

## 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:

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(A1, D1.W), D5`  
benötigt:  $4 + 10 + 0 = 14$  Zyklen

Alternativ könnten wir erzeugen:

<code>adda</code>	<code>#8, A<sub>1</sub></code>	Kosten: $8 + 8 + 0 = 16$
<code>adda</code>	<code>D<sub>1</sub>.W, A<sub>1</sub></code>	Kosten: $8 + 0 + 0 = 8$
<code>move.B</code>	<code>(A<sub>1</sub>), D<sub>5</sub></code>	Kosten: $4 + 4 + 0 = 8$

mit Gesamtkosten **32** oder:

<code>adda</code>	<code>D<sub>1</sub>.W, A<sub>1</sub></code>	Kosten: $8 + 0 + 0 = 8$
<code>move.B</code>	<code>8(A<sub>1</sub>), D<sub>5</sub></code>	Kosten: $4 + 8 + 0 = 12$

mit Gesamtkosten **20 :-)**

## Achtung:

- Die verschiedenen 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 :-)

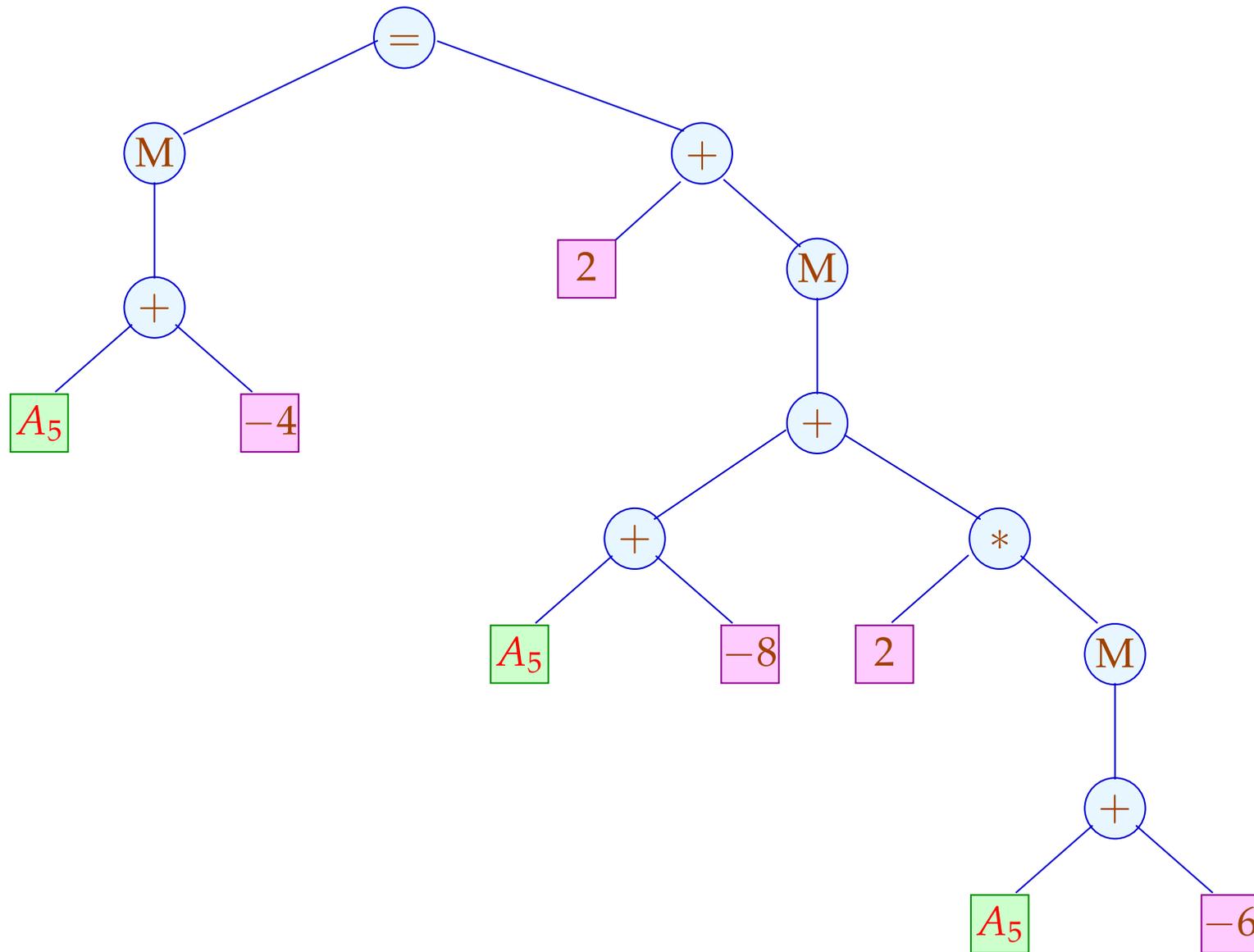
## Etwas größeres Beispiel:

```
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:

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

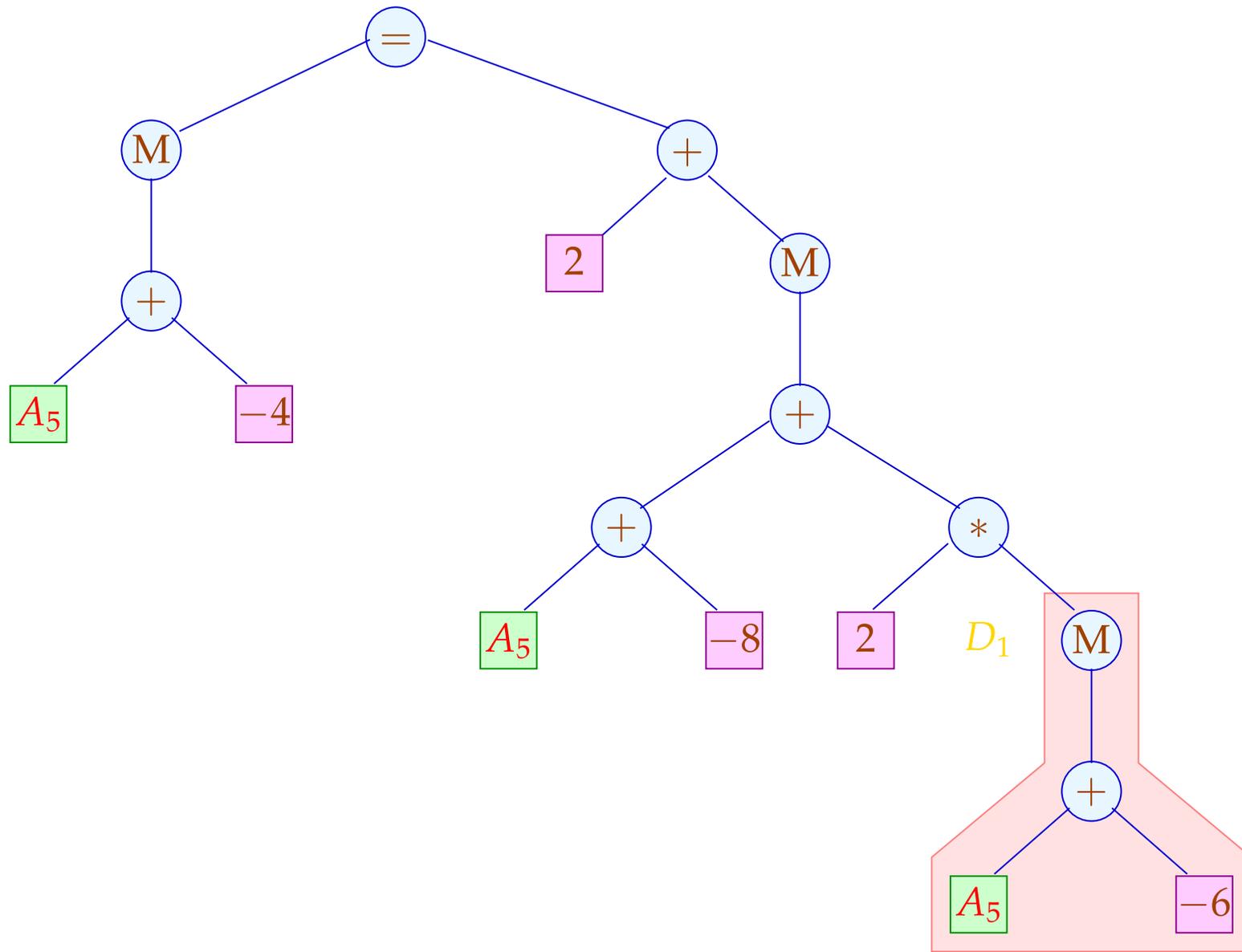
Das entspricht dem Syntaxbaum:

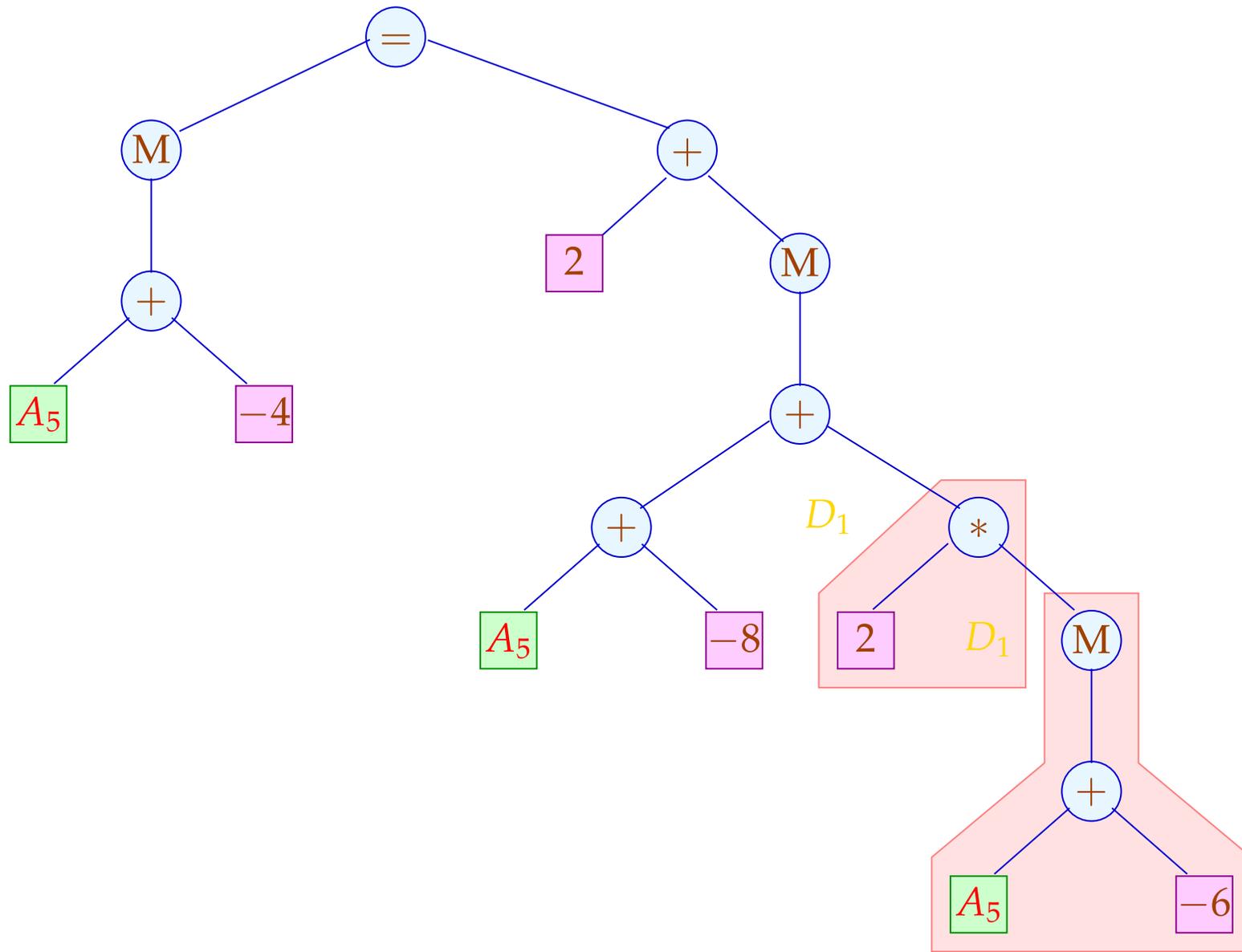


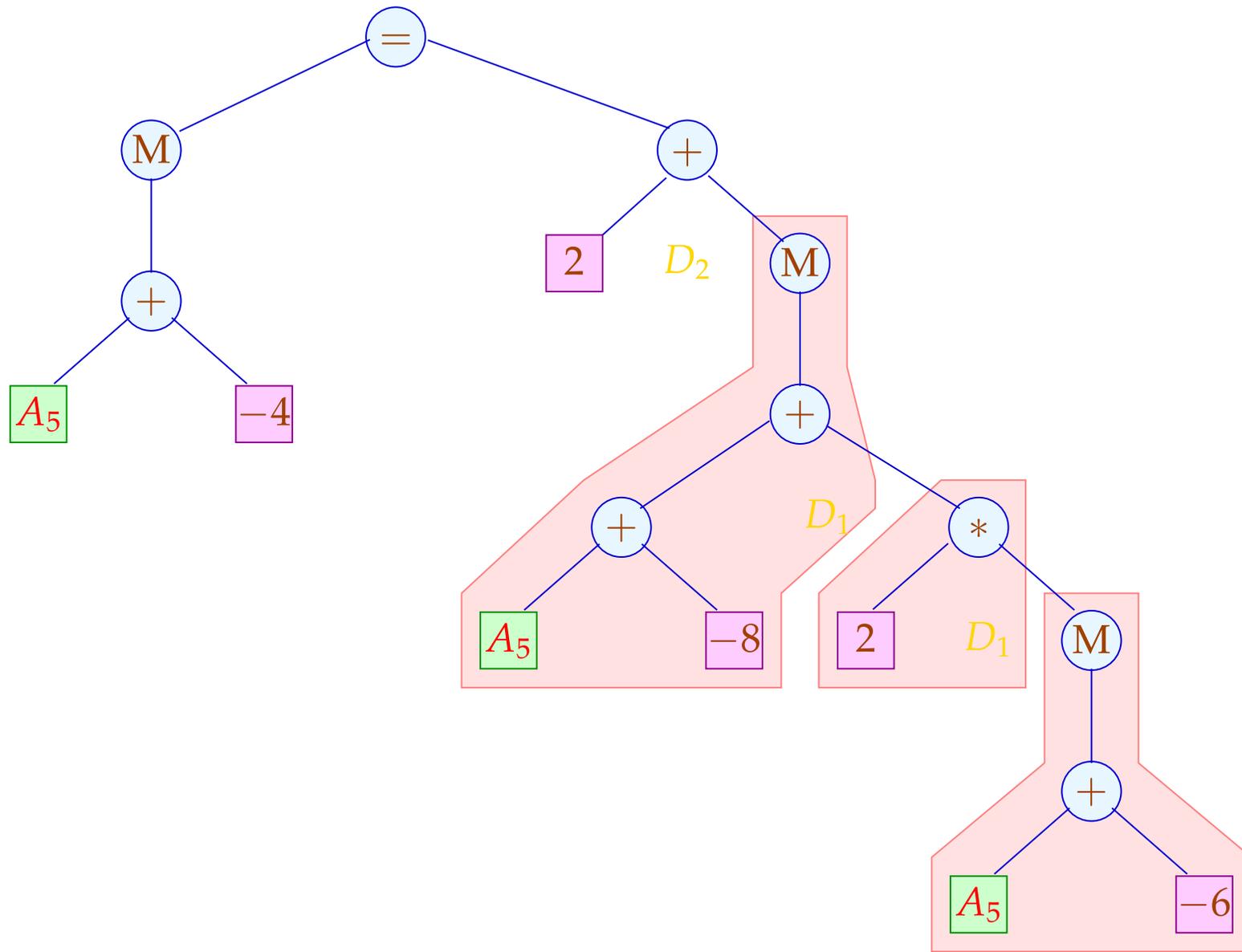
Eine mögliche Code-Sequenz:

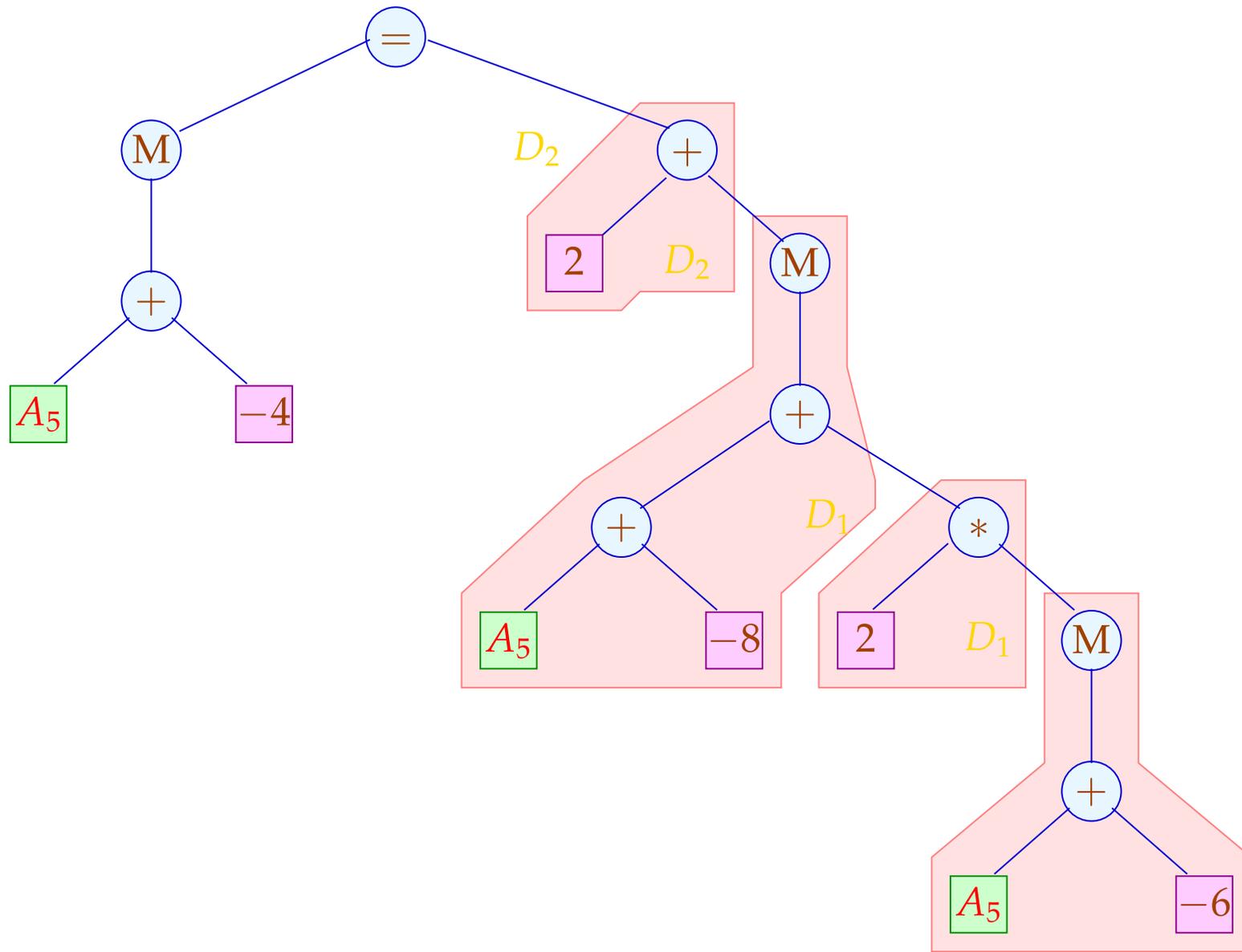
move	-6( $A_5$ ), $D_1$	Kosten:	12
add	$D_1$ , $D_1$	Kosten:	4
