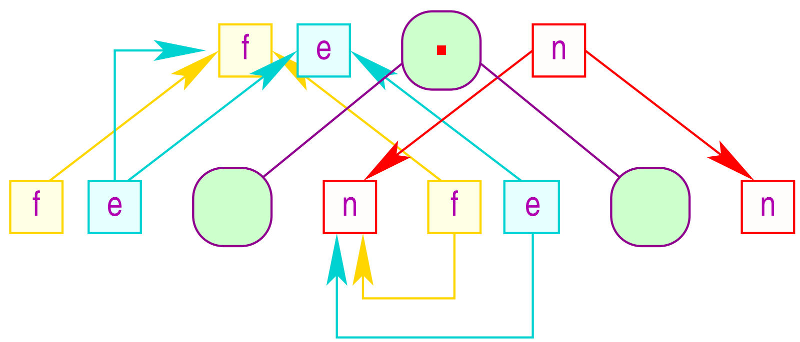
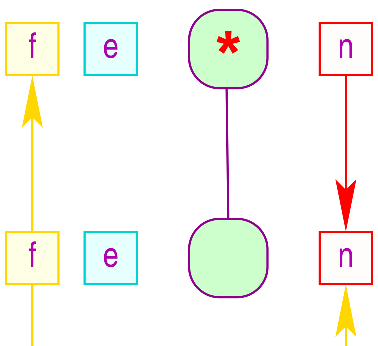
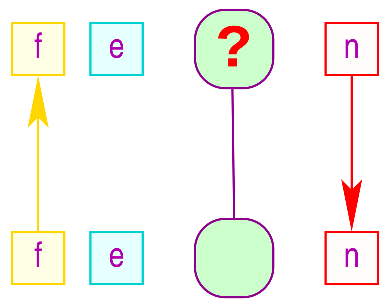
$$\boxed{\cdot} \quad : \quad \begin{aligned} \text{empty}[0] &:= \text{empty}[1] \wedge \text{empty}[2] \\ \text{first}[0] &:= \text{first}[1] \cup (\text{empty}[1])\ ?\ \text{first}[2] : \emptyset \\ \text{next}[1] &:= \text{first}[2] \cup (\text{empty}[2])\ ?\ \text{next}[0] \\ \text{next}[2] &:= \text{next}[0] \end{aligned}$$



$$\boxed{*} \quad : \quad \begin{array}{lll} \text{empty}[0] & := & t \\ \text{first}[0] & := & \text{first}[1] \\ \text{next}[1] & := & \text{first}[1] \cup \text{next}[0] \end{array}$$

$$\boxed{?} \quad : \quad \begin{array}{lll} \text{empty}[0] & := & t \\ \text{first}[0] & := & \text{first}[1] \\ \text{next}[1] & := & \text{next}[0] \end{array}$$





Problem:

- Eine Auswertungsstrategie kann es nur dann geben, wenn die Variablen-Abhängigkeiten in jedem attributierten Baum azyklisch sind !!!
- Es ist DEXPTIME-vollständig, herauszufinden, ob keine zyklischen Variablenabhängigkeiten vorkommen können :-(

Problem:

- Eine Auswertungsstrategie kann es nur dann geben, wenn die Variablen-Abhängigkeiten in jedem attributierten Baum azyklisch sind !!!
- Es ist DEXPTIME-vollständig, herauszufinden, ob keine zyklischen Variablenabhängigkeiten vorkommen können :-(

Ideen:

(1) Die Benutzerin soll die Strategie spezifizieren ;-)

(2) Bestimme die Strategie dynamisch ;-}

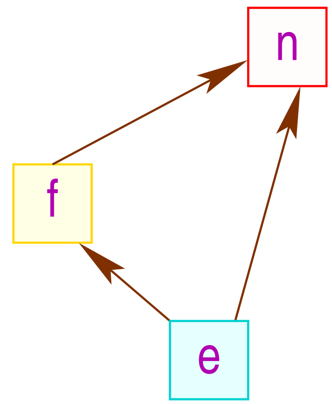
(3) Betrachte Teilklassen ...

