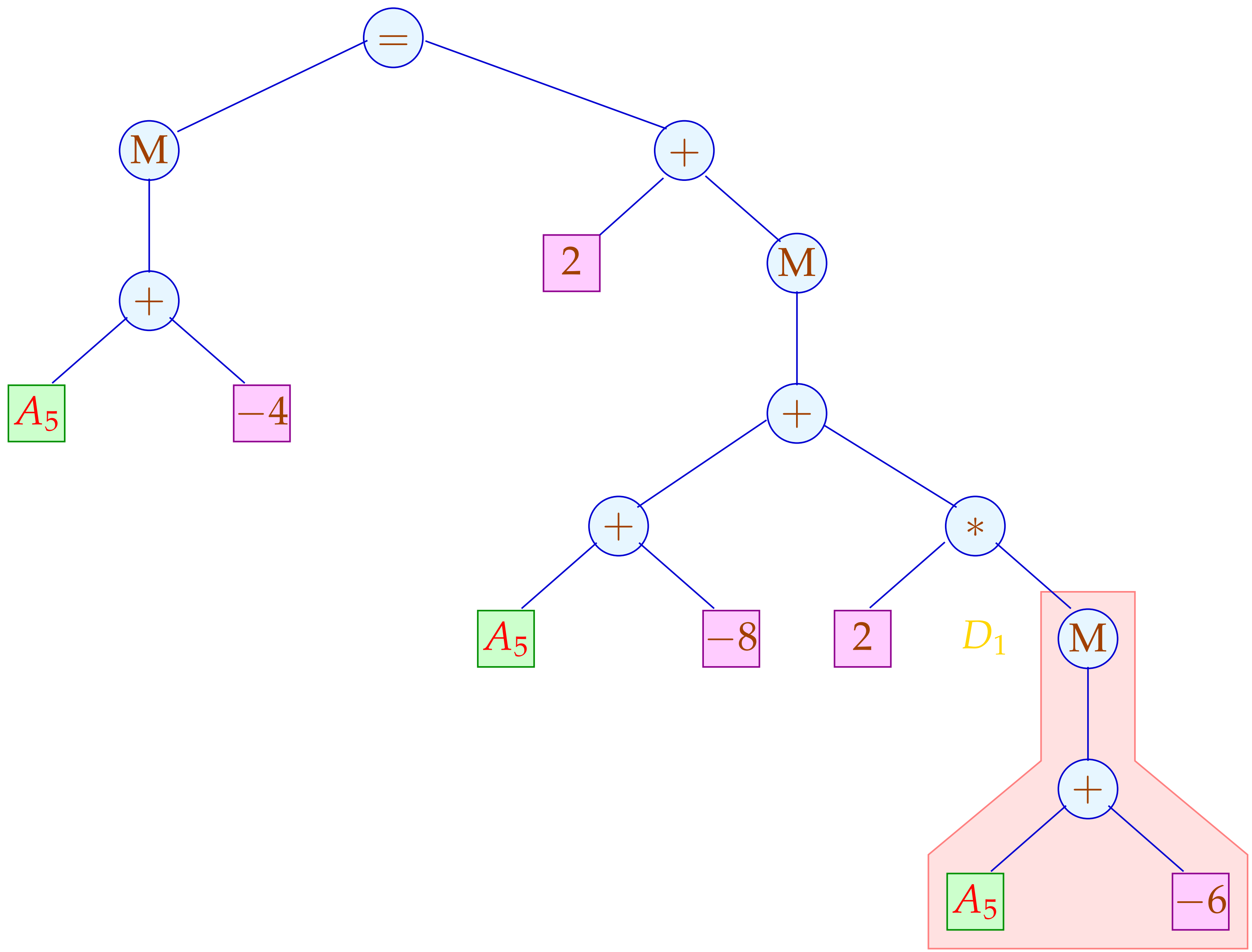
Das entspricht dem Syntaxbaum:

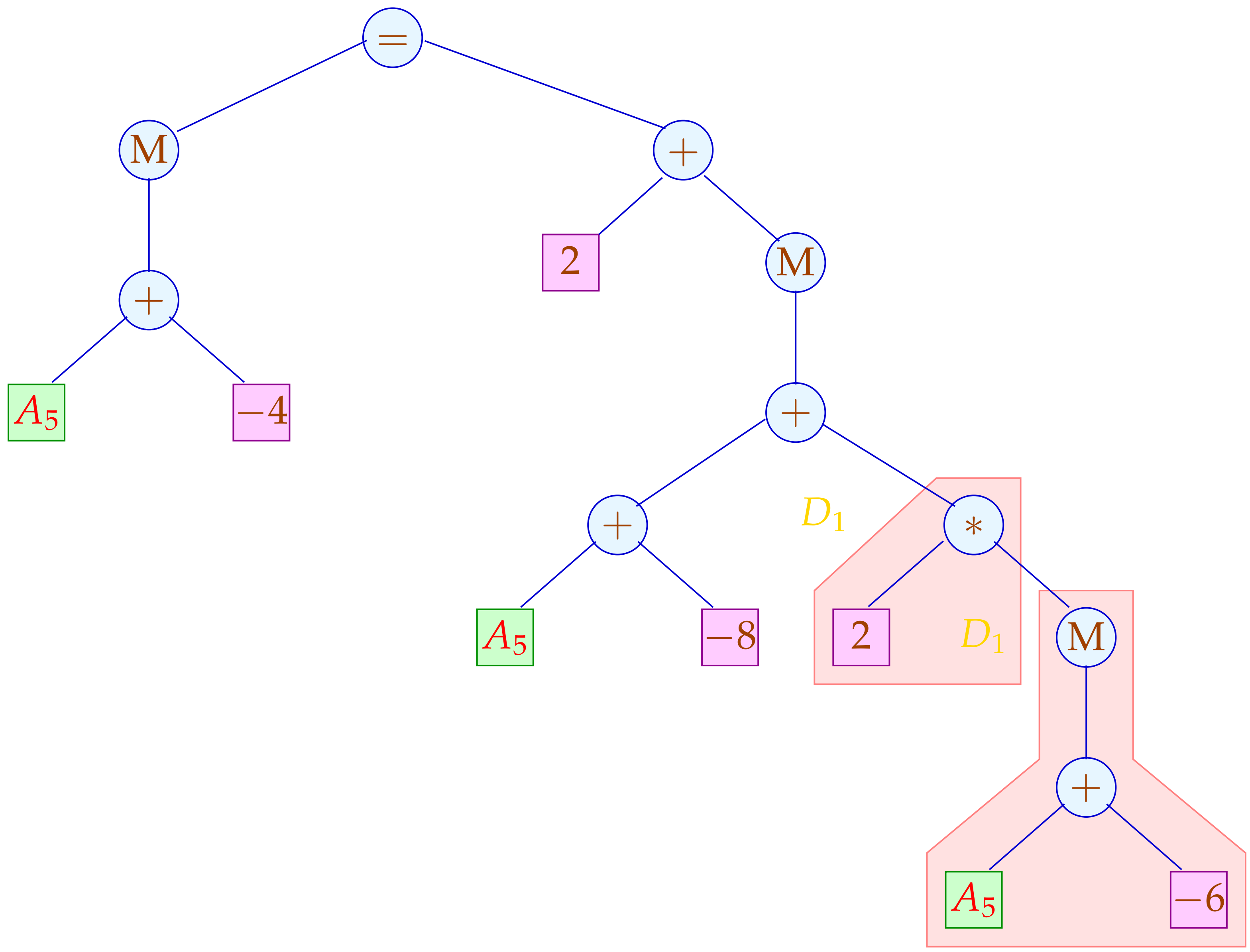

=
M
+
A_5
−4
+
2
M
+
+
A_5
−8
*
2
M
+
A_5
−6

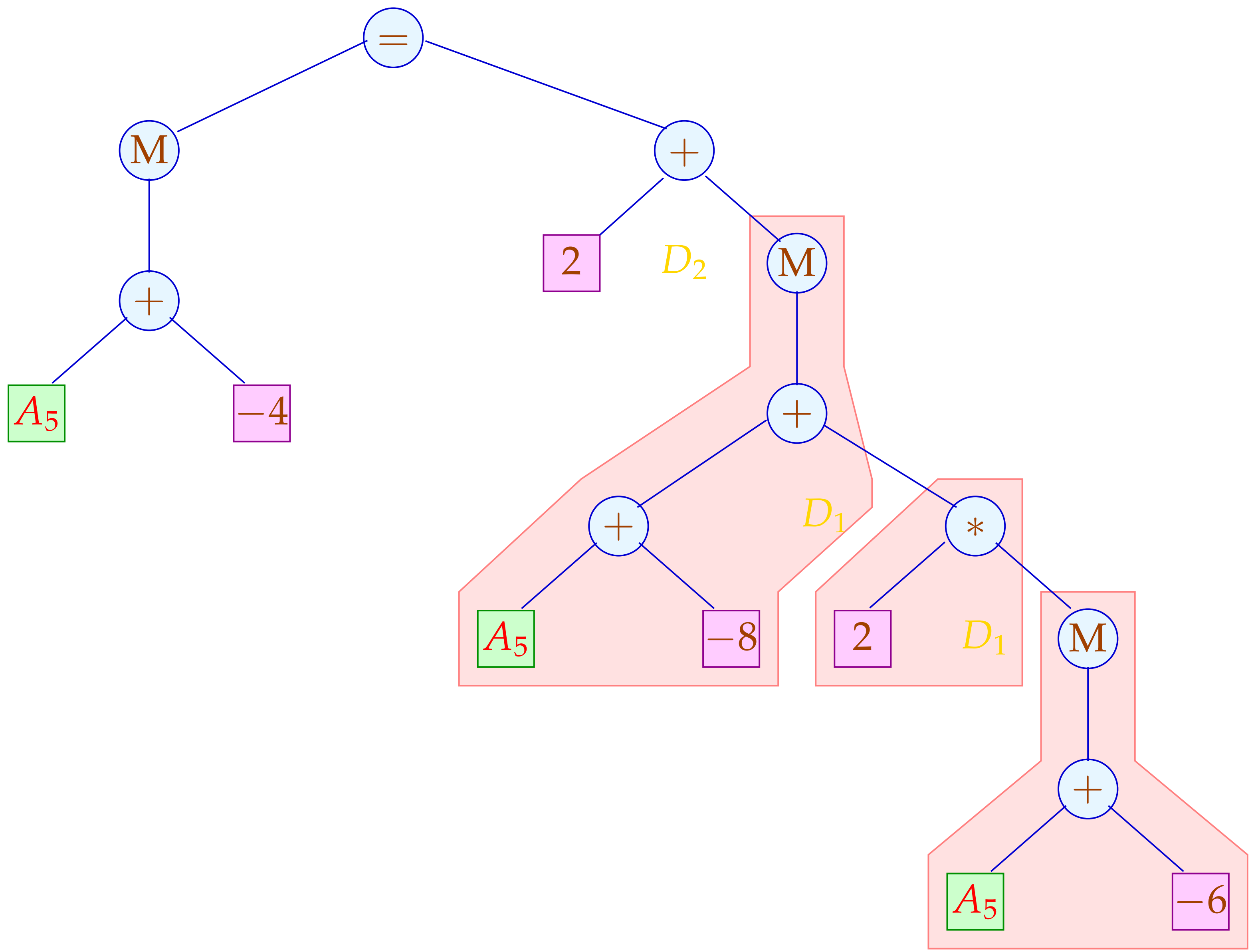
Eine mögliche Code-Sequenz:

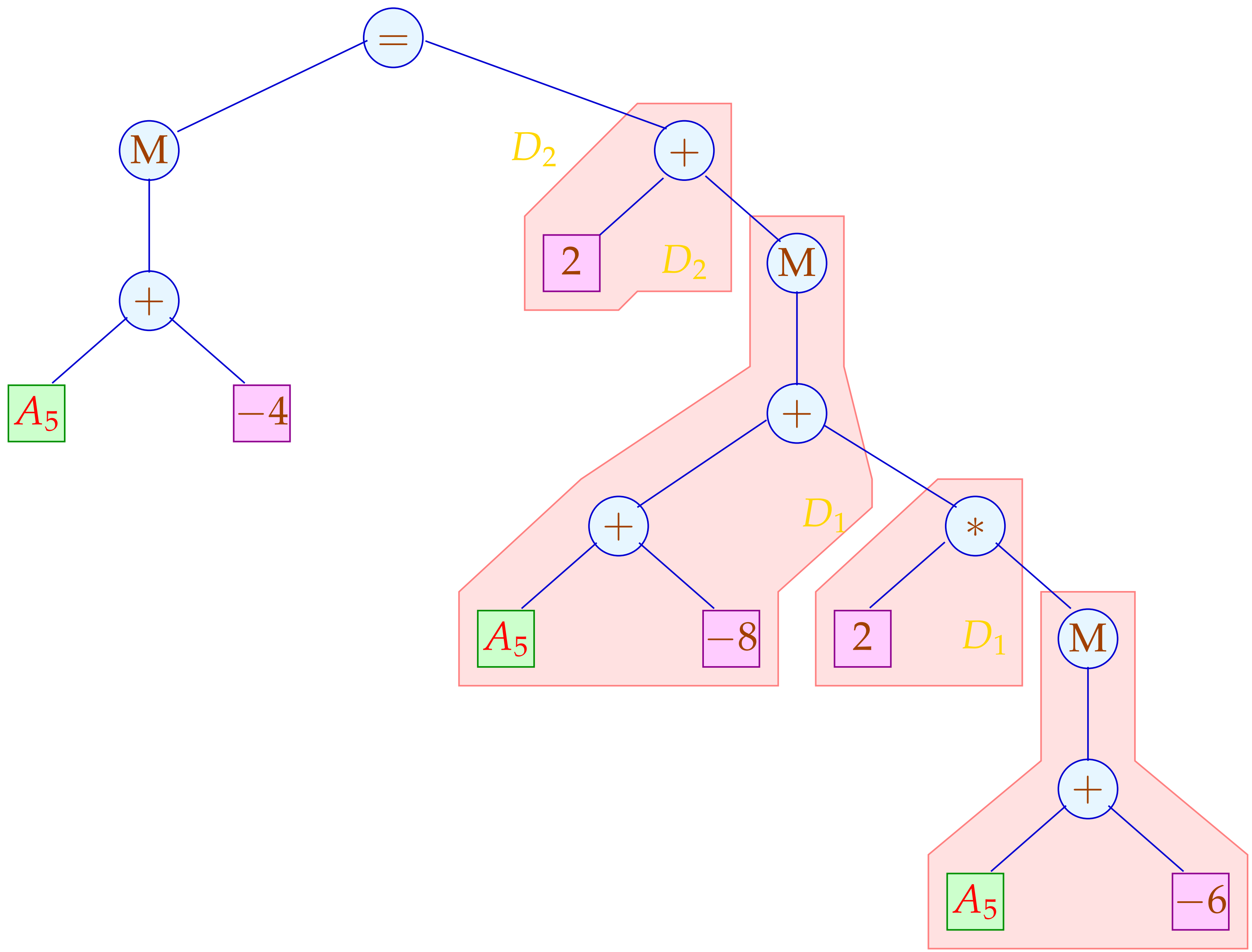
move	$-6(A_5),\ D_1$	Kosten:	12
add	$D_1,\ D_1$	Kosten:	4
move	$-8(A_5, D_1),\ D_2$	Kosten:	14
addq	$\#2,\ D_2$	Kosten:	4
move	$D_2,\ -4(A_5)$	Kosten:	12