move	-8( $A_5$ , $D_1$ ), $D_2$	Kosten:	14
addq	#2, $D_2$	Kosten:	4
move	$D_2$ , -4( $A_5$ )	Kosten:	12

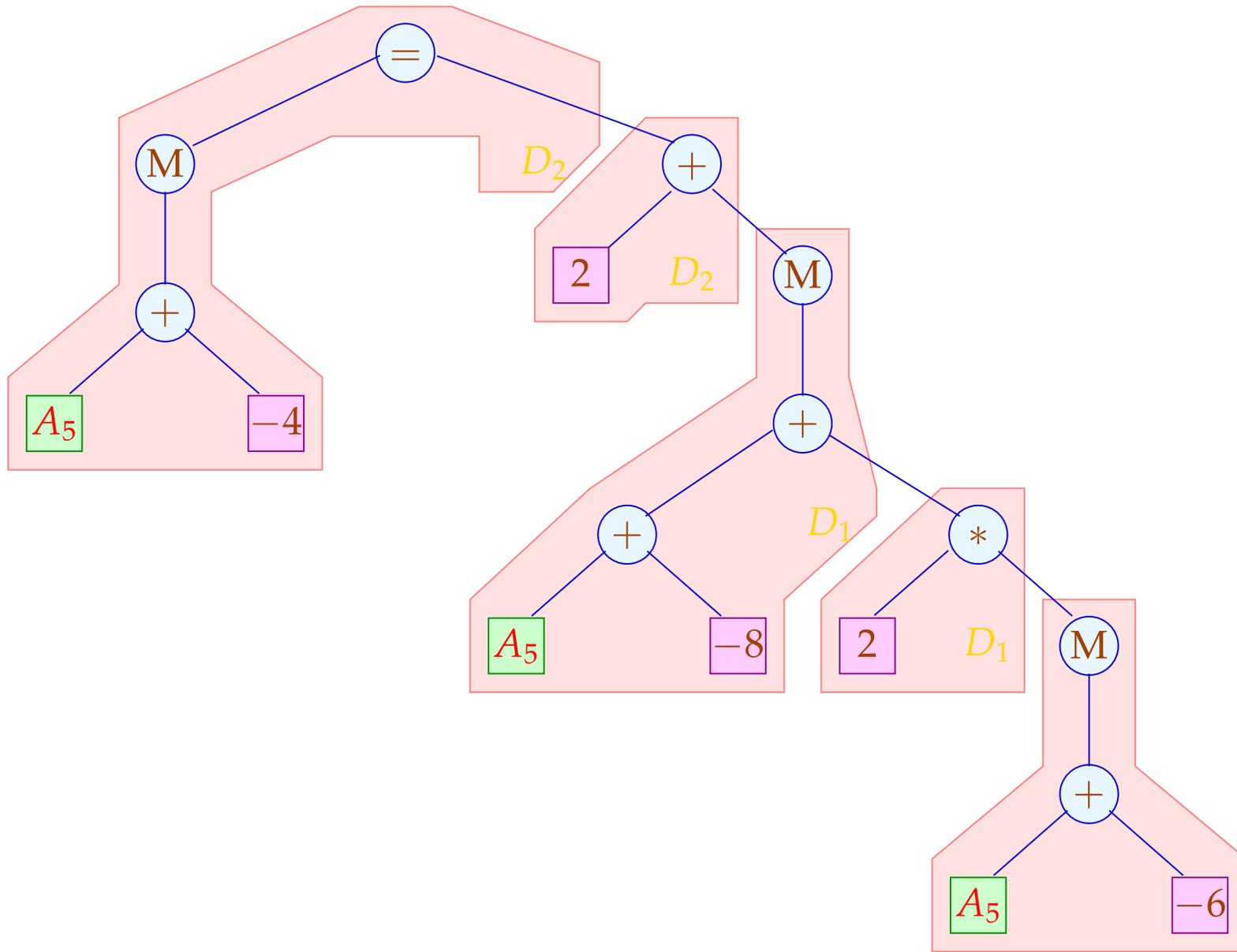
*Gesamtkosten :* 46











## 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
<i>Gesamtkosten :</i>			<b>124</b>

## Diskussion:

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

## Genereller Ansatz:

- Wir betrachten Basis-Blöcke **vor der Registerverteilung**:

$$A = a + I;$$

$$D_1 = M[A];$$

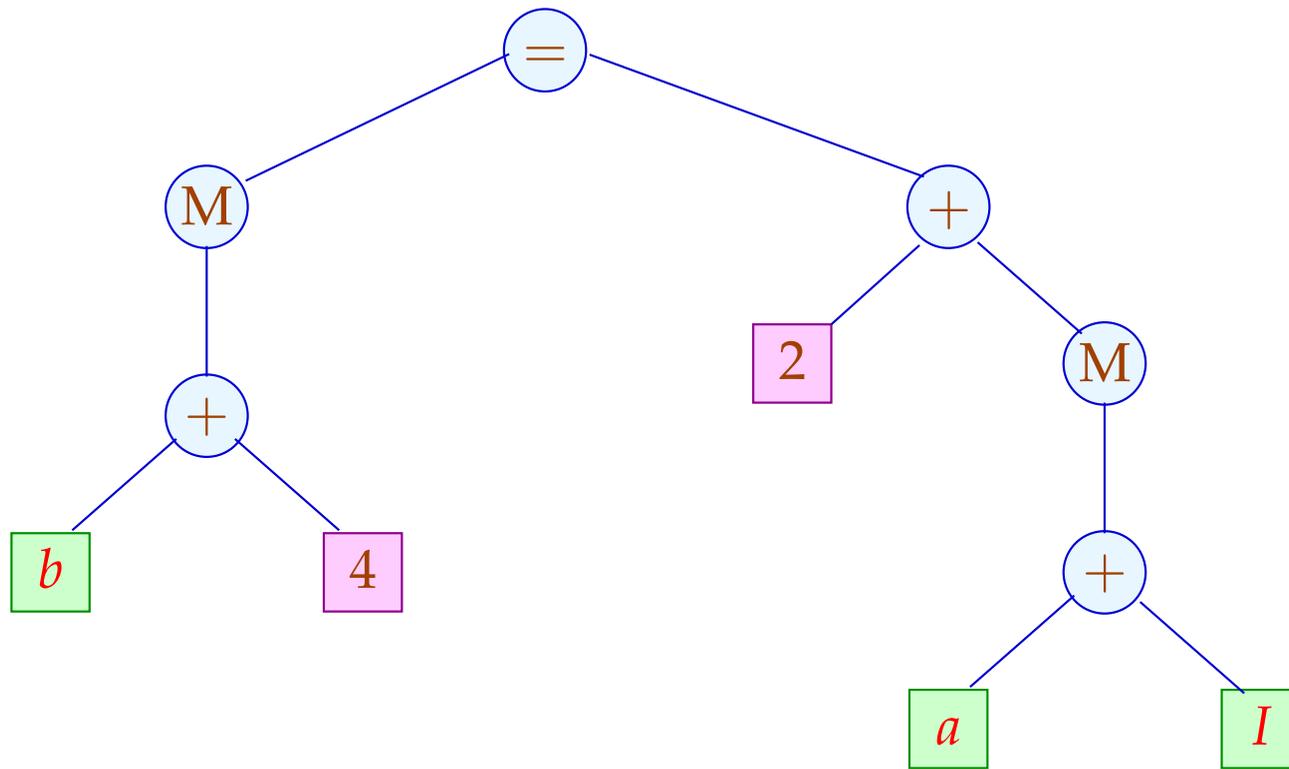
$$D_2 = D_1 + 2;$$

$$B = b + 4;$$

$$M[B] = D_2$$

- 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:



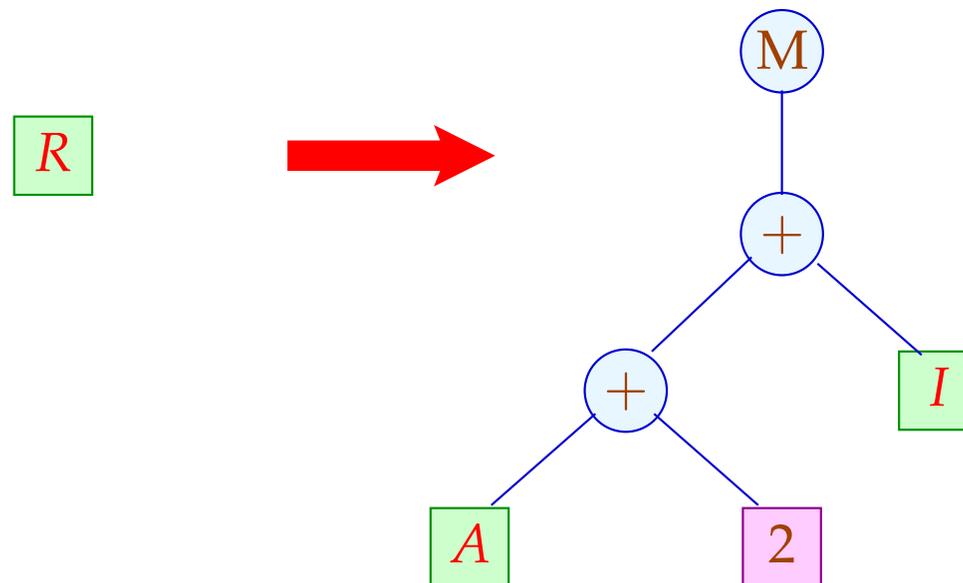
Die Hilfsvariablen  $A, B, D_1, D_2$  sind vorerst verschwunden :-)

Idee:

Beschreibe den Effekt einer Instruktion als **Ersetzungsregel** auf Bäumen:

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

entspricht zum Beispiel:

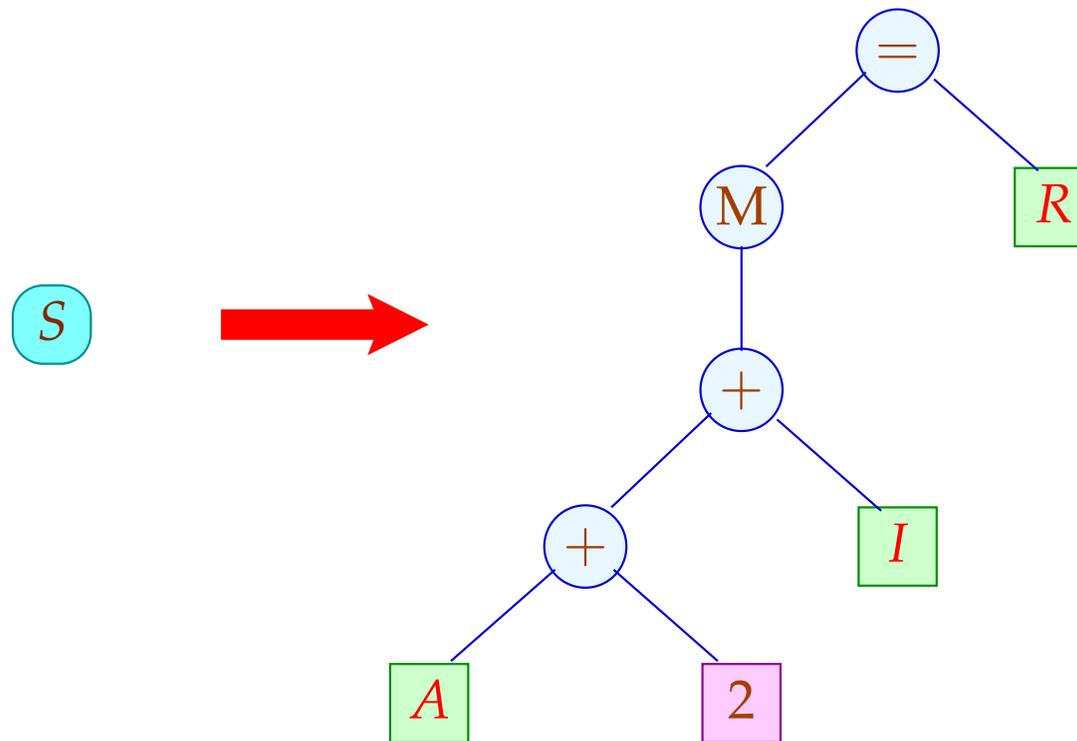


linke Seite	Ergebnisregister(klasse)
rechte Seite	berechneter Wert für Ergebnisregister
innere Knoten	<ul style="list-style-type: none"> <li>● Load <math>M</math></li> <li>● Arithmetik</li> </ul>
Blätter	<ul style="list-style-type: none"> <li>● Argumentregister(klassen)</li> <li>● Konstanten(klasse)</li> </ul>

