Von der Grammatik zum AST

- Welche Eigenschaften soll ein Parser haben?
- Wann ist eine Grammatik eindeutig?
- Wie sollte eine Grammatik aussehen?
- Theoretischer Hin tergrund: FIRST, FOLLOW
- Einschränkungen von LL(1)
- Praktischer Teil: LR(1) Grammatik mit java cup
- Und natürlich: Diskussion der Beispielsprache

Links:

Parser: http://www2.cs.tum.edu/projects/cup/

Scanner http://jflex.de/

Eclipse-Plugin (available software site): http://www2.in.tum.de/projects/cup/eclipse

Anforderungen an Parser

- Einfache Formulierung der Grammatik
- Einfache Erzeugung des Abstrakten Syntaxbaums
- Theorie sagt: Ein Satz gehört zur Sprache, wenn er abgeleitet werden kann. Was heißt das für mögliche Algorithmen?
- Wie ist es mit Backtracking? (Denken Sie an Programmier-Paradigmen)
- Wie weit schauen wir voraus, wenn wir (unviderrufliche) Entscheidungen treffen?

Eindeutigkeit

$E \rightarrow E$ **+** T

$E \rightarrow T$

$T \rightarrow T$ ***** **NUM**

$T \rightarrow$ **NUM**

LL

E

E **+** T

T **+** T

NUM **+** T

NUM **+** T ***** **NUM**

NUM **+** **NUM** ***** **NUM**

LR

E

E **+** T

E **+** T ***** **NUM**

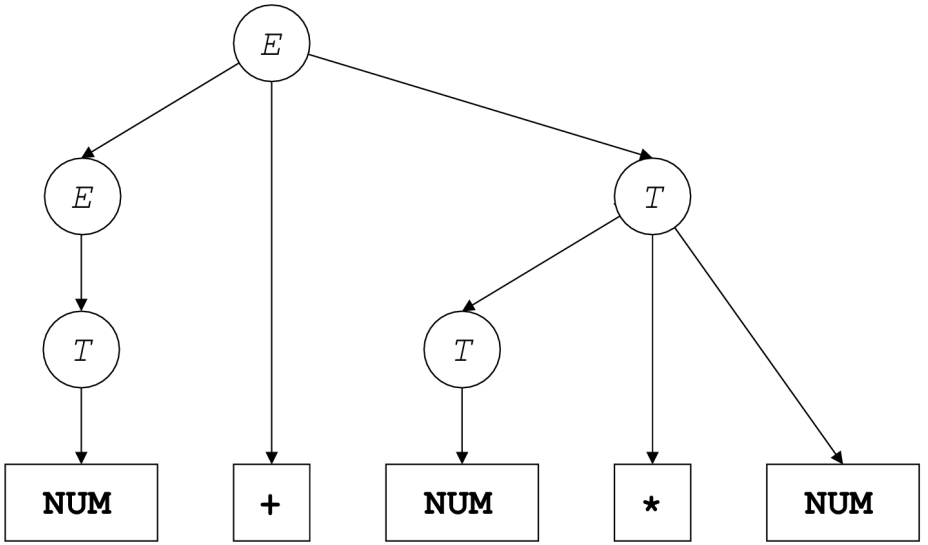
E **+** **NUM** ***** **NUM**

T **+** **NUM** ***** **NUM**

NUM **+** **NUM** ***** **NUM**

Ist diese Grammatik eindeutig?
Muss eine Grammatik eindeutig sein
Wie können wir Eindeutigkeit definieren?

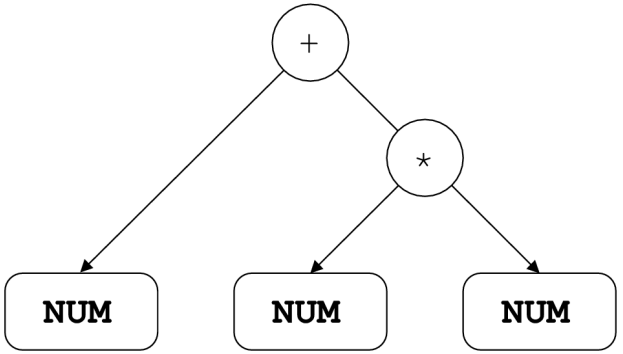
Kontextfreie Grammatik -> Syntaxbaum (Parse Tree)



- Welcher Ableitung entspricht der Baum?
- Gibt es Grammatiken, die zu mehreren Syntaxbäumen führen?
- Ist der Baum sinnvoll strukturiert? Wenn ja, warum?
- Kann man den Baum vereinfachen?

