Teil 3: Syntax-Diagramme

Literatur:

- Kathleen Jensen/Niklaus Wirth: PASCAL — User Manual and Report, 4th Edition. Springer, 1991.
- Niklaus Wirth: Compilerbau (in German). Teubner, 1986.
- Oracle8 SQL Reference, Oracle Corporation, 1997, Part No. A58225-01. Appendix A is a short introduction to syntax diagrams.
- Don Chamberlin: A Complete Guide to DB2 Universal Database. Morgan Kaufmann, 1998. Section 1.1.2 is a very quick introduction to syntax diagrams.

Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- Syntax-Diagramme lesen
- prüfen, ob eine gegebene Zeichenfolge einem gegebenen Syntax-Diagramm entspricht.

Inhalt

1. Allgemeines über Syntax-Formalismen

2. Mehrstufige Syntax-Analyse

3. Syntax-Diagramme

Syntax-Formalismen (1)

- SQL-Anfragen sind Folgen von Zeichen:

  S E L E C T ␣ * ␣ F R O M ␣ E M P

- Nicht jede Zeichenfolge ist eine SQL-Anfrage:

  D a s ␣ i s t ␣ M u r k s !

- Eine Menge von Zeichenfolgen über einem Alphabet (Zeichenvorrat) nennt man eine formale Sprache. Z.B. die Menge der SQL-Anfragen.

  Der Zeichenvorrat für SQL muß mindestens folgende Zeichen enthalten: Die Buchstaben `a-zA-Z` (ohne Umlaute), Ziffern, das Leerzeichen, und die Zeichen `"%&'()*+,-./:;<=>?_|`. U.U. kann man nationale Zeichen (z.B. Umlaute) verwenden (system-/konfigurationsabhängig).

Syntax-Formalismen (2)

- In der Informatik gibt es viele solche Computer-Sprachen (z.B. Programmiersprachen wie C oder Dokument-Beschreibungssprachen wie HTML).
- Diese Sprachen müssen präzise definiert werden:
  - ◇ Die Syntax einer Sprache legt fest, was gültige Zeichenfolgen sind (was also in der formalen Sprache enthalten ist).
  - ◇ Die Semantik der Sprache legt fest, was diese Zeichenfolgen bedeuten.

Syntax-Formalismen (3)

- Für die Definition der Syntax gibt es etablierte Formalismen, z.B.
  - ◇ Reguläre Ausdrücke (nur für einfache Sprachen).
  - ◇ Kontextfreie Grammatiken (auch BNF).
  - ◇ Syntax-Diagramme.
- Syntax-Diagramme und kontextfreie Grammatiken sind gleich-mächtig, können also die gleichen Sprachen beschreiben.

  Tatsächlich kann man darin nicht alle Einschränkungen von SQL oder von Programmiersprachen wie C, Java beschreiben. Mit dem Formalismus definiert man eine Obermenge der gültigen SQL-Anfragen, und fügt dann einige Zusatz-Bedingungen hinzu (meist nicht-formal).

Syntax-Formalismen (4)

- Schon kleine Abweichungen von einer gültigen Eingabe führen dazu, daß das Programm (z.B. DBMS) die Eingabe nicht mehr versteht:

  SELECT * FRIM EMP

- Man sagt dann, daß die Eingabe syntaktisch falsch ist, oder, daß sie einen Syntaxfehler enthält.

  Da das DBMS die Eingabe nicht versteht, kann es sie natürlich nicht ausführen (Das ist ein Unterschied zwischen Menschen und Computern: Menschen verstehen viel eher, was gemeint war.). Man muß die Eingabe dann korrigieren und erneut ausführen lassen.

Syntax-Formalismen (5)

- Letztendlich stellt natürlich jedes Programm eine formale Definition seiner Eingabesprache dar.
- Der Programmcode ist aber wesentlich schwieriger zu verstehen, als etwa eine kontextfreie Grammatik.
- Üblicherweise wird die Grammatik zuerst definiert, und dann erst ein Programm geschrieben.

  Es gibt Verfahren, mit denen man Programmcode zur Syntaxanalyse systematisch aus einer kontextfreien Grammatik erzeugen kann.

- Nur mit einer unabhängigen Definition der Syntax kann man sagen, daß das Programm einen Fehler ("Bug") enthält.

