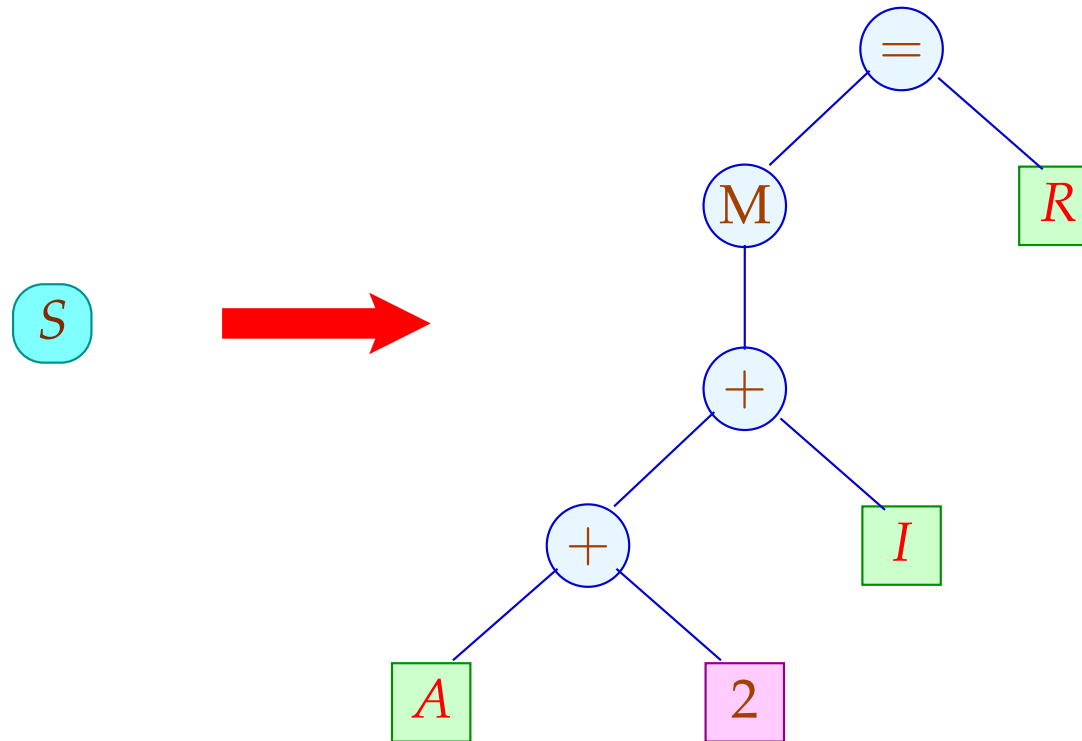


linke Seite	Ergebnisregister(klasse)
rechte Seite	berechneter Wert für Ergebnisregister
innere Knoten	<ul style="list-style-type: none"> <li>● Load <math>M</math></li> <li>● Arithmetik</li> </ul>
Blätter	<ul style="list-style-type: none"> <li>● Argumentregister(klassen)</li> <li>● Konstanten(klasse)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

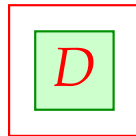
Loads :	Comps :	Moves :
$D \rightarrow M[A]$	$D \rightarrow c$	$D \rightarrow A$
$D \rightarrow M[A + A]$	$D \rightarrow D + D$	$A \rightarrow D$

- 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:

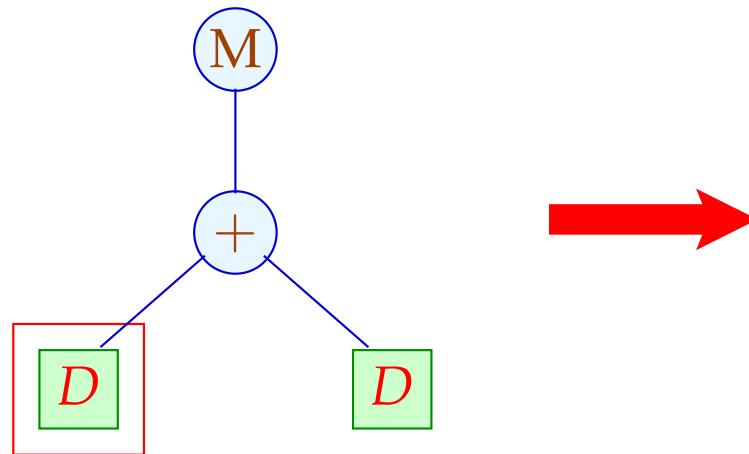
Finde Folge von Regeln, die das Target aus einem Nichtterminal erzeugt ...



Target:  $M[A + c]$

Aufgabe:

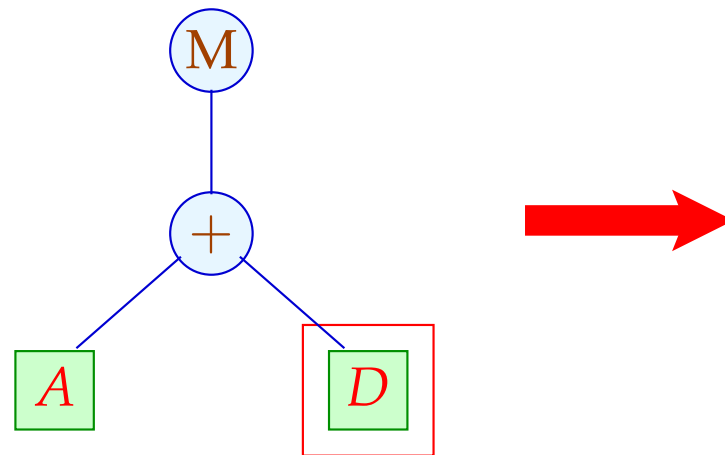
Finde Folge von Regelnanwendungen, die das Target aus einem Nichtterminal erzeugt ...



Target:  $M[A + c]$

Aufgabe:

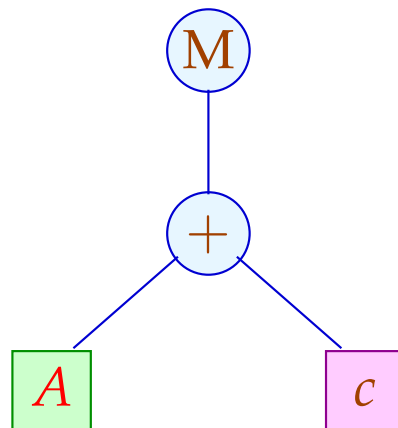
Finde Folge von Regelnanwendungen, die das Target aus einem Nichtterminal erzeugt ...



Target:  $M[A + c]$

Aufgabe:

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



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

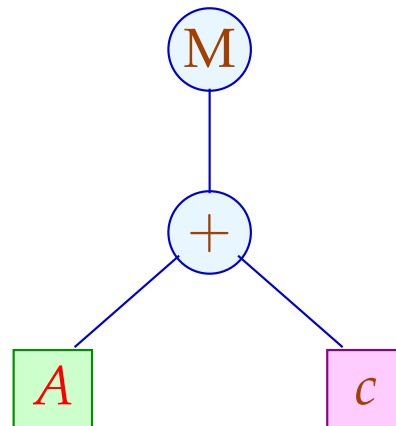
Verschiedene Ableitungen liefern verschiedene Folgen ...

