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 Regelanwendungen, die das Target aus einem Nichtterminal erzeugt ...

Target: $M[A + c]$

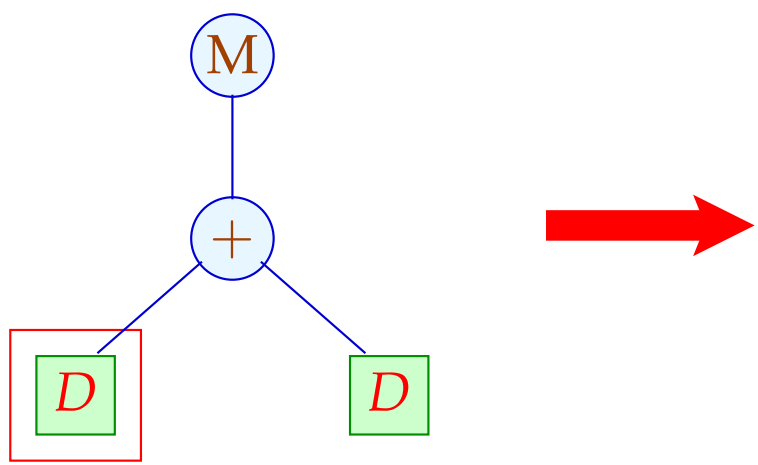
Aufgabe:

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

Target: $M[A + c]$

Aufgabe:

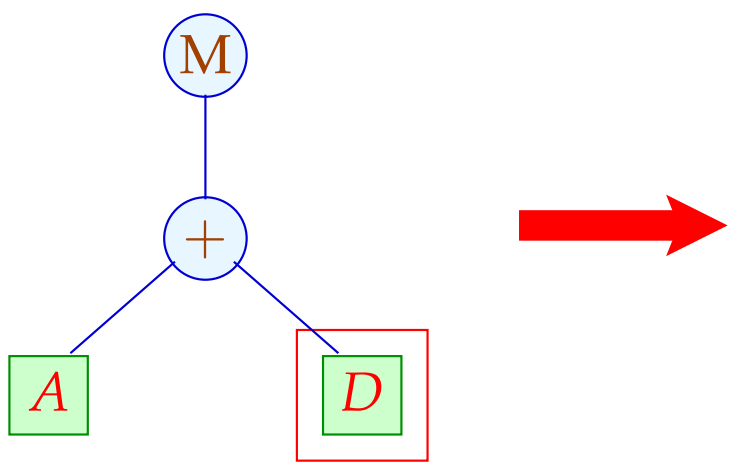
Finde Folge von Regelanwendungen, die das Target aus einem Nichtterminal erzeugt ...



Target: $M[A + c]$

Aufgabe:

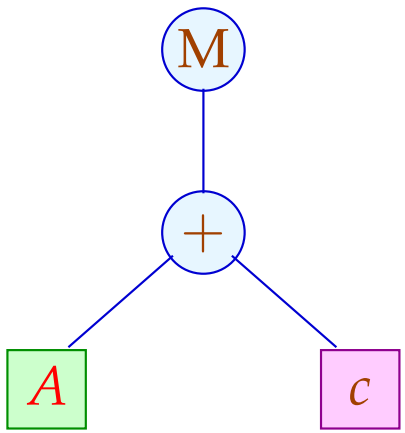
Finde Folge von Regelanwendungen, die das Target aus einem Nichtterminal erzeugt ...



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:

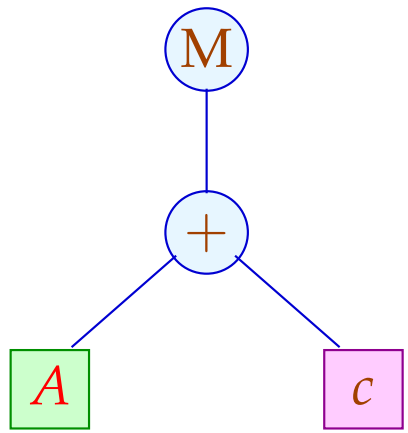
→ Wie durchsuchen wir systematisch die Menge aller Ableitungen ?

→ 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

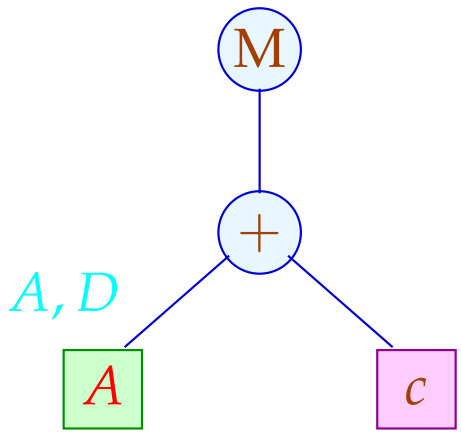
  $\Longrightarrow$ Tree parsing
- Dazu lesen wir die Regeln von rechts nach links ...