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         |

⇒ reguläre Baumgrammatik

## Triviales Beispiel:

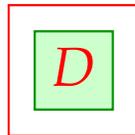
Loads :	Comps :	Moves :
$D \rightarrow M[A]$	$D \rightarrow c$	$D \rightarrow A$
$D \rightarrow M[A + A]$	$D \rightarrow D + D$	$A \rightarrow D$

- 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 ...



Target:  $M[A + c]$

Aufgabe:

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



Target:  $M[A + c]$

Aufgabe:

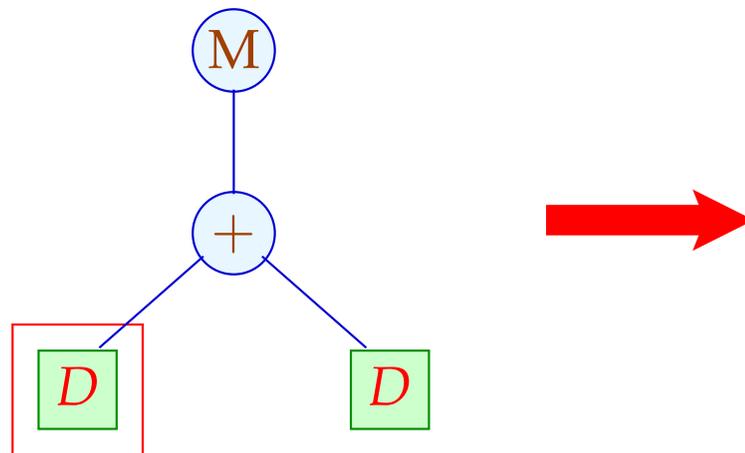
Finde Folge von Regelnanwendungen, die das Target aus einem Nichtterminal erzeugt ...