Syntax-Formalismen (6)

- Um eine Sprache wie SQL zu beherrschen, sollte man eine formale Syntax-Definition lesen können.

  Dies ist z.B. bei Zweifelsfällen wichtig, außerdem hilft es aber auch, das Verständnis von der Sprache zu verbessern. Z.B. entsprechen die syntaktischen Kategorien, die in der Sprachdefinition vorkommen, oft nützlichen Konzepten. Natürlich sind auch Beispiele wichtig, um eine Sprache zu erlernen. Aber wenn man nur Beispiele gesehen hat, ist die Extrapolation auf den allgemeinen Fall schwierig.

- Das SQL-Referenzhandbuch von Oracle verwendet Syntax-Diagramme (ebenso wie viele Lehrbücher).

  Der SQL-Standard enthält eine kontextfreie Grammatik.

Syntax-Formalismen (7)

- Jedes DBMS hat einen eigenen SQL-Dialekt.

  (Fast) alle bieten mindestens SQL-89, plus eigene Erweiterungen.

- Eine Anfrage heißt portabel, wenn sie von verschiedenen DBMS ausgeführt werden kann.

  Das ist nur wichtig, wenn man die Anfrage nicht nur ein einziges Mal ausführen will (z.B. weil die Anfrage in einem Programm enthalten ist, was später eventuell auch ein anderes DBMS benutzen soll).

- Das kann man schlecht durch Ausprobieren herausfinden, dazu muß man den Standard lesen.

  Bei manchen DBMS kann man allerdings Warnungen für systemspezifische SQL-Erweiterungen anschalten.

Wichtiger Rat

- Man kann aus Fehlern viel lernen, aber nur, wenn man sie wirklich aufklärt.
- Wenn sich das DBMS nicht wie erwartet verhält (z.B. eine komische Fehlermeldung ausgibt), versuchen Sie den Grund zu verstehen.

  Übungsleiter und Professor werden Ihnen gerne helfen. Am besten kopieren Sie die kritische Anfrage in eine Datei, so daß das komische Verhalten reproduzierbar ist.

- Natürlich kann man auch etwas herumprobieren, aber Ziel muß dabei sein, das Problem zu verstehen.

  Geben Sie sich nicht zufrieden, wenn es irgendwann zufällig läuft.

Inhalt

1. Allgemeines über Syntax-Formalismen
2. Mehrstufige Syntax-Analyse
3. Syntax-Diagramme

Lexikalische Syntax (1)

- Die Syntax von Sprachen wie SQL (auch Java etc.) wird normalerweise in drei Schritten definiert:
  - ◇ Zuerst definiert man, wie lexikalische Symbole ("Token") aus einzelnen Zeichen aufgebaut sind.
  - ◇ Dann definiert man, wie Anfragen aus solchen lexikalischen Symbolen aufgebaut sind.
  - ◇ Schließlich gibt man weitere Einschränkungen an, die sich nicht mit einer kontextfreien Grammatik ausdrücken lassen, z.B. daß Datentypen passen oder Spaltennamen in der Tabelle existieren.

Lexikalische Syntax (2)

- Ursprüngliche Eingabe:

  S E L E C T ␣ * ␣ F R O M ␣ E M P

  W H E R E ␣ S A L ␣ > = ␣ 1 0 0 0

- Nach der lexikalischen Analyse (Zusammenfassung zu Wortsymbolen):

  SELECT | * | FROM | EMP

  WHERE | SAL | >= | 1000

- Der zweistufige Ansatz reduziert die Komplexität.

Lexikalische Syntax (3)

- In SQL sind Wortsymbole (Token) z.B.
  - ◇ Schlüsselworte mit einer besonderen Bedeutung, z.B. `SELECT`.
  - ◇ Bezeichner (etwa für Tabellen und Spalten), z.B. `EMP`.
  - ◇ Datentyp-Konstanten ("Literale"), z.B. `'DBMS'` or `123`.
  - ◇ Vergleichs- und Datentyp-Operatoren, z.B. `=`, `<=`, `+`, `||`.
  - ◇ Weitere Zeichen (zur Strukturierung), wie z.B. Klammern und Komma.