Beobachtung:

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

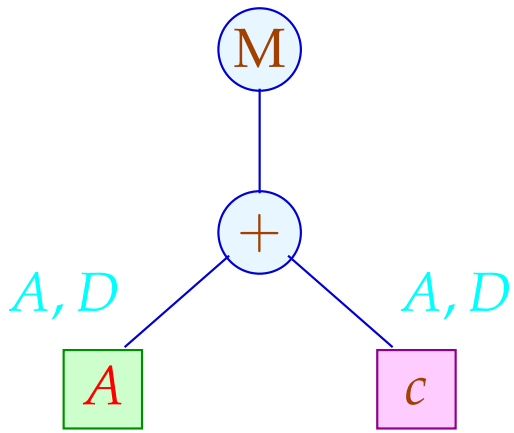
  $\Longrightarrow$ Tree parsing
- Dazu lesen wir die Regeln von rechts nach links ...



Beobachtung:

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

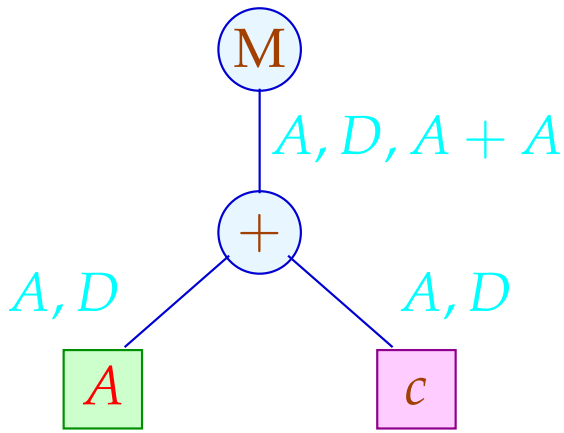
  $\Longrightarrow$ Tree parsing
- Dazu lesen wir die Regeln von rechts nach links ...



Beobachtung:

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

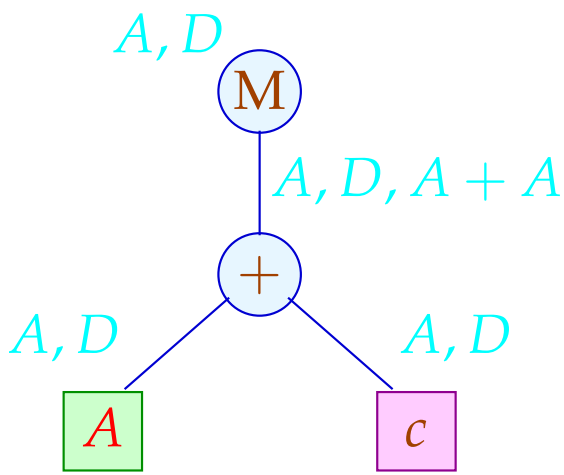
  $\Longrightarrow$ Tree parsing
- Dazu lesen wir die Regeln von rechts nach links ...



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

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

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:

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

Die Hilfsfunktion Move bildet den Abschluss unter Regelanwendungen:

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

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:

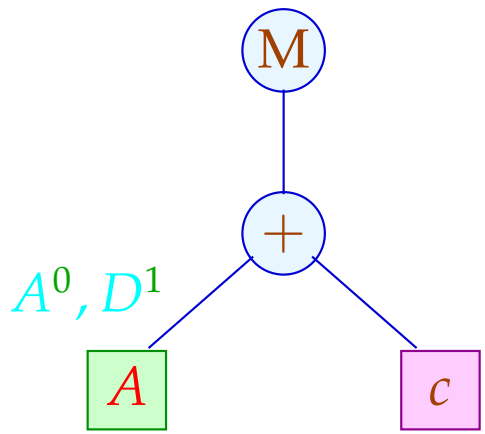
Wähle eine Instruktionsfolge mit minimalen Kosten !

Idee:

Sammle Ableitungen bottom-up auf unter

* Kostenkalkulation und
* Auswahl.

