Karlsruher Institut für Technologie

Praktikum Compilerbau
Sitzung 4 – Abstrakter Syntaxbaum

Prof. Dr.-Ing. Gregor Snelting
Andreas Zwinkau

IPD Snelting, Lehrstuhl für Programmierparadigmen

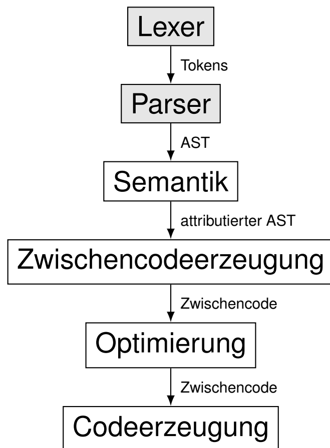
Letzte Woche

Karlsruher Institut für Technologie

- Was waren eure Erfahrungen?

Compilerphasen

Karlsruher Institut für Technologie



Aufgaben für diese Woche

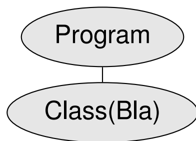
Karlsruher Institut für Technologie

- Entwurf des AST als abstrakte Algebra.
- Implementierung als reale Datenstrukturen.
- Anknüpfen der AST-Erzeugung an den Parser.

Was gehört in den AST?

- Semantisch unwichtige Dinge können weggelassen werden

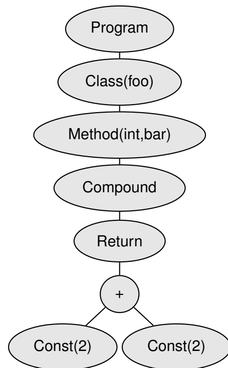
class Bla { }



Was gehört in den AST?

- Code ist kompositional
- AST stellt Hierarchie dar

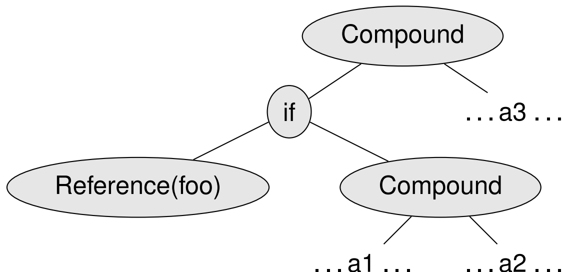
```
class Foo {
  public int bar() { return 2 + 2; }
}
```



Was gehört nicht in den AST?

- Wörter die Konstrukte voneinander trennen

if (foo) { a1(); a2(); } a3();

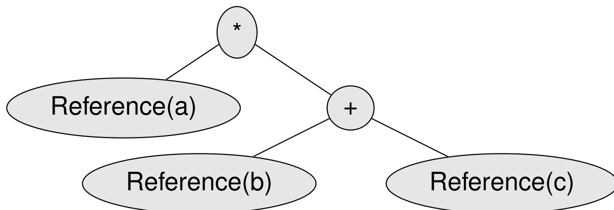


Was gehört nicht in den AST?

- Wörter die Mehrdeutigkeiten in der Hierarchie verhindern

$$a*(b+c)$$

Keine Klammern:



Syntax als abstrakte Algebra

Karlsruher Institut für Technologie

Beispiel: abstrakte Syntax für Expressions und Statements

$$
\begin{aligned}
\mathit{Stmt} &= \mathit{IfStmt} \mid \mathit{IfElseStmt} \mid \mathit{WhileStmt} \mid \mathit{Assignment} \mid \mathit{Block} \ldots \\
\mathit{IfStmt} &:: \mathit{Expr}\ \mathit{Stmt} \\
\mathit{IfElseStmt} &:: \mathit{Expr}\ \mathit{Stmt}\ \mathit{Stmt} \\
\mathit{WhileStmt} &:: \mathit{Expr}\ \mathit{Stmt} \\
\mathit{Block} &:: \mathit{Decls}\ \mathit{StmtList} \\
\mathit{StmtList} &:: \mathit{Stmt}\ + \\
\mathit{Assignment} &:: \mathit{Var}\ \mathit{Expr} \\
\mathit{Var} &:: \texttt{Symbol}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Expr} &= \mathit{Addop} \mid \mathit{MultOp} \mid \mathit{Var} \mid \ldots \\
\mathit{Addop} &:: \mathit{Expr}\ \mathit{Expr} \\
\mathit{Multop} &:: \mathit{Expr}\ \mathit{Expr}
\end{aligned}
$$

siehe Vorlesung *Sprachtechnologie und Compiler*

Implementierung abstrakte Syntax

Objektorientiert:

- je 1 Klasse pro syntaktische Kategorie
- Alternativregeln

  $$X = X1 \mid X2 \mid \ldots$$

  werden zu Unterklassen:

  ```
  class X { /* ... */ }
  class X1 extends X { /* ... */ }
  class X2 extends X { /* ... */ }
  ```

- Baumaufbauregeln

  $$X :: Y1\ Y2$$

  werden zu Konstruktorfunktionen:

  ```
  class X {
      public X(Y1 y1, Y2 y2) { /* ... */ }
  }
  ```

Tipps

Karlsruher Institut für Technologie

- Nicht jede Produktion der Grammatik muss ein eigener AST-Knoten werden!
- Gemeinsame Basisklassen sinnvoll wo Alternativen in der Grammatik vorhanden sind.
  - Statements
  - Expressions
  - Types?
  - ClassMember?

Weitere Fragen

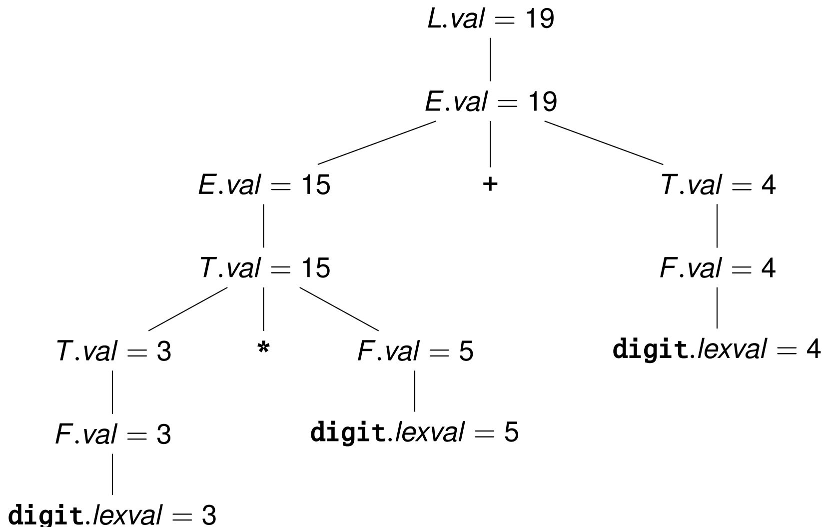
Karlsruher Institut für Technologie

- Braucht man Verweise auf das Quellprogramm?
  - Warum (nicht)?
- Was sollte ein Attribut werden, was ein eigener Knoten im AST?
  - Wie ist das bei Typen oder Bezeichnern?

Beispiel: Taschenrechner mit Attributierter Grammatik

	Produktion	Semantische Regeln
1)	$L \rightarrow E$	$L.val = E.val$
2)	$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
3)	$E \rightarrow T$	$E.val = T.val$
4)	$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
5)	$T \rightarrow F$	$T.val = F.val$
6)	$F \rightarrow (E)$	$F.val = E.val$
7)	$F \rightarrow \mathbf{digit}$	$F.val = \mathbf{digit}.lexval$