Definition: Eine Grammatik ist mehrdeutig, wenn es für einen Satz unterschiedliche Syntaxbäume gibt.

Der Abstrakte Syntaxbaum (AST)



- Beim Abstrakten Syntaxbaum fehlen alle unwesentlichen, grammatikspezifischen Elemente
- Der Abstrakte Syntaxbaum gibt die logische Struktur des Ausdrucks wieder
- Der Syntaxbaum kann einfach (rekursiv) interpretiert werden
- Deklarationen von Bezeichnern erfordern eine Symboltabelle
- Kann der Syntaxbaum einfach erzeugt werden?

Attributierte Grammatik

E:a → E:b + T:c	a = +(b, c)
E:a → T:b	a = b
T:a → T:b * NUM:c	a = *(b, NUM(c))
T:a → NUM:b	a = Num(b)

- Attributierte Grammatik: Symbole haben ein Attribut
- Terminalsymbole: Scanner bestimmt das Attribut
- Nichtterminalsymbole: Attribute aus anderen Attributen ermittelt (Semantikfunktion)
- Synthetisierte Attribute: Attribut der linken Seite ist Funktion der rechten Seite
- Schwerpunkt (von meiner Seite her) weniger die perfekte Generierung von optimalem Code als die exemplarische Untersuchung moderner Techniken
- Ideal: Parser baut direkt den AST auf!
- Nicht alle Parser erlauben immer die Berechnung synthetisierter Attribute!!!

Analyse einer Regel: FIRST-Menge

Es ist nicht einfach einen effizienten Algorithmus zu definieren.

Deklarativ kann man die FIRST-Menge wie folgt beschreiben:

- Die FIRST-Menge ordnet einer Symbolfolge α eine Menge von Terminalsymbolen und dem leeren Symbol ε zu.
- Informativ: Menge der "Anfangssymbole"
- Herleitungsregeln:
  - Regel 1: FIRST(T α) = { T }, T = Terminalsymbol, α = beliebig
  - Regel 2: FIRST(ε) = { ε }
  - Regel 2: FIRST(N) = $\cup$ FIRST(R_i), mit $N \rightarrow R_i$
  - Regel 3: FIRST(Nα) = FIRST(N), wenn $\varepsilon \notin$ FIRST(N)
    = FIRST(N) \ ε $\cup$ FIRST(α), wenn $\varepsilon \in$ FIRST(N)

Analyse einer Regel: Follow-Menge

Deklarativ kann man die FOLLOW-Menge wie folgt beschreiben:

- Die FOLLOW-Menge ist die Menge der in einer korrekten Ableitung evtl. folgenden Terminalsymbole (ohne .
- Informativ: Menge der "Folgesymbole"
- Herleitungsregeln:
  - Regel 1: { $ } $\subset$ FOLLOW(S), S = Startsymbol, $ Satzende
  - Regel 2: FIRST(β) \ ε $\subset$ FOLLOW(N) , für X -> α N β
  - Regel 3: FOLLOW(X) $\subset$ FOLLOW(N), für X -> α N β und $\varepsilon \in F\beta$

LL(1) - Parser

Idee von LL(1)

- Man beginnt mit dem Startsymbol (prädikativ, top down)
- Das jeweils linkeste Nichtterminalsymbol wird abgeleitet
- Zur Regelauswahl wird 1 Voraussschausymbol verwendet (lookahead)
- Terminalsymbole werden "gematched"

Der entscheidende Punkt ist die Regelauswahl für aktuelles NT und aktuelles lookahead:

- Eine Regel wird gewählt, wenn das Voirausschausymbol in der FIRST-Menge ist
- Ist e in der FIRST-Menge enthalten, trifft die Regel für die Elemente der FOLLOW-Menge zu

Können gleichzeitig 2 Regeln verwendet werden, hat man einen Konflikt: die Grammatik ist durch einen LL(1)-Parser nicht zu erkennen