Stark azyklische Attributierung:

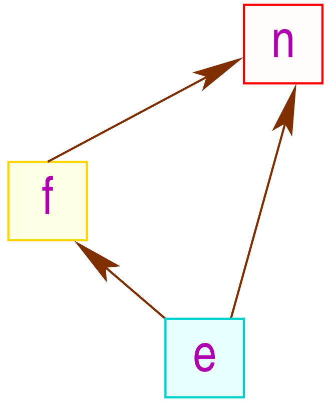
Berechne eine partielle Ordnung auf den Attributen eines Knotens, die kompatibel mit den lokalen Attribut-Abhängigkeiten ist:

- Wir starten mit der trivialen Ordung $\sqsubseteq \; = \; =$:-)
- Die aktuelle Ordnung setzen wir an den Sohn-Knoten in die lokalen Abhängigkeitsgraphen ein.
- Ergibt sich ein Kreis, geben wir auf :-))
- Andernfalls fügen wir alle Beziehungen $a \sqsubseteq b$ hinzu, für die es jetzt einen Pfad von $a[0]$ nach $b[0]$ gibt.
- Lässt sich $\sqsubseteq$ nicht mehr vergrößern, hören wir auf ...

... im Beispiel:



... im Beispiel:



Diskussion:

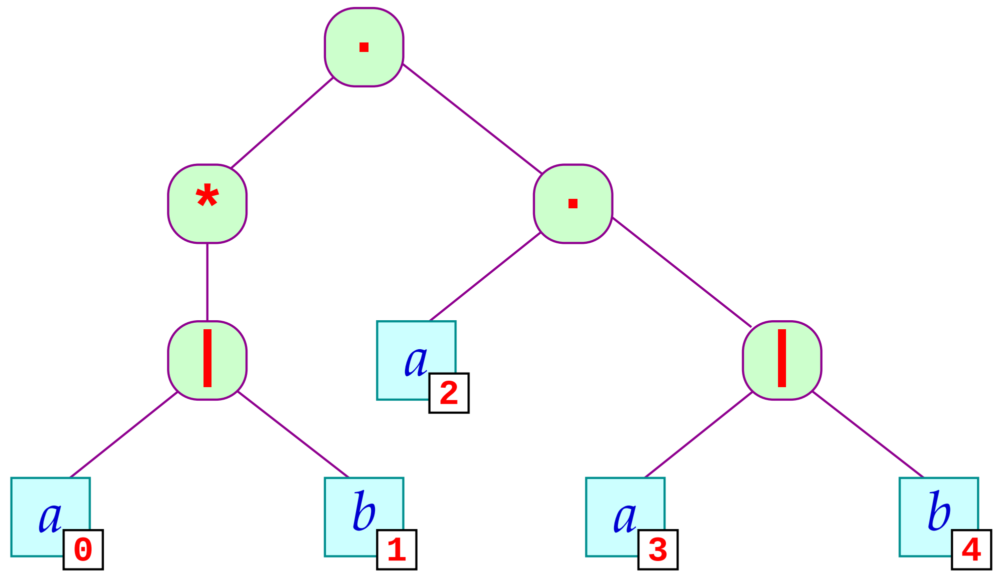
- Die Berechnung der partiellen Ordnung $\sqsubseteq$ ist eine Fixpunkt-Berechnung :-)
- Die partielle Ordnung können wir in eine lineare Ordnung einbetten ...
- Die lineare Ordnung gibt uns an, in welcher Reihenfolge die Attribute berechnet werden müssen :-)
- Die lokalen Abhängigkeitsgraphen zusammen mit der linearen Ordnung erlauben die Berechnung einer Strategie ...

Mögliche Strategien:

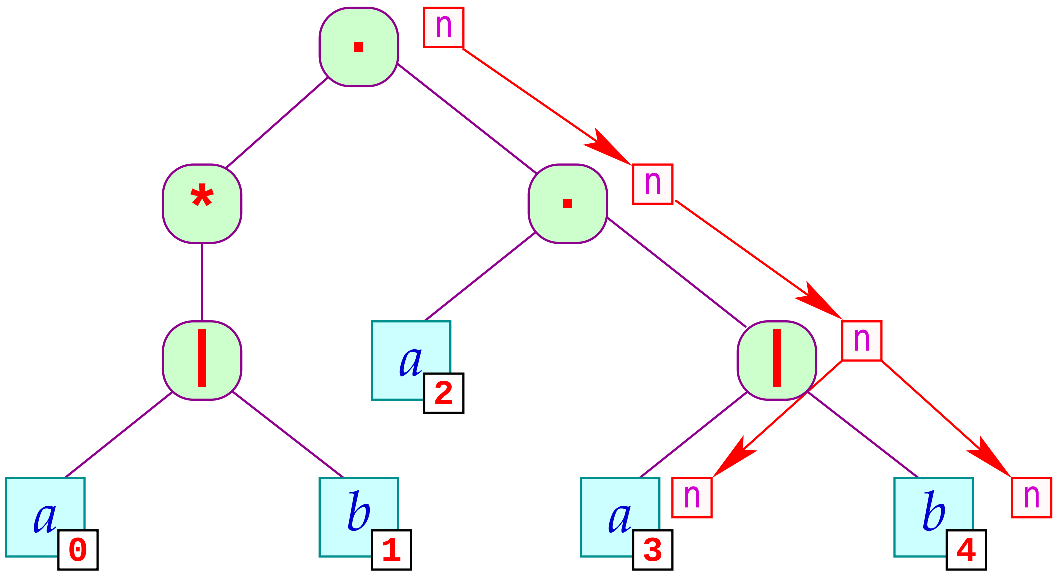
(1) Bedarfsgetriebene Auswertung:

- Beginne mit der Berechnung eines Attributs.
- Sind die Argument-Attribute noch nicht berechnet, berechne rekursiv deren Werte :-)
- Besuche die Knoten des Baum nach Bedarf...

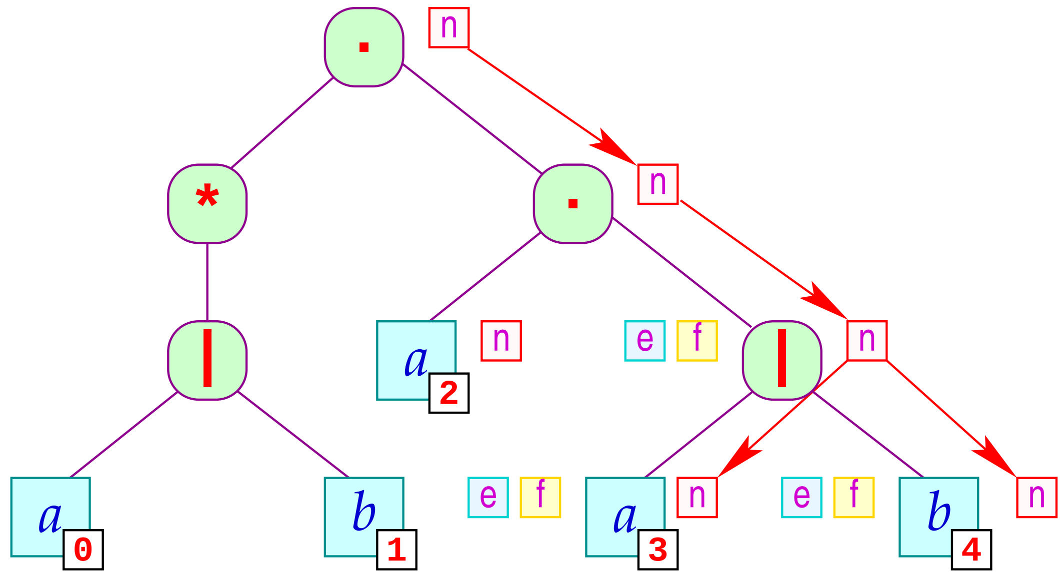
Beispiel, bedarfsgetrieben:



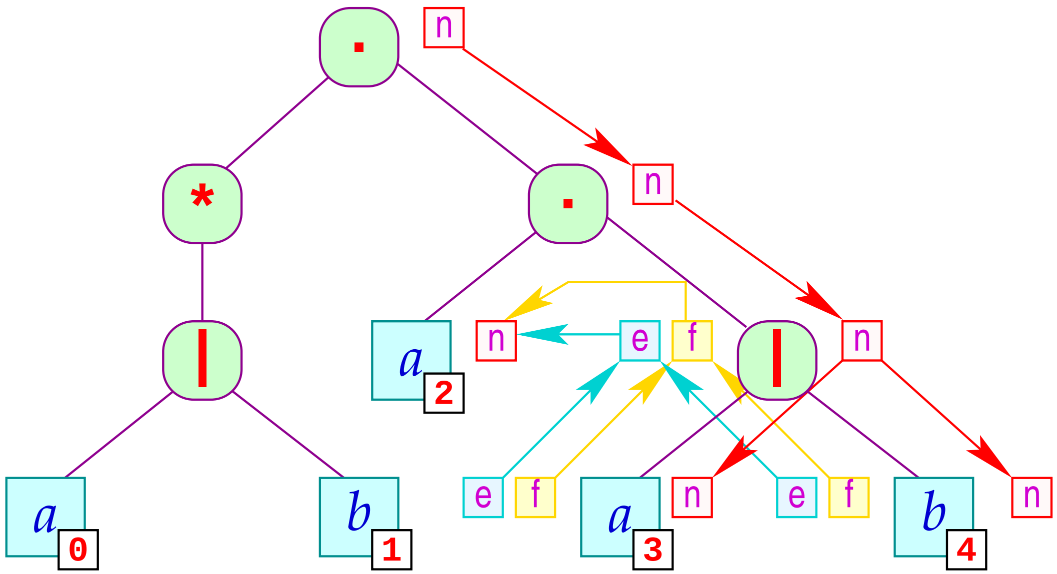
Beispiel, bedarfsgetrieben:



Beispiel, bedarfsgetrieben:



Beispiel, bedarfsgetrieben:



Diskussion:

- Die Reihenfolge hängt i.a. vom zu attributierenden Baum ab.
- Der Algorithmus muss sich merken, welche Attribute er bereits berechnete :-(
- Der Algorithmus besucht manche Knoten unnötig oft.
- Der Algorithmus ist nicht-lokal :-((

Mögliche Strategien (Forts.):

(2) Auswertung in Pässen:

- Minimiere die Anzahl der Besuche an jedem Knoten.
- Organisiere die Auswertung in Durchläufe durch den Baum.
- Berechne für jeden Pass eine Besuchsstrategie für die Knoten zusammen mit einer lokalen Strategie für jeden Knoten-Typ ...

Achtung:

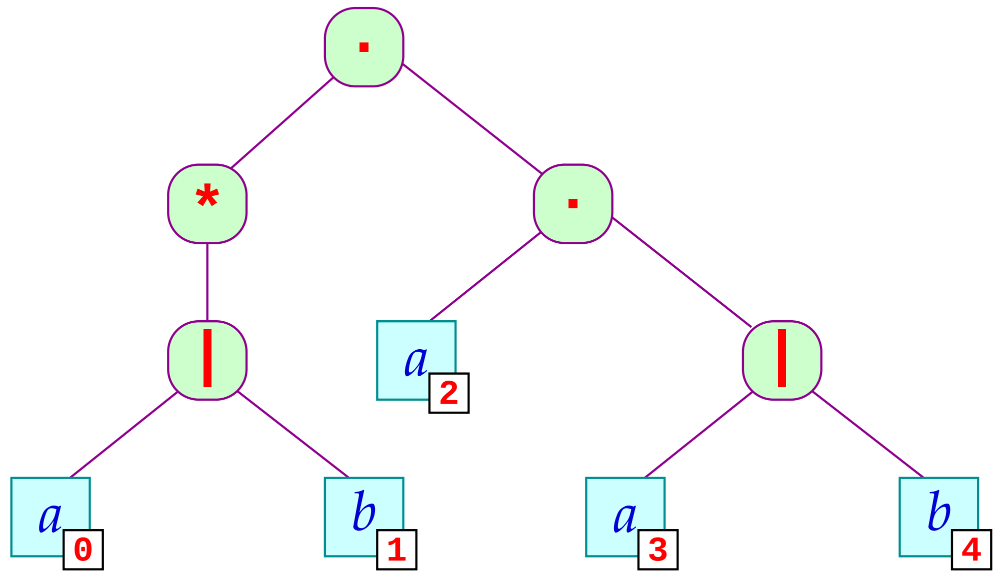
- Das minimale Attribut in der Anordnung für stark azyklische Attributierungen lässt sich stets in einem Pass berechnen :-)
- Man braucht folglich für stark azyklische Attributierungen maximal so viele Pässe, wie es Attribute gibt :-))
- Hat man einen Baum-Durchlauf zur Berechnung einiger Attribute, kann man überprüfen, ob er geeignet ist, gleichzeitig weitere Attribute auszuwerten $\Longrightarrow$ Optimierungsproblem