... im Beispiel:

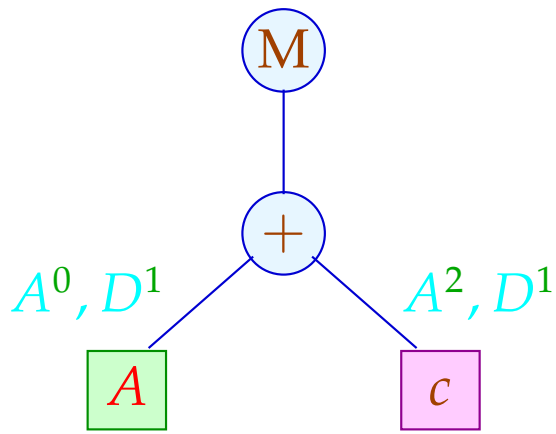


Idee:

Sammle Ableitungen bottom-up auf unter

* Kostenkalkulation und
* Auswahl.

... im Beispiel:

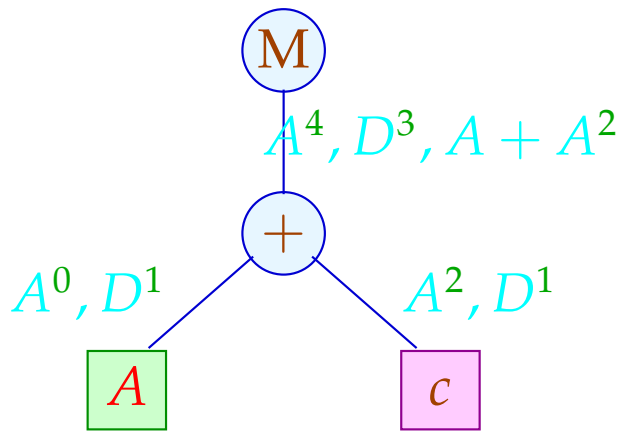


Idee:

Sammle Ableitungen bottom-up auf unter

* Kostenkalkulation und
* Auswahl.

... im Beispiel:

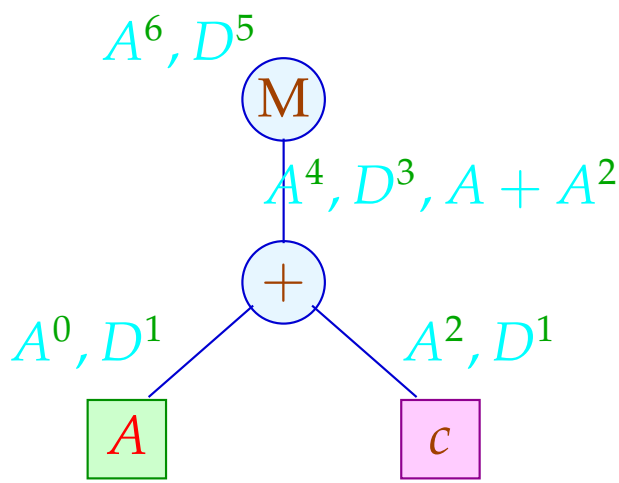


Idee:

Sammle Ableitungen bottom-up auf unter

* Kostenkalkulation und
* Auswahl.

... im Beispiel:

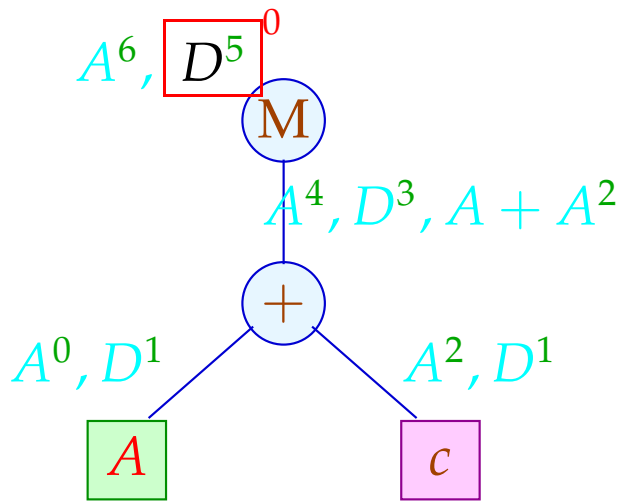


Idee:

Sammle Ableitungen bottom-up auf unter

* Kostenkalkulation und
* Auswahl.