Gesamtkosten : 46

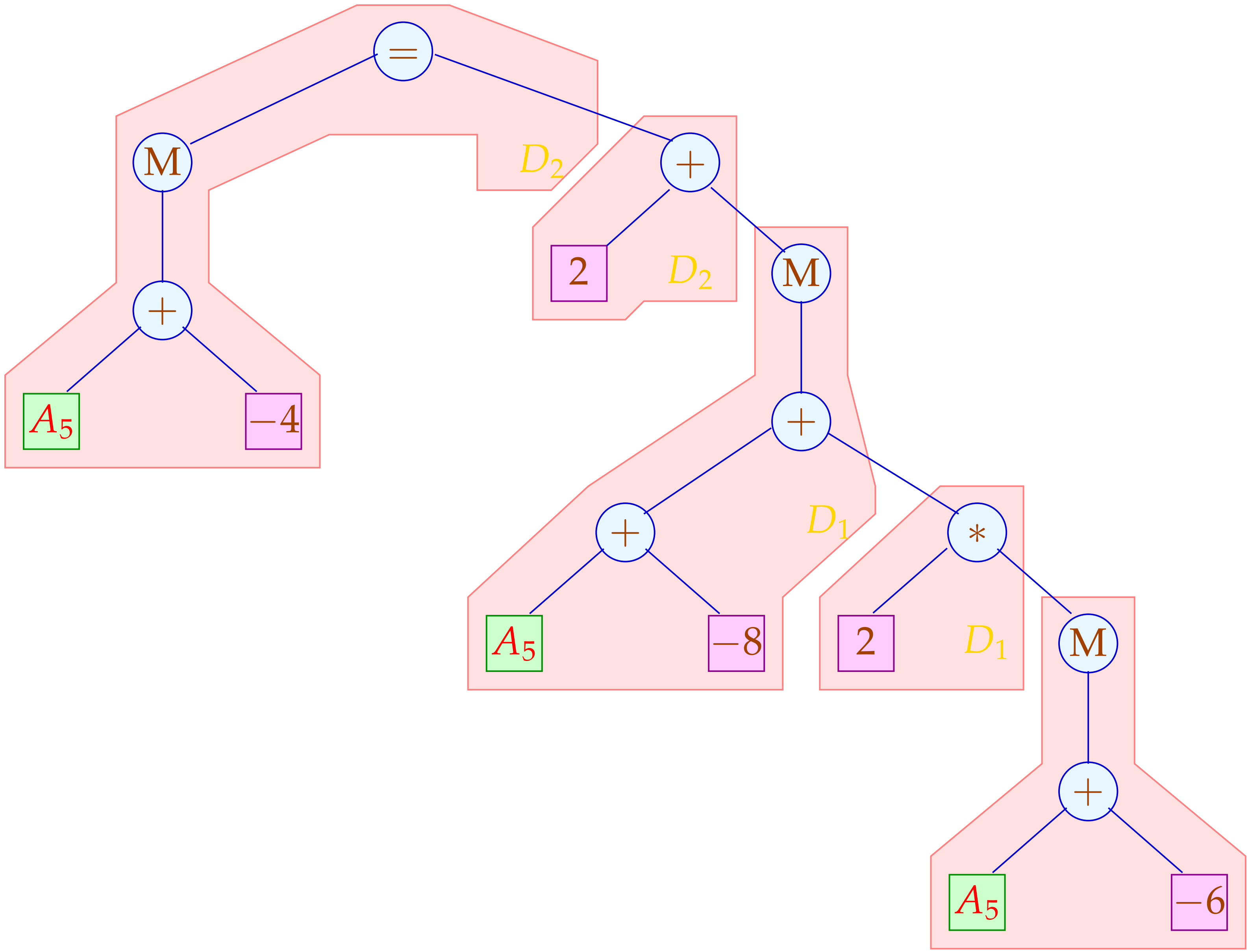

=
M
+
+
2
M
A5
−4
+
+
*
A5
−8
2
D1
M
+
A5
−6


=
M
+
2
M
+
A_5
-4
+
D_1
*
A_5
-8
2
D_1
M
+
A_5
-6


=
M
+
2
D_2
M
+
A_5
−4
+
*
D_1
+
A_5
−8
2
D_1
M
+
A_5
−6



=
M
+
A_5
-4
D_2
+
2
D_2
M
+
+
A_5
-8
D_1
*
2
D_1
M
+
A_5
-6


=
M
+
A_5
-4
D_2
+
2
D_2
M
+
+
A_5
-8
D_1
*
2
D_1
M
+
A_5
-6

Eine alternative Code-Sequenz:

move.L	$A_5,\ A_1$	Kosten:	4
adda.L	$\#-6,\ A_1$	Kosten:	12
move	$(A_1),\ D_1$	Kosten:	8
mulu	$\#2,\ D_1$	Kosten:	44
move.L	$A_5,\ A_2$	Kosten:	4
adda.L	$\#-8,\ A_2$	Kosten:	12
adda.L	$D_1,\ A_2$	Kosten:	8
move	$(A_2),\ D_2$	Kosten:	8
addq	$\#2,\ D_2$	Kosten:	4
move.L	$A_5,\ A_3$	Kosten:	4
adda.L	$\#-4,\ A_3$	Kosten:	12
move	$D_2,\ (A_3)$	Kosten:	8

Gesamtkosten : 124

Diskussion:

- $\longrightarrow$ Die Folge ohne komplexe Adressierungsarten ist erheblich teurer :-(
- $\longrightarrow$ Sie benötigt auch mehr Hilfsregister :-(
- $\longrightarrow$ Die beiden Folgen sind nur äquivalent im Hinblick auf den Speicher — die Register haben anschließend verschiedene Inhalte ...
- $\longrightarrow$ Eine korrekte Folge von Instruktionen kann als eine Pflasterung des Syntaxbaums aufgefasst werden !!!