Target:  $M[A + c]$

Aufgabe:

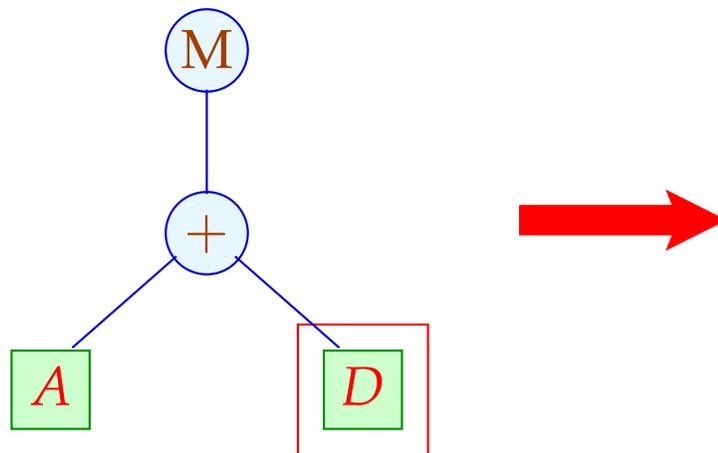
Finde Folge von Regelnanwendungen, die das Target aus einem Nichtterminal erzeugt ...



Target:  $M[A + c]$

Aufgabe:

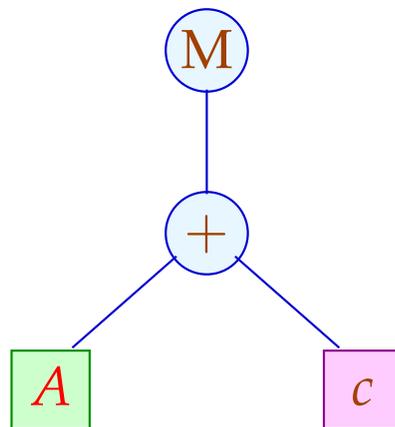
Finde Folge von Regelnanwendungen, die das Target aus einem Nichtterminal erzeugt ...



Target:  $M[A + c]$

Aufgabe:

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



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

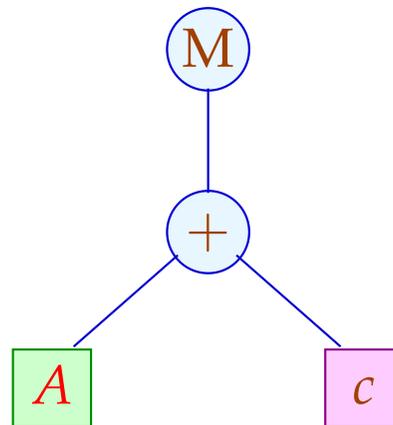
Verschiedene Ableitungen liefern verschiedene Folgen ...

## Problem:

- Wie durchsuchen wir systematisch die Menge aller Ableitungen ?
- Wie finden wir die **beste** ??

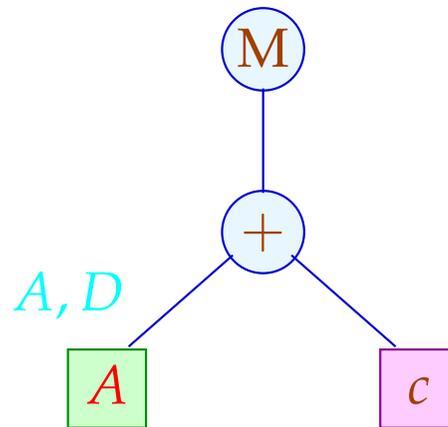
## Beobachtung:

- Nichtterminale stehen stets an den **Blättern**.
- Statt eine Ableitung für das Target topdown zu raten, sammeln wir sämtliche Möglichkeiten bottom-up auf  
 $\implies$  **Tree parsing**
- Dazu lesen wir die Regeln **von rechts nach links ...**



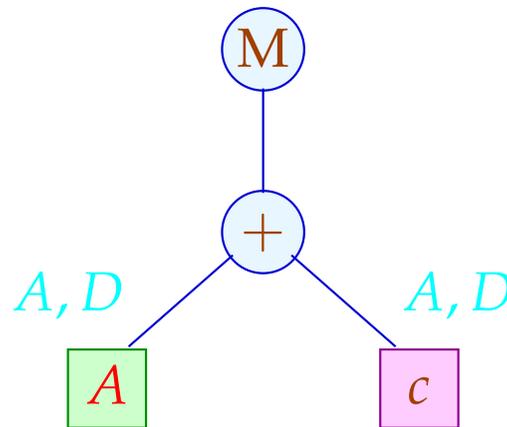
## Beobachtung:

- Nichtterminale stehen stets an den **Blättern**.
- Statt eine Ableitung für das Target topdown zu raten, sammeln wir sämtliche Möglichkeiten bottom-up auf  
 $\implies$  **Tree parsing**
- Dazu lesen wir die Regeln **von rechts nach links ...**



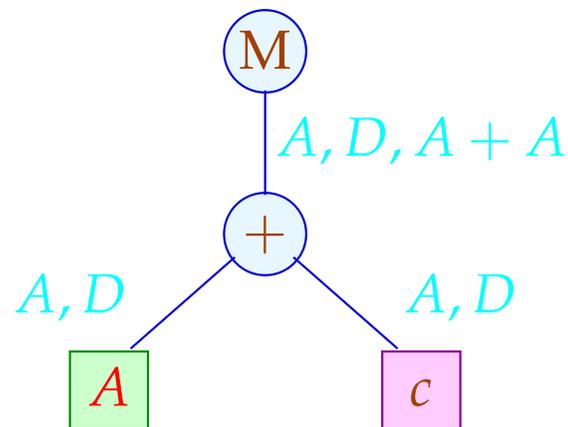
## Beobachtung:

- Nichtterminale stehen stets an den **Blättern**.
- Statt eine Ableitung für das Target topdown zu raten, sammeln wir sämtliche Möglichkeiten bottom-up auf  
 $\implies$  **Tree parsing**
- Dazu lesen wir die Regeln **von rechts nach links ...**



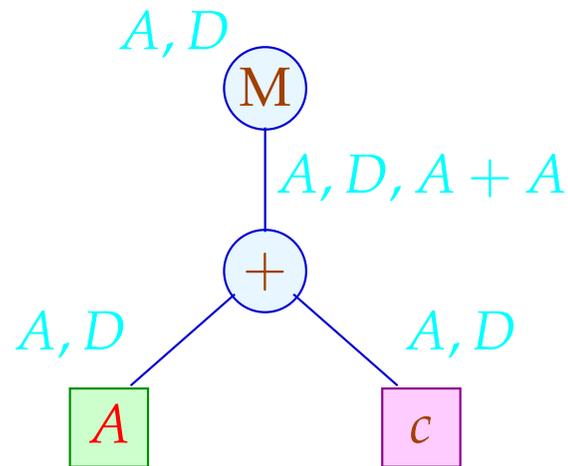
## Beobachtung:

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

$$Q(R) = \text{Move} \{R\}$$

$$Q(c) = \text{Move} \{c\}$$

$$Q(a(t_1, \dots, t_k)) = \text{Move} \{s = a(s_1, \dots, s_k) \in \text{Term} \mid s_i \in Q(t_i)\}$$

// normalerweise  $k \leq 2$  :-)

Die Hilfsfunktion **Move** bildet den Abschluss unter  
Regelanwendungen:

$$\text{Move}(L) \supseteq L$$

$$\text{Move}(L) \supseteq \{R \in \text{Reg} \mid \exists s \in L : R \rightarrow s\}$$

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**:

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

	$c$	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:

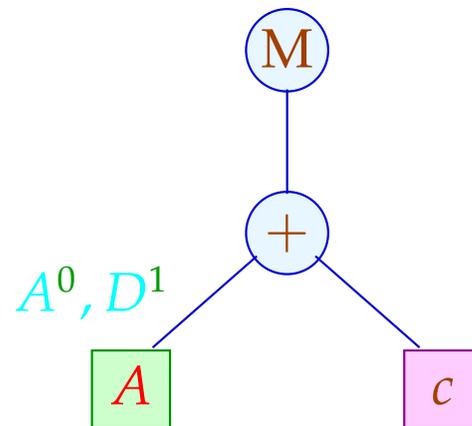
Wähle eine Instruktionsfolge mit minimalen Kosten !