Lexikalische Syntax (4)

- Zwischen zwei Wortsymbolen (Token) dürfen in formatfreien Sprachen wie SQL beliebig viele Leerzeichen, Zeilenumbrüche und Tabulatoren stehen.

  Bei einer Leerzeile nimmt Oracle SQL*Plus allerdings an, daß die Eingabe zu Ende ist. Der Standard hat keine solchen Einschränkungen.

  Die Leerzeichen etc. werden in der ersten Stufe der Syntaxanalyse entfernt, spielen daher für die anschließende Verarbeitung keine Rolle.

- Man darf auch gar keinen Leerplatz zwischen zwei Wortsymbolen verwenden, wenn sie dadurch nicht zu einem verschmelzen, z.B.
  - ◇ `SAL>=1000` ist möglich, `SELECTENAME` dagegen nicht.

Inhalt

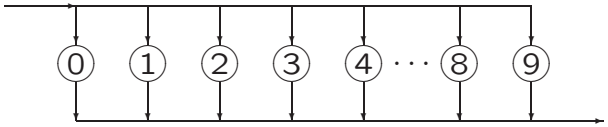
1. Allgemeines über Syntax-Formalismen

2. Mehrstufige Syntax-Analyse

3. Syntax-Diagramme

Syntax-Diagramme (1)

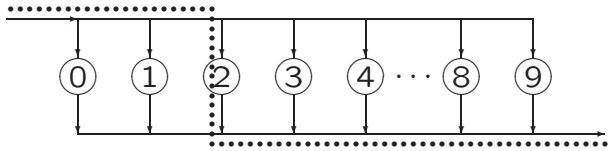
- Ziffer:



- Ein Syntax-Diagramm besteht aus:
  - ◇ Name, im Beispiel "Ziffer".
  - ◇ Start: Pfeil, der in das Diagramm hineingeht.
  - ◇ Ziel: Pfeil, der das Diagramm verläßt.
  - ◇ Ovale/Kreise und Rechtecke, die mit Pfeilen verbunden sind, und eine Beschriftung enthalten.

Syntax-Diagramme (2)

- Die durch dieses Diagramm definierte formale Sprache ist: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
- Um zu zeigen, daß eine Eingabe, z.B. "2", zu der formalen Sprache "Ziffer" gehört, die durch dieses Diagramm definiert ist, muß man einen Weg vom Start zum Ziel finden, der der Eingabe entspricht:



Syntax-Diagramme (3)

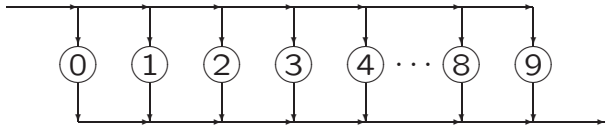
- Man kann Syntax-Diagramme auf zwei Arten verwenden:
  - ◇ Man kann syntaktisch korrekte Zeichenfolgen erzeugen, indem man den Pfeilen folgt und jedes Zeichen in einem Oval/Kreis ausgibt.
  - ◇ Man kann gegebene Zeichenfolgen analysieren, indem man versucht, einen Pfad zu finden, der der Zeichenfolge entspricht. Immer wenn man ein Oval/Kreis durchläuft, muß das betreffene Zeichen das nächste Zeichen der Eingabe sein.

Syntax-Diagramme (4)

- Jede Linie (Kante) hat nur eine mögliche Richtung.

  Sie ergibt sich aus der nächstgelegenen Pfeilspitze.

- Wenn man die Richtungen für alle Segmente explizit macht, würde das Diagramm so aussehen:



- Da das etwas kompliziert (überladen) aussieht, läßt man einige Pfeilspitzen weg, wenn die Richtung aus dem Kontext "offensichtlich" ist.

Syntax-Diagramme (5)

- Es gibt Verzweigungspunkte im Diagramm, wo man zwischen verschiedenen Pfaden wählen kann.

  Sonst würde das Diagramm nur eine Zeichenfolge beschrieben. Im Beispiel kann man etwa nach dem eingehenden Pfeil entweder nach unten zur Ziffer 0 gehen oder weiter nach rechts für eine andere Ziffer.

- Wenn man eine gegebene Eingabe auf Korrektheit prüft, hilft das nächste Eingabezeichen meist, die notwendige Richtung eindeutig festzulegen.