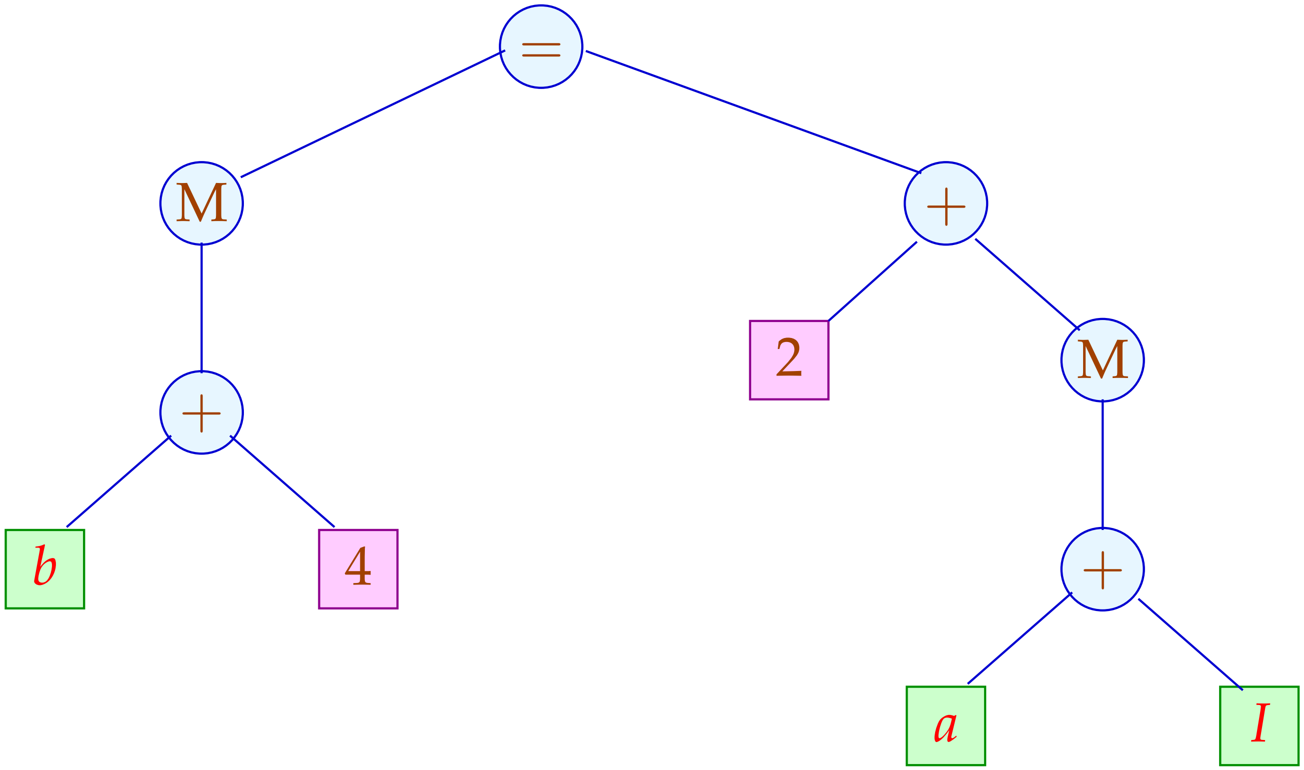
Genereller Ansatz:

- Wir betrachten Basis-Blöcke vor der Registerverteilung:

$$
\begin{aligned}
A &= a + I; \\
D_1 &= M[A]; \\
D_2 &= D_1 + 2; \\
B &= b + 4; \\
M[B] &= D_2
\end{aligned}
$$

- Wir fassen diese als Folge von Bäumen auf. Wurzeln:
  - Werte, die mehrmals verwendet werden;
  - Variablen, die am Ende des Blocks lebendig sind;
  - Stores.

... im Beispiel:



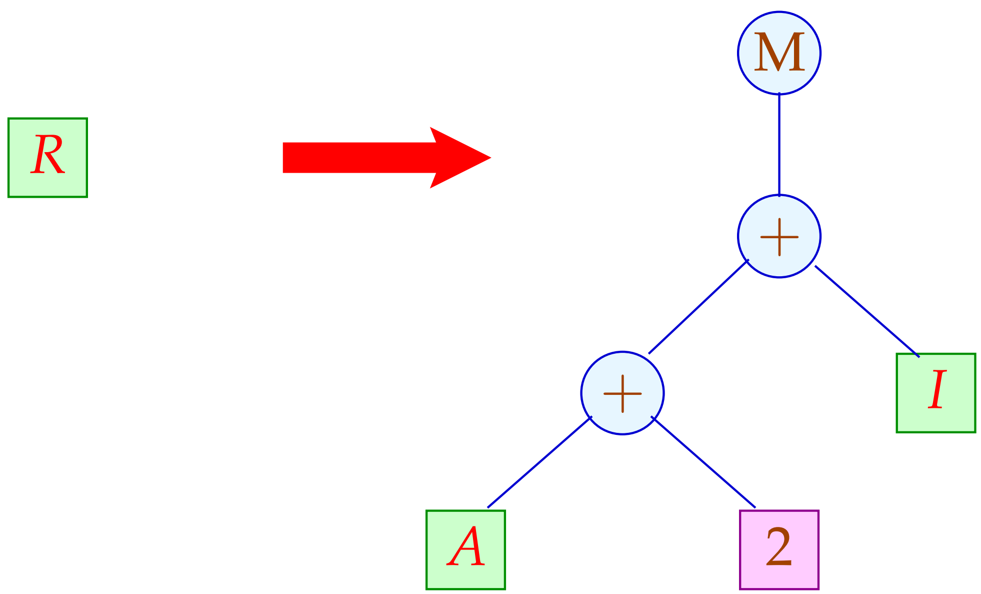
Die Hilfsvariablen A, B, D_1, D_2 sind vorerst verschwunden :-)

Idee:

Beschreibe den Effekt einer Instruktion als Ersetzungsregel auf Bäumen:

Die Instruktion: $R = M[A + 2 + D];$

entspricht zum Beispiel:

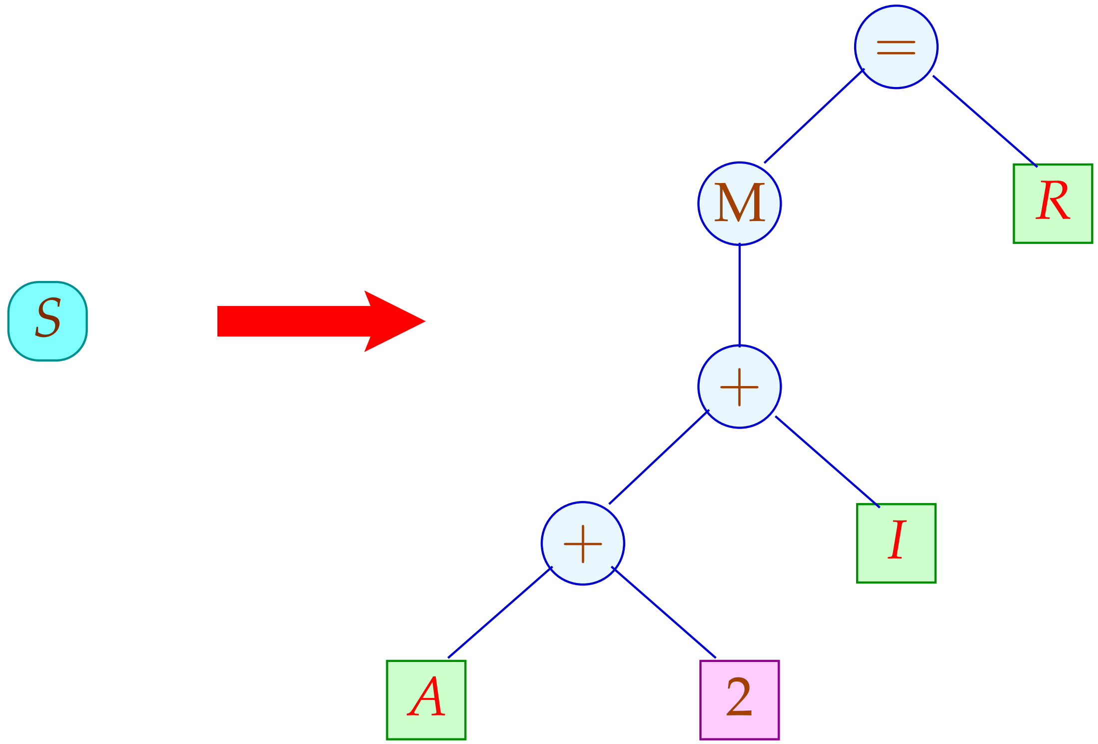


linke Seite	Ergebnisregister(klasse)
rechte Seite	berechneter Wert für Ergebnisregister
innere Knoten	• Load M • Arithmetik
Blätter	• Argumentregister(klassen) • Konstanten(klasse)

Die Grundidee erweitern wir (evt.) um eine Store-Operation.

Für die Instruktion: $M[A + 2 + D] = R;$

erlauben wir uns:



Die linke Seite S kommt nicht in rechten Seiten vor :-)

Spezifikation des Instruktionssatzes:

(1)	verfügbare Registerklassen	//	Nichtterminale
(2)	Operatoren und Konstantenklassen	//	Terminale
(3)	Instruktionen	//	Regeln

$\Longrightarrow$ reguläre Baumgrammatik

Triviales Beispiel:

Loads :	Comps :	Moves :
$D \rightarrow M[A]$	$D \rightarrow c$	$D \rightarrow A$
$D \rightarrow M[A + A]$	$D \rightarrow D + D$	$A \rightarrow D$

- Registerklassen D (Data) und A (Address).
- Arithmetik wird nur für Daten unterstützt ...
- Laden nur für Adressen :-)
- Zwischen Daten- und Adressregistern gibt es Moves.

Target: $M[A + c]$

Aufgabe:

Finde Folge von Regelanwendungen, die das Target aus einem Nichtterminal erzeugt ...

$\boxed{\boxed{D}}$

Target: $M[A + c]$

Aufgabe:

Finde Folge von Regelanwendungen, die das Target aus einem Nichtterminal erzeugt ...

Target: $M[A + c]$

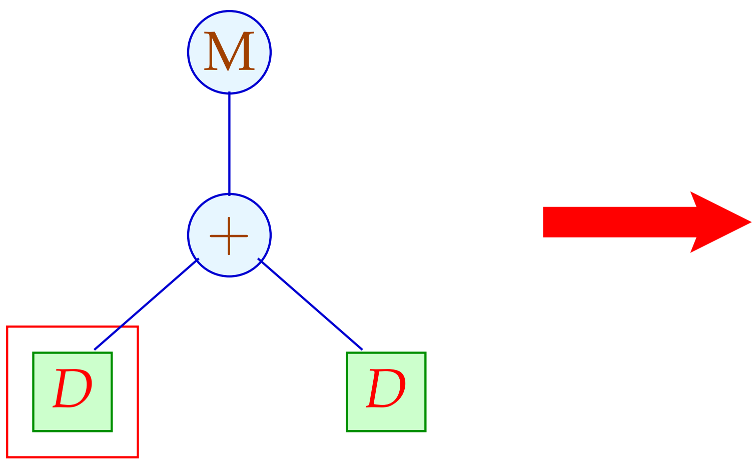
Aufgabe:

Finde Folge von Regelanwendungen, die das Target aus einem Nichtterminal erzeugt ...

Target: $M[A + c]$

Aufgabe:

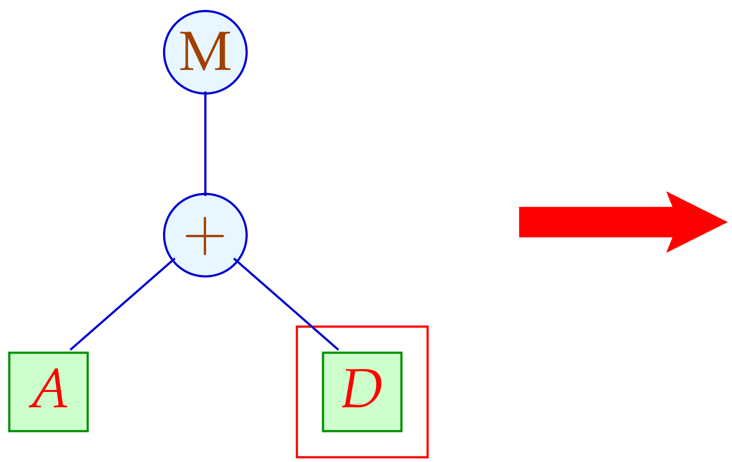
Finde Folge von Regelanwendungen, die das Target aus einem Nichtterminal erzeugt ...



Target: $M[A + c]$

Aufgabe:

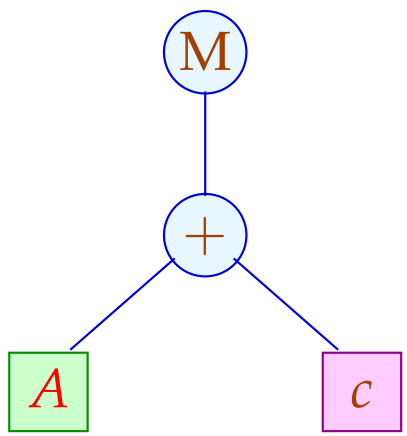
Finde Folge von Regelanwendungen, die das Target aus einem Nichtterminal erzeugt ...