... im Beispiel:

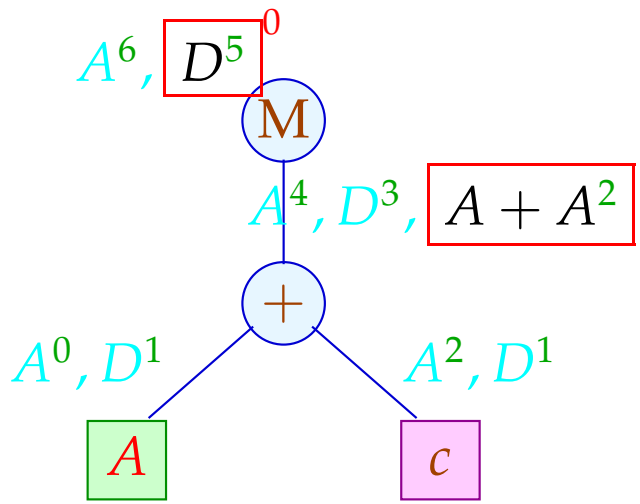


Idee:

Sammle Ableitungen bottom-up auf unter

* Kostenkalkulation und
* Auswahl.

... im Beispiel:

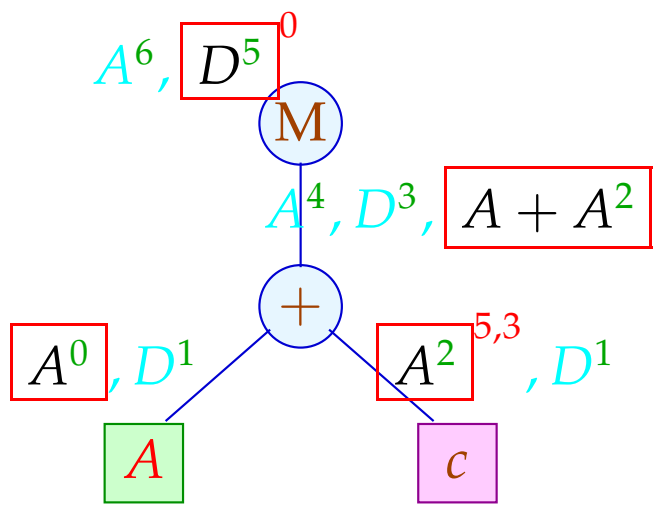


Idee:

Sammle Ableitungen bottom-up auf unter

* Kostenkalkulation und
* Auswahl.

... im Beispiel:



Kostenkalkulation:

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

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

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:

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

mit Kosten 5 . Die Alternative:

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

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:

$$
\begin{array}{lcl}
Q & = & \text{endliche Menge von Zuständen} \\
\Sigma & = & \text{Operatoren und Konstanten} \\
\delta_a & = & \text{Übergangsfunktion für } a \in \Sigma \\
F \subseteq Q & = & \text{akzeptierende Zustände}
\end{array}
$$

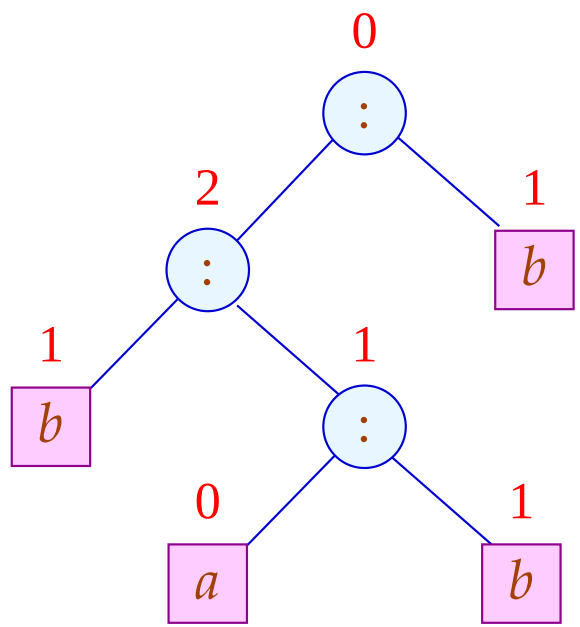
Dabei ist:

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

Beispiel:

$$\begin{array}{lll} Q = \{0,1,2\} & & F = \{0\} \\ \Sigma = \{a,b,:\} & & \\ \delta_a = 0 & & \delta_b = 1 \\ & \delta_:(s_1,s_2) = (s_1+s_2)\%3 & \end{array}$$

// 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 (-:

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

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 $\mathsf{Reg} \cup \mathsf{Term} \cup \{S\}$

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

F $\;=\;$ gewünschter Effekt

δ_R $\;=\;$ $\mathsf{Move}\,\{R\}$

δ_c $\;=\;$ $\mathsf{Move}\,\{c\}$

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