Idee:

Sammele Ableitungen bottom-up auf unter

- \* Kostenkalkulation und
- \* Auswahl.

... im Beispiel:

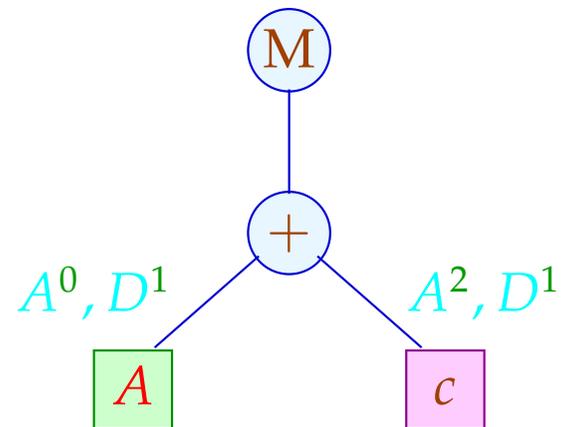


Idee:

Sammele Ableitungen bottom-up auf unter

- \* Kostenkalkulation und
- \* Auswahl.

... im Beispiel:

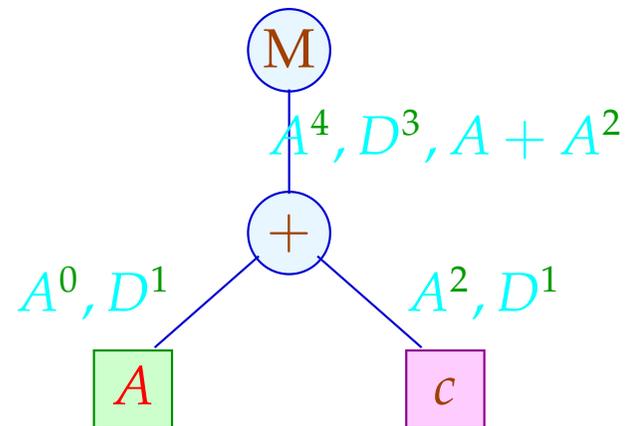


Idee:

Sammele Ableitungen bottom-up auf unter

- \* Kostenkalkulation und
- \* Auswahl.

... im Beispiel:

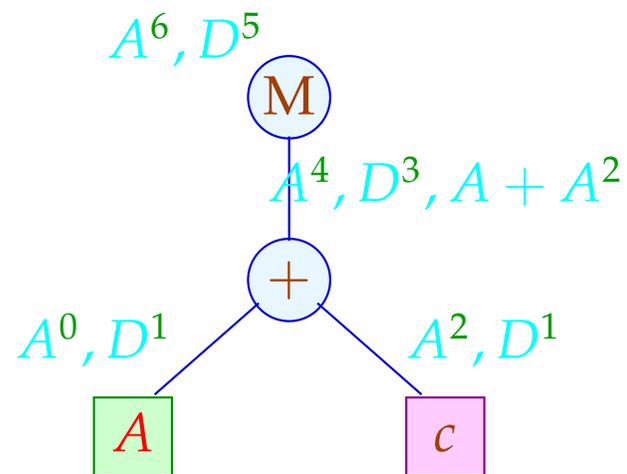


Idee:

Samme Ableitungen bottom-up auf unter

- \* Kostenkalkulation und
- \* Auswahl.

... im Beispiel:

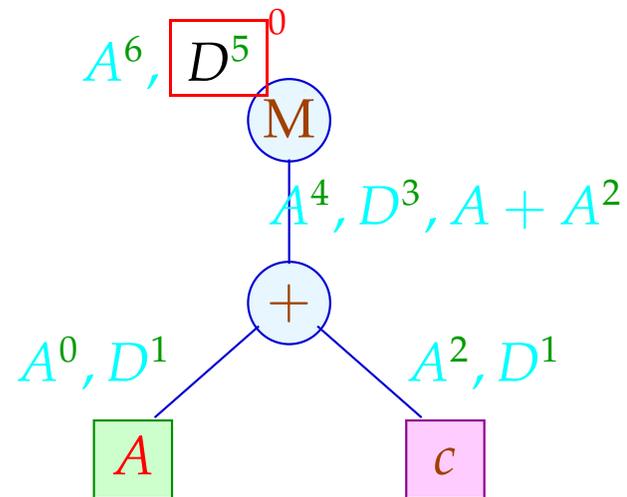


Idee:

Samme Ableitungen bottom-up auf unter

- \* Kostenkalkulation und
- \* Auswahl.

... im Beispiel:

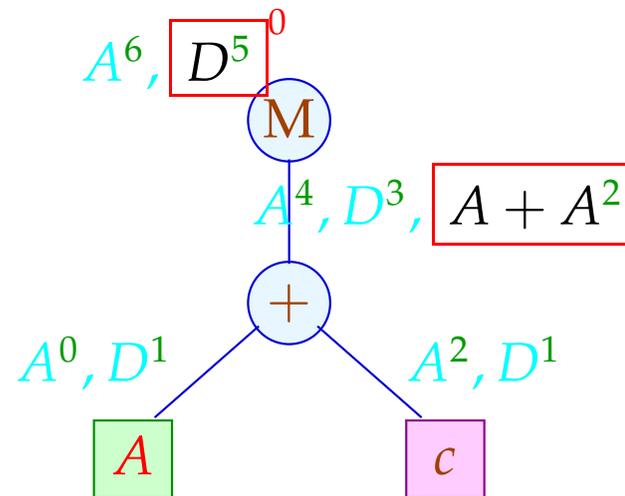


Idee:

Samme Ableitungen bottom-up auf unter

- \* Kostenkalkulation und
- \* Auswahl.