Target: $M[A + c]$

Aufgabe:

Finde Folge von Regelanwendungen, die das Target aus einem Nichtterminal erzeugt ...



Die umgekehrte Folge der Regelanwendungen liefert eine geeignete Instruktionsfolge :-)

Verschiedene Ableitungen liefern verschiedene Folgen ...

Problem:

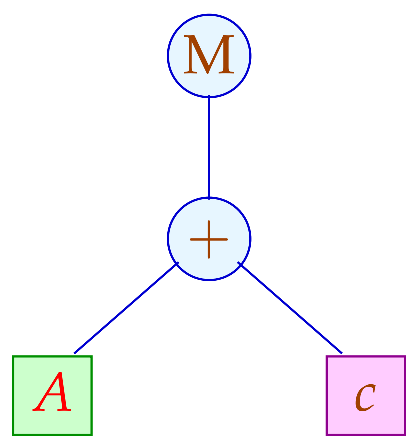
$\longrightarrow$ Wie durchsuchen wir systematisch die Menge aller Ableitungen ?

$\longrightarrow$ Wie finden wir die beste ??

Beobachtung:

- Nichtterminale stehen stets an den Blättern.
- Statt eine Ableitung für das Target topdown zu raten, sammeln wir sämtliche Möglichkeiten bottom-up auf

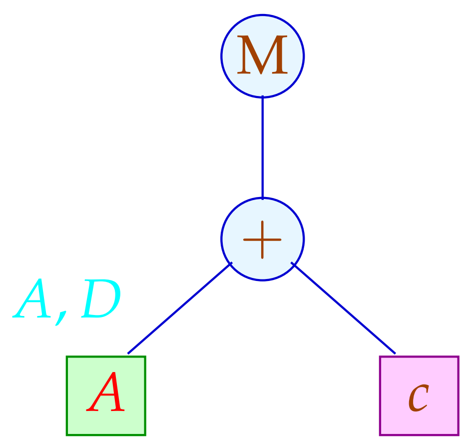
  $\Longrightarrow$ Tree parsing
- Dazu lesen wir die Regeln von rechts nach links ...



Beobachtung:

- Nichtterminale stehen stets an den Blättern.
- Statt eine Ableitung für das Target topdown zu raten, sammeln wir sämtliche Möglichkeiten bottom-up auf

  $\Longrightarrow$ Tree parsing
- Dazu lesen wir die Regeln von rechts nach links ...

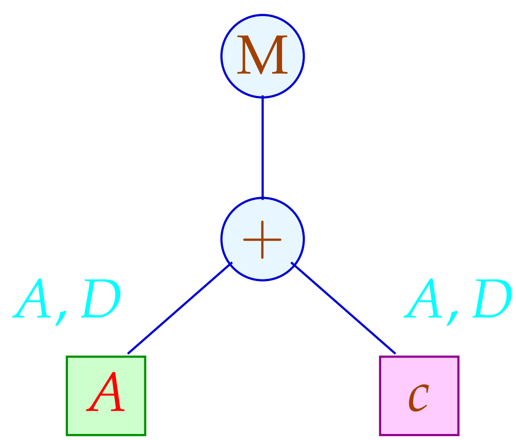


Beobachtung:

- Nichtterminale stehen stets an den Blättern.
- Statt eine Ableitung für das Target topdown zu raten, sammeln wir sämtliche Möglichkeiten bottom-up auf

$\Longrightarrow$ Tree parsing

- Dazu lesen wir die Regeln von rechts nach links ...

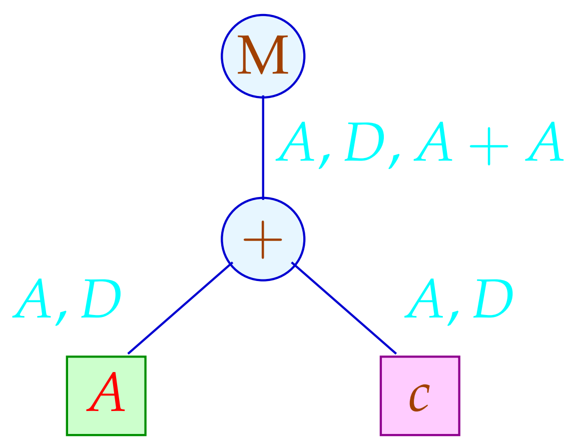


Beobachtung:

- Nichtterminale stehen stets an den Blättern.
- Statt eine Ableitung für das Target topdown zu raten, sammeln wir sämtliche Möglichkeiten bottom-up auf

$\Longrightarrow$ Tree parsing

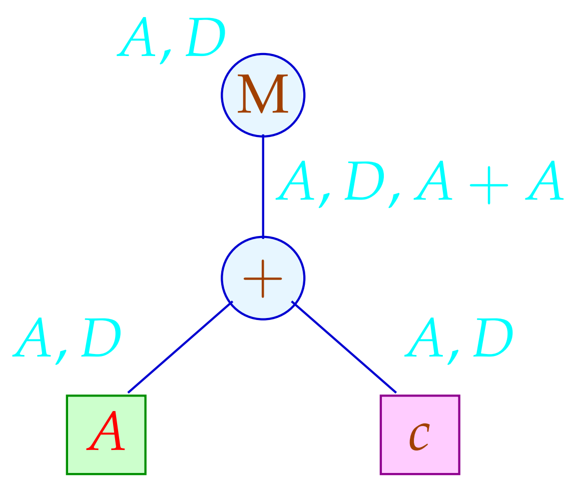
- Dazu lesen wir die Regeln von rechts nach links ...



Beobachtung:

- Nichtterminale stehen stets an den Blättern.
- Statt eine Ableitung für das Target topdown zu raten, sammeln wir sämtliche Möglichkeiten bottom-up auf

  $\Longrightarrow$ Tree parsing
- Dazu lesen wir die Regeln von rechts nach links ...



Für jeden Teilbaum t des Targets sammeln wir die Menge

$$Q(t) \subseteq \{S\} \cup \text{Reg} \cup \text{Term}$$

Reg die Menge der Registerklassen,
Term die Menge der Teilbäume rechter Seiten — auf mit:

$$Q(t) = \{s \mid s \Rightarrow^* t\}$$

Diese ergeben sich zu:

$$\begin{array}{lll}
Q(R) & = & \text{Move}\,\{R\} \\
Q(c) & = & \text{Move}\,\{c\} \\
Q(a(t_1, \ldots, t_k)) & = & \text{Move}\,\{s = a(s_1, \ldots, s_k) \in \text{Term} \mid s_i \in Q(t_i)\} \\
 & & \qquad // \quad \text{normalerweise} \quad k \leq 2 \quad :\text{-})
\end{array}$$

Die Hilfsfunktion $\quad \textsf{Move} \quad$ bildet den Abschluss unter Regelanwendungen:

$$\begin{array}{lll} \textsf{Move}\,(L) & \supseteq & L \\ \textsf{Move}\,(L) & \supseteq & \{R \in \textsf{Reg} \mid \exists\, s \in L : \quad R \rightarrow s\} \end{array}$$

Die kleinste Lösung dieses Constraint-Systems lässt sich aus der Grammatik in linearer Zeit berechnen :-)

// Im Beispiel haben wir in $Q(t)$ auf s verzichtet,
// falls s kein echter Teilterm einer rechten Seite ist :-)

Auswahlkriterien:

- Länge des Codes;
- Laufzeit der Ausführung;
- Parallelisierbarkeit;
- ...