## Problem:

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

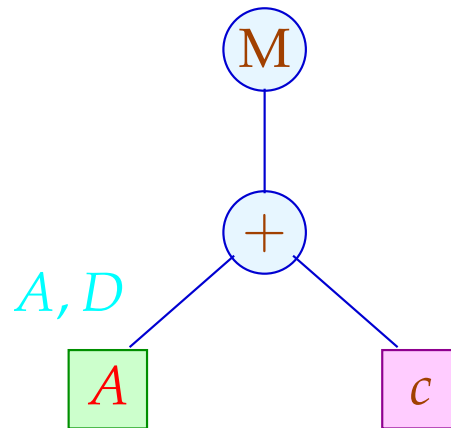
## Beobachtung:

- 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 ...**



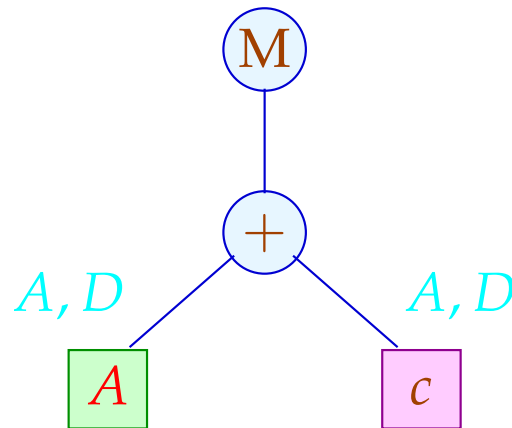
## Beobachtung:

- 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 ...**



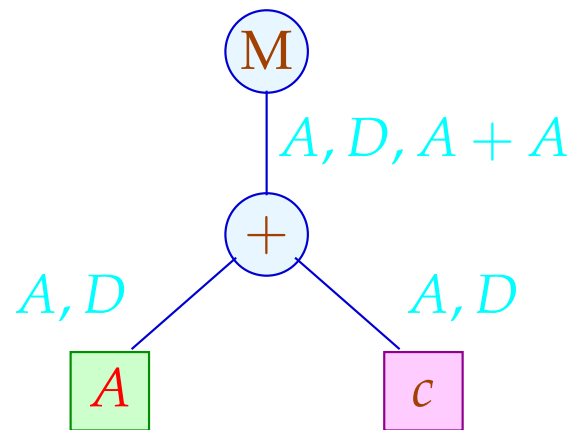
## Beobachtung:

- 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 ...**



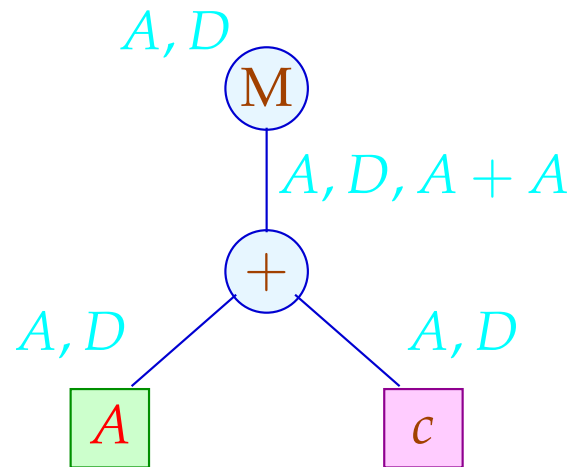
## Beobachtung:

- 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:

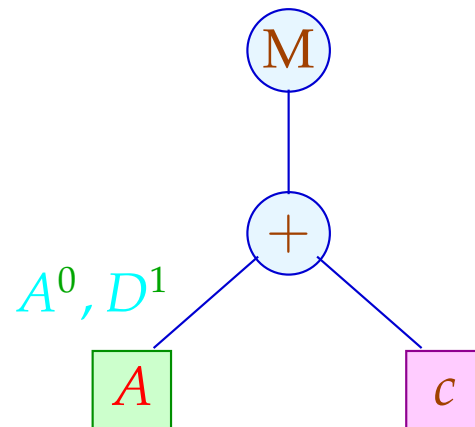
Wähle eine Instruktionsfolge mit minimalen Kosten !

Idee:

Samme Ableitungen bottom-up auf unter

- \* Kostenkalkulation und
- \* Auswahl.