... im Beispiel:

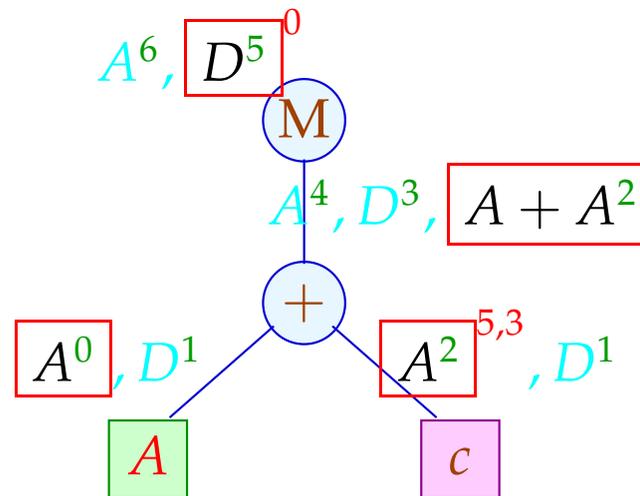


Idee:

Samme Ableitungen bottom-up auf unter

- \* Kostenkalkulation und
- \* Auswahl.

... im Beispiel:



## Kostenkalkulation:

$$c_t[s] = c_{t_1}[s_1] + \dots + c_{t_k}[s_k] \quad \text{falls } s = a(s_1, \dots, s_k), t = a(t_1, \dots, t_k)$$

$$c_t[R] = \bigcap \{c[R, s] + c_t[s] \mid s \in Q(t)\} \quad \text{wobei}$$

$$c[R, s] \leq c[r] \quad \text{falls } r : R \rightarrow s$$

$$c[R, s] \leq c[r] + c[R', s] \quad \text{falls } r : R \rightarrow R'$$

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:

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

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

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

Im Beispiel:

$$D_2 = c;$$

$$A_2 = D_2;$$

$$D_1 = M[A_1 + A_2];$$

mit Kosten 5. Die Alternative:

$$D_2 = c;$$

$$D_3 = A_1;$$

$$D_4 = D_3 + D_2;$$

$$A_2 = D_4;$$

$$D_1 = M[A_2];$$

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:

$Q$	$\equiv$	endliche Menge von Zuständen
$\Sigma$	$\equiv$	Operatoren und Konstanten
$\delta_a$	$\equiv$	Übergangsfunktion für $a \in \Sigma$
$F \subseteq Q$	$\equiv$	akzeptierende Zustände

Dabei ist:

$\delta_c : Q$  falls  $c$  Konstante

$\delta_a : Q^k \rightarrow Q$  falls  $a$   $k$ -stellig

Beispiel:

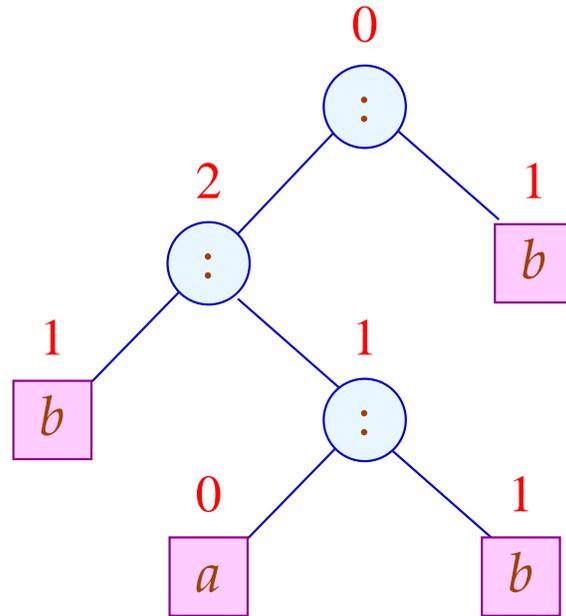
$$Q = \{0, 1, 2\} \quad F = \{0\}$$

$$\Sigma = \{a, b, :\}$$

$$\delta_a = 0 \quad \delta_b = 1$$

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

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



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

$$Q(c) = \delta_c$$

$$Q(a(t_1, \dots, t_k)) = \delta_a(Q(t_1), \dots, Q(t_k))$$

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

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

... in unserer Anwendung:

$Q$   $\equiv$  Teilmengen von  $\text{Reg} \cup \text{Term} \cup \{S\}$

// I.a. werden nicht **sämtliche** Teilmengen benötigt :-)

$F$   $\equiv$  gewünschter Effekt

$\delta_R$   $\equiv$  Move  $\{R\}$

$\delta_c$   $\equiv$  Move  $\{c\}$

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

... im Beispiel:

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

$$\begin{aligned}\delta_+(q_0, q_0) &= \{A, D, A + A\} = q_1 \\ &= \delta_+(q_0, -) \\ &= \delta_+(-, q_0)\end{aligned}$$

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

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  $\implies$   
Die Kosten zwischen Registern differieren nur um eine Konstante :-)
- Komplexe rechte Seiten lassen sich i.a. mittels **elementarerer** Instruktionen simulieren  $\implies$   
Die Kosten zwischen Teilausdrücken und Registern differieren nur um eine Konstante :-))
- Die Kostenberechnung ist additiv  $\implies$   
Wir können statt mit absoluten Kosten-Angaben auch mit Kosten-Differenzen rechnen !!!  
Von diesen gibt es nur **endlich viele** :-)

... im Beispiel:

$$\begin{aligned}\delta_c &= \{A \mapsto 1, D \mapsto 0\} = \bar{q}_0 \\ &= \delta_D\end{aligned}$$

$$\delta_A = \{A \mapsto 0, D \mapsto 1\} = \bar{q}_1$$

$$\delta_+(\bar{q}_1, \bar{q}_0) = \{A \mapsto 2, D \mapsto 1, A + A \mapsto 0\} = \bar{q}_2$$

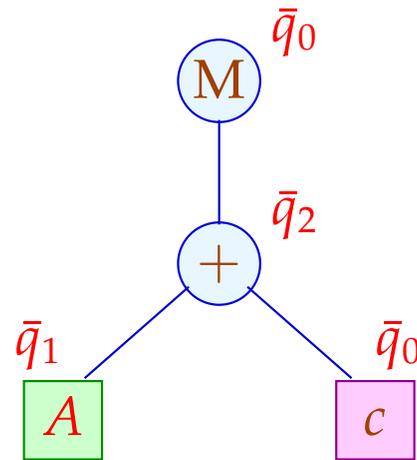
$$\delta_+(\bar{q}_0, \bar{q}_0) = \{A \mapsto 1, D \mapsto 0, A + A \mapsto 1\} = \bar{q}_3$$

$$\delta_+(\bar{q}_1, \bar{q}_1) = \{A \mapsto 4, D \mapsto 3, A + A \mapsto 0\} = \bar{q}_4$$

...

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

... das liefert die folgende Berechnung:



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

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

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

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

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 :-)

⇒ 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

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

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 **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:

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

```
 $\delta_+(\bar{q}_i, \bar{q}_j) =$  let  $c = \text{class}[\bar{q}_i];$   
                   $d = \text{disp}[c];$   
in if ( $valid[d + j] \equiv c$ )  
    then  $A[d + j]$   
    else default  
end
```



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 :-)