Attributierter Parsebaum für 3 * 5 + 4



Implementierung Attributierter Grammatiken

Karlsruher Institut für Technologie

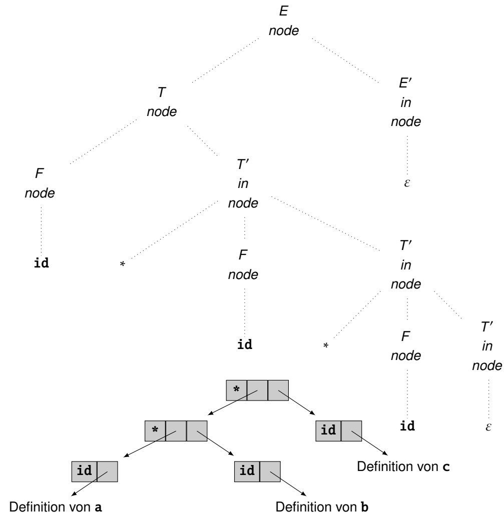
- ererbte Attribute werden zu Parameter
- synthetisierte Attribute werden zu Rückgabewerten

AST-Aufbau für $\mathrm{LL}(1)$-Beispielgrammatik

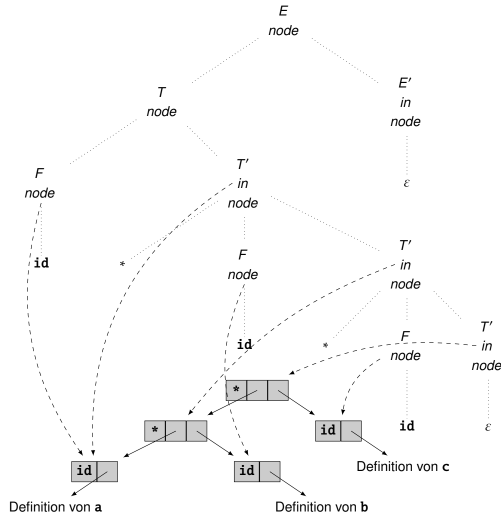
Karlsruher Institut für Technologie

	Produktion	Semantische Regeln
1)	$E \rightarrow T\ E'$	$E.node = E'.node$
		$E'.in = T.node$
2)	$E' \rightarrow \varepsilon$	$E'.node = E'.in$
3)	$E'_1 \rightarrow \mathbf{+}\ T\ E'_2$	$E'_2.in = \texttt{new Node}(\mathbf{+}, E'_1.in, T.node)$
		$E'_1.node = E'_2.node$
4)	$T \rightarrow F\ T'$	$T.node = T'.node$
		$T'.in = F.node$
5)	$T' \rightarrow \varepsilon$	$T'.node = T'.in$
6)	$T'_1 \rightarrow \mathbf{*}\ F\ T'_2$	$T'_2.in = \texttt{new Node}(\mathbf{*}, T'_1.in, F.node)$
		$T'_1.node = T'_2.node$
7)	$F \rightarrow \mathbf{id}$	$F.node = \texttt{new Leaf}(\mathbf{id}, \mathbf{id}.entry)$
8)	$F \rightarrow \mathbf{(}\ E\ \mathbf{)}$	$F.node = E.node$

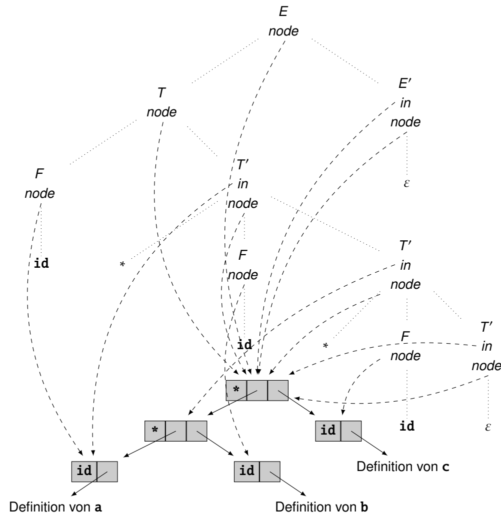
AST für $a * b * c$



AST für $a * b * c$



AST für $a * b * c$



Feedback! Fragen? Probleme?

Karlsruher Institut für Technologie

- Wie läuft die Arbeitseinteilung?
- Anmerkungen?
- Probleme?
- Fragen?