... im Beispiel:

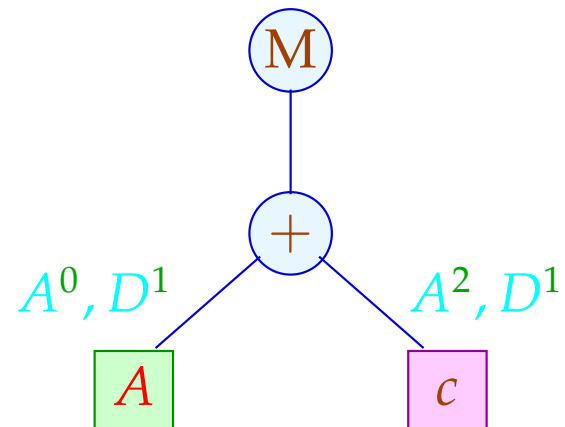


Idee:

Sammele Ableitungen bottom-up auf unter

- \* Kostenkalkulation und
- \* Auswahl.

... im Beispiel:

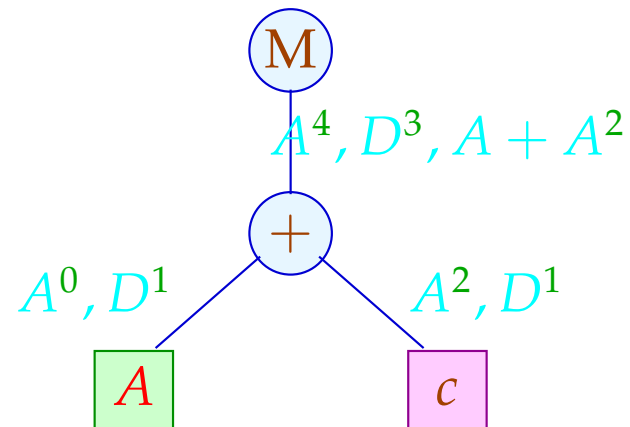


Idee:

Sammele Ableitungen bottom-up auf unter

- \* Kostenkalkulation und
- \* Auswahl.

... im Beispiel:

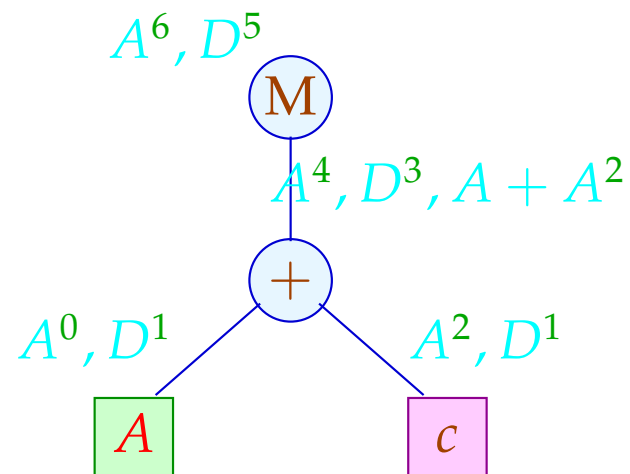


Idee:

Samme Ableitungen bottom-up auf unter

- \* Kostenkalkulation und
- \* Auswahl.

... im Beispiel:

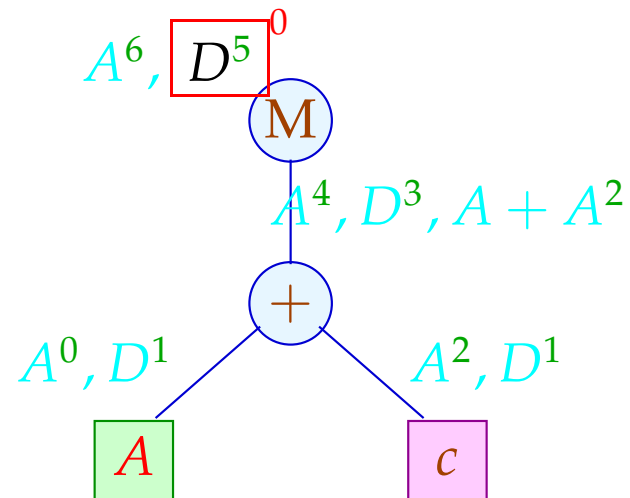


Idee:

Samme Ableitungen bottom-up auf unter

- \* Kostenkalkulation und
- \* Auswahl.