Achtung:

Die Laufzeit von Instruktionen kann vom Kontext abhängen !!?

Vereinfachung:

Jede Instruktion r habe Kosten $c[r]$.

Die Kosten einer Instruktionsfolge sind additiv:

$$c[r_1 \ldots r_k] = c[r_1] + \ldots + c[r_k]$$

	c	Instruktion
0	3	$D \rightarrow M[A + A]$
1	2	$D \rightarrow M[A]$
2	1	$D \rightarrow D + D$
3	1	$D \rightarrow c$
4	1	$D \rightarrow A$
5	1	$A \rightarrow D$

Aufgabe:

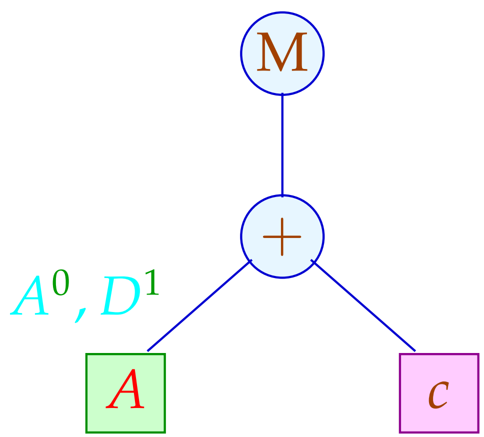
Wähle eine Instruktionsfolge mit minimalen Kosten !

Idee:

Sammle Ableitungen bottom-up auf unter

* Kostenkalkulation und
* Auswahl.

... im Beispiel:

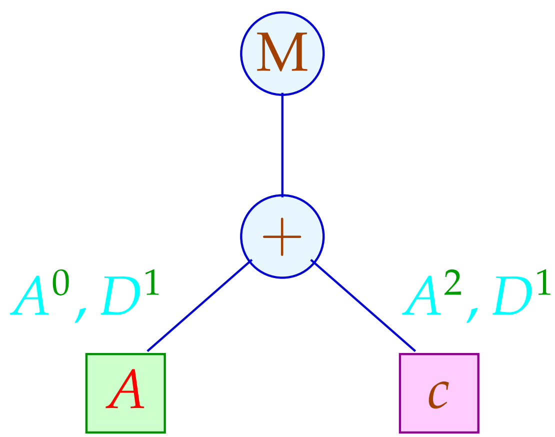


Idee:

Sammle Ableitungen bottom-up auf unter

* Kostenkalkulation und
* Auswahl.

... im Beispiel:

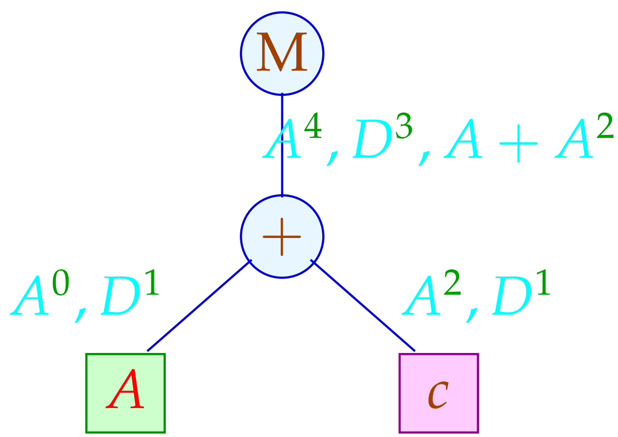


Idee:

Sammle Ableitungen bottom-up auf unter

- Kostenkalkulation und
- Auswahl.

... im Beispiel:

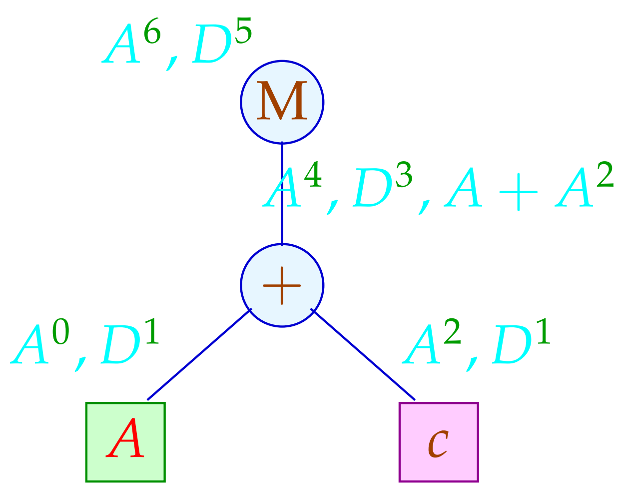


Idee:

Sammle Ableitungen bottom-up auf unter

* Kostenkalkulation und
* Auswahl.

... im Beispiel:

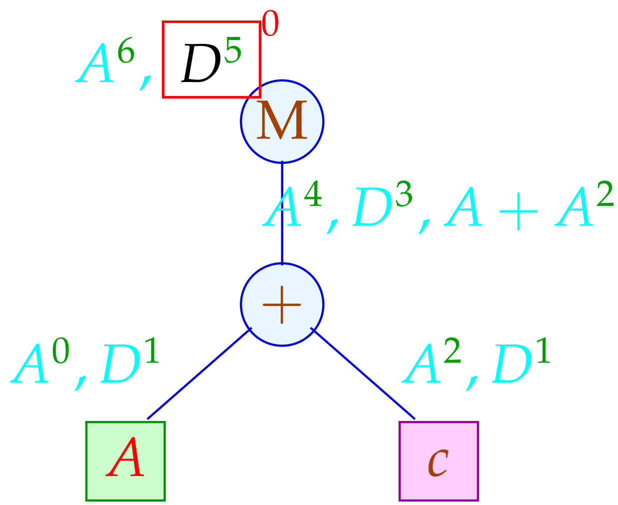


Idee:

Sammle Ableitungen bottom-up auf unter

- Kostenkalkulation und
- Auswahl.

