LL-Parser

BC George (HSBI)

Wiederholung

PDAs und kontextfreie Grammatiken

- Warum reichen uns DFAs nicht zum Matchen von Eingabezeichen?
- Wie könnnen wir sie minimal erweitern?
- Sind PDAs deterministisch?
- Wie sind kontextfreie Grammatiken definiert?
- Sind kontextfreie Grammatiken eindeutig?

Motivation

Was brauchen wir für die Syntaxanalyse von Programmen?

- einen Grammatiktypen, aus dem sich manuell oder automatisiert ein Programm zur deterministischen Syntaxanalyse (=Parser) erstellen lässt
- einen Algorithmus zum Parsen von Programmen mit Hilfe einer solchen Grammatik

Themen für heute

- Automatische Generierung von Top-Down-Parsern aus LL-Grammatiken

Syntaxanalyse

Syntax

Wir verstehen unter Syntax eine Menge von Regeln, die die Struktur von Daten (z. B. Programmen) bestimmen.

Syntaxanalyse ist die Bestimmung, ob Eingabedaten einer vorgegebenen Syntax entsprechen.

Diese vorgegebene Syntax wird im Compilerbau mit einer kontextfreien Grammatik beschrieben und mit einem sogenannten **Parser** analysiert.

Wir beshäftigen uns heute mit LL-Parsing, mit dem man eine Teilmenge der eindeutigen kontextfreien Grammatiken syntaktich analysieren kann.

Der Ableitungsbaumwird von oben nach unten aufgebaut.

Ziele der Syntaxanalyse

- aussagekräftige Fehlermeldungen, wenn ein Eingabeprogramm syntaktisch nicht korrekt ist
- evtl. Fehlerkorrektur
- Bestimmung der syntaktischen Struktur eines Programms
- Erstellung des AST (abstrakter Syntaxbaum): Der Parse Tree ohne Symbole, die nach der Syntaxanalyse inhaltlich irrelevant sind (z. B. Semikolons, manche Schlüsselwörter)
- die Symboltablelle(n) mit Informationen bzgl. Bezeichner (Variable, Funktionen und Methoden, Klassen, benutzerdefinierte Typen, Parameter, ...), aber auch die Gültigkeitsbereiche.

LL(k)-Grammatiken

First-Mengen

$S \to A \mid B \mid C$

Welche Produktion nehmen?

Wir brauchen die "terminalen k-Anfänge" von Ableitungen von Nichtterminalen, um eindeutig die nächste zu benutzende Produktion festzulegen. k ist dabei die Anzahl der Vorschautoken.

Def.: Wir definieren *First* - Mengen einer Grammatik wie folgt:

- $a \in T^*, |a| \leq k : First_k(a) = \{a\}$
- $a \in T^*, |a| > k : First_k(a) = \{v \in T^* \mid a = vw, |v| = k\}$
- $\alpha \in (N \cup T)^* \backslash T^* : First_k(\alpha) = \{v \in T^* \mid \alpha \overset{*}{\Rightarrow} w, \text{mit } w \in T^*, First_k(w) = \{v\}\}$

Linksableitungen

Def.: Bei einer kontextfreien Grammatik G ist die *Linksableitung* von $\alpha \in (N \cup T)^*$ die Ableitung, die man erhält, wenn in jedem Schritt das am weitesten links stehende Nichtterminal in α abgeleitet wird.

Man schreibt $\alpha \overset{*}{\Rightarrow}_l \beta$.

LL(k)-Grammatiken

Def.: Eine kontextfreie Grammatik $G = (N, T, P, S)$ ist genau dann eine *LL(k)*-Grammatik, wenn für alle Linksableitungen der Form:

$S \overset{*}{\Rightarrow}_l \ wA\gamma \Rightarrow_l w\alpha\gamma \overset{*}{\Rightarrow}_l wx$

und

$S \overset{*}{\Rightarrow}_l wA\gamma \Rightarrow_l w\beta\gamma \overset{*}{\Rightarrow}_l wy$

mit $(w, x, y \in T^*, \alpha, \beta, \gamma \in (N \cup T)^*, A \in N)$ und $First_k(x) = First_k(y)$ gilt:

$\alpha = \beta$

LL(1)-Grammatiken

LL(k)-Sprachen

Die von *LL(k)*-Grammatiken erzeugten Sprachen sind eine echte Teilmenge der deterministisch parsbaren Sprachen.