... im Beispiel:

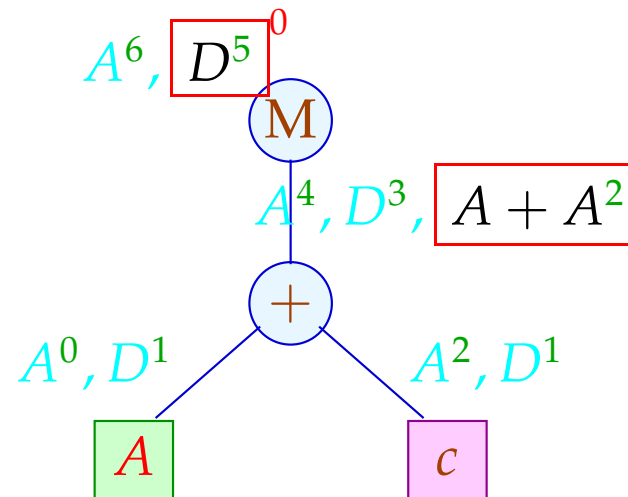


Idee:

Samme Ableitungen bottom-up auf unter

- \* Kostenkalkulation und
- \* Auswahl.

... im Beispiel:

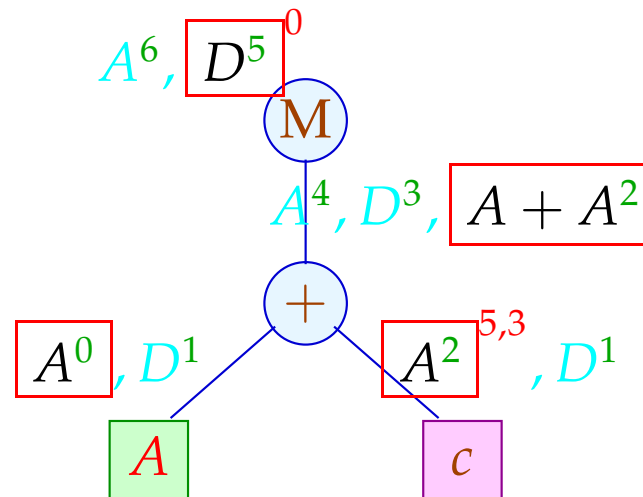


Idee:

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:

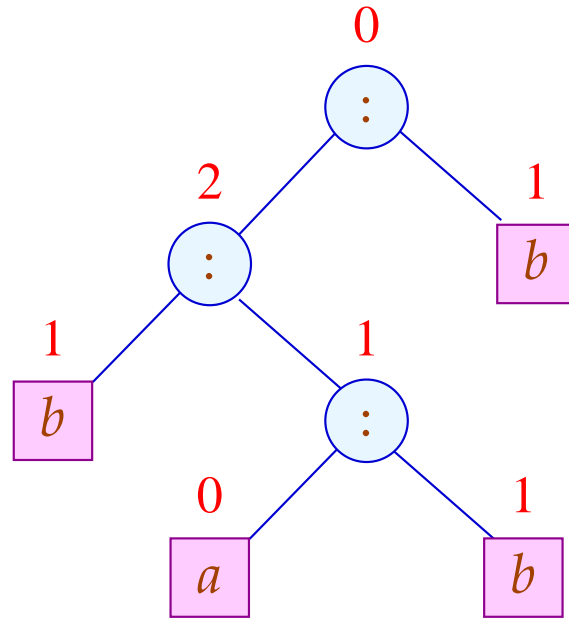
$$Q = \{0, 1, 2\} \quad F = \{0\}$$

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

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

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

// 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** :-)