... im Beispiel:

empty und first lassen sich gemeinsam berechnen.

next muss in einem weiteren Pass berechnet werden :-(

Weiteres Beispiel: Nummerierung der Blätter eines Baums:

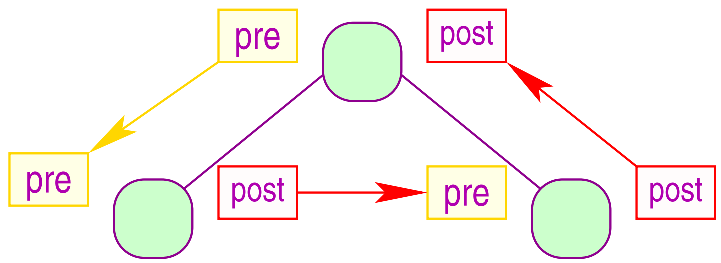


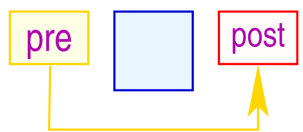
Idee:

- Führe Hilfsattribute pre und post ein !
- Mit pre reichen wir einen Zählerstand nach unten
- Mit post reichen wir einen Zählerstand wieder nach oben ...

$$
\begin{array}{llll}
\text{Root:} & \text{pre}[0] & := & 0 \\
 & \text{pre}[1] & := & \text{pre}[0] \\
 & \text{post}[0] & := & \text{post}[1] \\
 & & & \\
\text{Node:} & \text{pre}[1] & := & \text{pre}[0] \\
 & \text{pre}[2] & := & \text{post}[1] \\
 & \text{post}[0] & := & \text{post}[2] \\
 & & & \\
\text{Leaf:} & \text{post}[0] & := & \text{pre}[0] + 1
\end{array}
$$

... die lokalen Attribut-Abhängigkeiten:





- Die Attributierung ist offenbar stark azyklisch :-)
- Man kann alle Attribute in einem Links-Rechts-Durchlauf auswerten :-))
- So etwas nennen wir L-Attributierung.
- L-Attributierung liegt auch unseren Query-Tools zur Suche in XML-Dokumenten zugrunde $\Longrightarrow$ fxgrep

Praktische Erweiterungen:

- Symboltabellen, Typ-Überprüfung / Inferenz und (einfache) Codegenerierung können durch Attributierung berechnet werden :-)
- In diesen Anwendungen werden stets Syntaxbäume annotiert.
- Die Knoten-Beschriftungen entsprechen den Regeln einer kontextfreien Grammatik :-)
- Knotenbeschriftungen können in Sorten eingeteilt werden — entsprechend den Nichtterminalen auf der linken Seite ...
- Unterschiedliche Nichtterminale benötigen evt. unterschiedliche Mengen von Attributen.
- Eine attributierte Grammatik ist eine CFG erweitert um:
  - → Attribute für jedes Nichtterminal;
  - → lokale Attribut-Gleichungen.
- Damit können die syntaktische, Teile der semantischen Analyse wie der Codeerzeugung generiert werden :-)

4 Die Optimierungsphase

1. Vermeidung überflüssiger Berechnungen

   $\longrightarrow$ verfügbare Ausdrücke

   $\longrightarrow$ Konstantenpropagation/Array-Bound-Checks

   $\longrightarrow$ Code Motion

2. Ersetzen teurer Berechnungen durch billige

   $\longrightarrow$ Peep Hole Optimierung

   $\longrightarrow$ Inlining

   $\longrightarrow$ Reduction of Strength

   ...

3. Anpassung an Hardware

   $\longrightarrow$ Instruktions-Selektion

   $\longrightarrow$ Registerverteilung

   $\longrightarrow$ Scheduling

   $\longrightarrow$ Speicherverwaltung