Die von *LL(k)*-Grammatiken erzeugten Sprachen sind eine echte Teilmenge der von *LL(k+1)*-Grammatiken erzeugten Sprachen.

Für eine kontextfreie Grammatik G ist nicht entscheidbar, ob es eine *LL(1)* - Grammatik G' gibt mit $L(G) = L(G')$.

In der Praxis reichen $LL(1)$ - Grammatiken oft. Hier gibt es effiziente Parsergeneratoren (hier: ANTLR), deren Eingabe eine LL(k)- (meist LL(1)-) Grammatik ist, und die als Ausgabe den Quellcode eines (effizienten) tabellengesteuerten Parsers generieren.

Algorithmus: Konstruktion einer LL-Parsertabelle

Eingabe: Eine Grammatik G = (N, T, P, S)

Ausgabe: Eine Parsertabelle *P*

```
for each production X → α
    for each a ∈ First(α)
        add X → α to P[X, a]

    if ϵ ∈ First(α)
        for each b ∈ Follow(α)
            add X → α to P[X, b]
            if ϵ ∈ First(α) and ⊥ ∈ Follow(X)
                add X → α to P[X, ⊥]
```

Abbildung 1: Algorithmus zur Generierung einer LL-Parsertabelle

Hier ist $\perp$ das Endezeichen des Inputs. Statt $First_1(\alpha)$ wird oft nur $First(\alpha)$ geschrieben.

LL-Parsertabellen

LL-Parsertabellen

Rekursive Programmierung bedeutet, dass das Laufzeitsystem einen Stack benutzt. Diesen Stack kann man auch “selbst programmieren”, d. h. einen PDA implementieren. Dabei wird ebenfalls die oben genannte Tabelle zur Bestimmung der nächsten anzuwendenden Produktion benutzt. Der Stack enthält die zu erwartenden Eingabezeichen, wenn immer eine Linksableitung gebildet wird. Diese Zeichen im Stack werden mit dem Input gematcht.

Algorithmus: Tabellengesteuertes LL-Parsen mit einem PDA

Eingabe: Eine Grammatik G = (N, T, P, S), eine Parsertabelle P mit "$w \perp$" als initialem Kellerinhalt

Ausgabe: Wenn $w \in L(G)$, eine Linksableitung von w, Fehler sonst

```
a = next_token()
X = top of stack  // entfernt X vom Stack

while X ≠⊥

   if X = a
      a = next_token()

   else if X ∈ T
      error

   else if P[X, a] leer
      error

   else if P[X, a] = X → Y1Y2 ... Yk
      process_production(X → Y1Y2 ... Yk)
      push(Y1Y2 ... Yk) //Y1 = top of stack

   X = top of stack
```

Abbildung 2: Algorithmus zum tabellengesteuerten LL-Parsen

Ergebnisse der Syntaxanalyse

- eventuelle Syntaxfehler mit Angabe der Fehlerart und des -Ortes
- Fehlerkorrektur
- Format für die Weiterverarbeitung:
  - Ableitungsbaum oder Syntaxbaum oder Parse Tree
  - abstrakter Syntaxbaum (AST): Der Parse Tree ohne Symbole, die nach der Syntaxanalyse inhaltlich irrelevant sind (z. B. ;, Klammern, manche Schlüsselwörter, ...)
- Symboltabelle

Wrap-Up

- Syntaxanalyse wird mit deterministisch kontextfreien Grammatiken durchgeführt.
- Eine Teilmenge der dazu gehörigen Sprachen lässt sich top-down parsen.
- Ein effizienter LL(k)-Parser realisiert einen DPDA und kann automatisch aus einer LL(k)-Grammatik generiert werden.
- Der Parser liefert in der Regel einen abstrakten Syntaxbaum.

LICENSE