... im Beispiel:

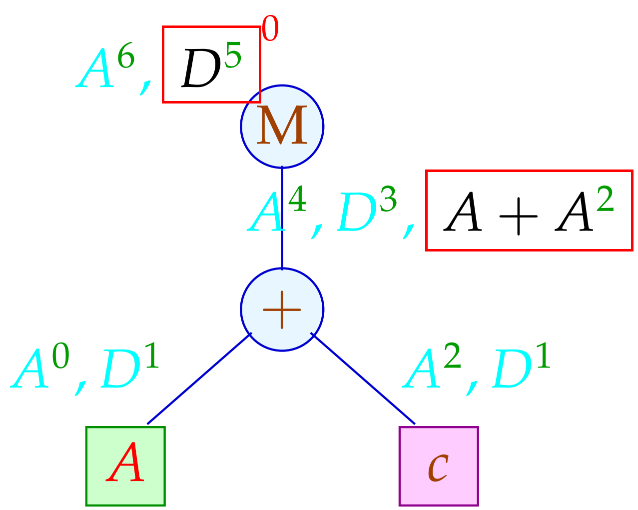


Idee:

Sammle Ableitungen bottom-up auf unter

* Kostenkalkulation und
* Auswahl.

... im Beispiel:

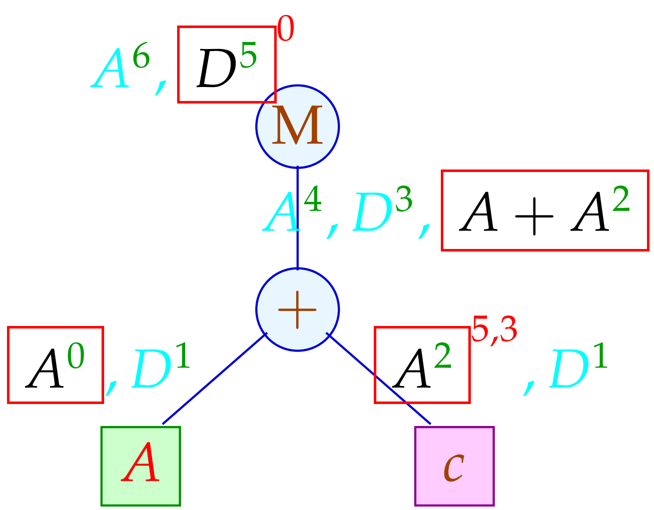


Idee:

Sammle Ableitungen bottom-up auf unter

- Kostenkalkulation und
- Auswahl.

... im Beispiel:



Kostenkalkulation:

$$
\begin{array}{lcll}
c_t[s] & = & c_{t_1}[s_1] + \ldots + c_{t_k}[s_k] & \text{falls} \quad s = a(s_1, \ldots, s_k), t = a(t_1, \ldots, t_k) \\
c_t[R] & = & \bigsqcap\{c[R,s] + c_t[s] \mid s \in Q(t)\} & \text{wobei}
\end{array}
$$

$$
\begin{array}{lcll}
c[R,s] & \leq & c[r] & \text{falls} \quad r \;:\; R \to s \\
c[R,s] & \leq & c[r] + c[R',s] & \text{falls} \quad r \;:\; R \to R'
\end{array}
$$

Das Constraint-System für $c[R,s]$ kann in Zeit $\mathcal{O}(n \cdot \log n)$ gelöst werden — falls n die Anzahl der Paare R, s ist :-)

Für jedes R, s liefert die Fixpunkt-Berechnung eine Folge:

$$\pi[R,s] \;:\; R \Rightarrow R_1 \Rightarrow \ldots \Rightarrow R_k \Rightarrow s$$

deren Kosten gerade $c[R,s]$ ist :-)

Mithilfe der $\pi[R, s]$ lässt sich eine billigste Ableitung topdown rekonstruieren :-)

Im Beispiel:

$$\begin{array}{lll} D_2 & = & c; \\ A_2 & = & D_2; \\ D_1 & = & M[A_1 + A_2]; \end{array}$$

mit Kosten 5 . Die Alternative:

$$\begin{array}{lll} D_2 & = & c; \\ D_3 & = & A_1; \\ D_4 & = & D_3 + D_2; \\ A_2 & = & D_4; \\ D_1 & = & M[A_2]; \end{array}$$

hätte Kosten 7 :-)

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:

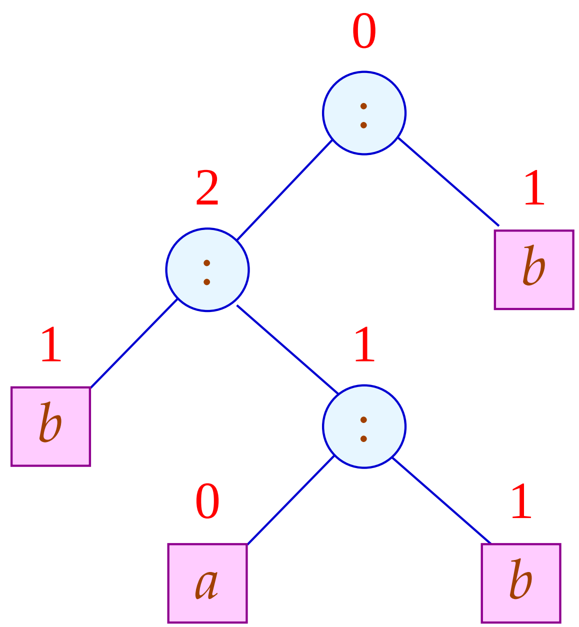
$$\begin{array}{llll} \delta_c : & Q & \text{falls} & c \quad \text{Konstante} \\ \delta_a : & Q^k \to Q & \text{falls} & a \quad k\text{-stellig} \end{array}$$

Beispiel:

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

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

// 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:

Q $==$ Teilmengen von $\textsf{Reg} \cup \textsf{Term} \cup \{S\}$

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

F $==$ gewünschter Effekt

δ_R $==$ $\textsf{Move}\,\{R\}$

δ_c $==$ $\textsf{Move}\,\{c\}$

$\delta_a(Q_1, \ldots, Q_k)$ $==$ $\textsf{Move}\,\{s = a(s_1, \ldots, s_k) \in \textsf{Term} \mid s_i \in Q_i\}$

... im Beispiel:

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

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

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

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 :-)

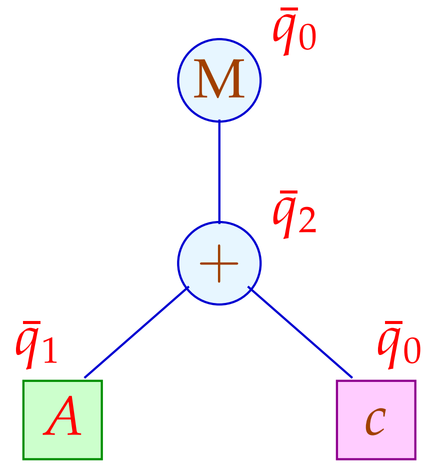
... im Beispiel:

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

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

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

... 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

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

Im Beispiel:

$$\begin{aligned} Q_1 &= \{\bar{q}_0, \bar{q}_2, \bar{q}_3, \bar{q}_4\} \\ Q_2 &= \{\bar{q}_1\} \end{aligned}$$

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 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 :-)