  Man versucht, Syntaxdiagramme so zu konstruieren, daß bei einer Verzweigung die Ovale/Kreise, die man in den beiden Richtungen erreichen kann, keine gleichen Symbole enthalten.

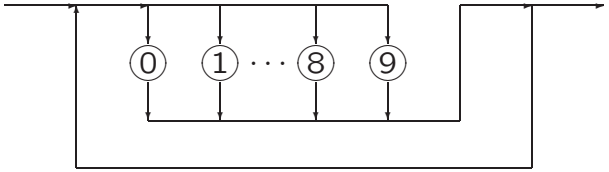
- Kann man keinen Pfad finden, ist die Eingabe falsch.

Syntax-Diagramme (6)

- Syntax-Diagramme können Zyklen enthalten.

  Z.B. kann man immer längere SQL-Anfragen aufbauen.

- Ziffernfolge:

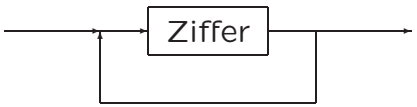


- Aufgabe: Finden Sie einen Pfad durch dieses Diagramm, der zeigt, daß "81" korrekt ist.

  Man darf die gleiche Linie mehrfach durchlaufen.

Syntax-Diagramme (7)

- Schon definierte Syntax-Diagramme kann man in neuen Diagrammen verwenden.
- Dazu zeichnet man ein Rechteck, in dem der Name des verwendeten Syntax-Diagramms steht:
- Ziffernfolge:



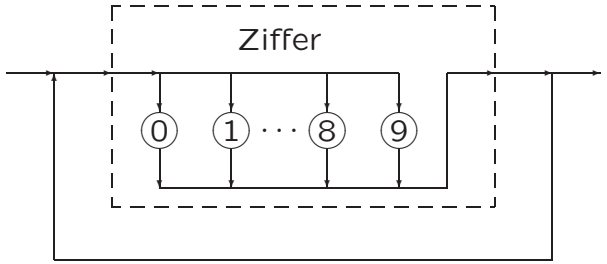
- Das Rechteck steht also für eine syntaktische Kategorie (wie Subjekt, Prädikat, Objekt).

Syntax Diagrams (8)

- Man kann das Rechteck durch das definierende Diagramm ersetzen.

  Ein Rechteck hat eine eingehende und eine ausgehende Kante (Pfeil), wie auch ein ganzes Diagramm. Daher passt das Diagramm hinein.

- Ziffernfolge:



Syntax-Diagramme (9)

- Man muß natürlich nicht explizit das Diagramm einfügen, sondern kann auch so vorgehen:
  - ◇ Wenn man ein Rechteck betritt, merkt man sich die Stelle in dem Diagramm (d.h. das Rechteck),
  - ◇ und geht nun zu dem Diagramm, dessen Name in dem Rechteck steht,
  - ◇ durchläuft dieses Diagramm vom Start zum Ziel,
  - ◇ und kehrt dann zum ursprünglichen Diagramm zurück und verläßt dort das Rechteck auf der anderen Seite.

Syntax-Diagramme (10)

- Natürlich lernt man nach einiger Zeit, was eine "Ziffer" ist, und braucht das Diagramm dann nicht mehr aufzuschlagen.
- Jedes Diagramm definiert eine formale Sprache (Menge von Zeichenfolgen).

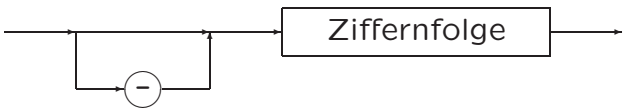
  Z.B. Ziffer: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

- Wenn man ein Rechteck durchläuft, das mit "X" beschriftet ist, muß man ein Zeichenfolge aus der formalen Sprache "X" einsetzen.

  Am Ende interessiert uns nur die formale Sprache "SQL-Anfrage". Die anderen Sprachen wie "Ziffer" sind nur Zwischenschritte.

Syntax-Diagramme (11)

- Man kann Ovale und Rechtecke beliebig in einem Diagramm mischen:
- Ganze Zahl:



- Das Diagramm "Ganze Zahl" beschreibt eine Sprache, die z.B. folgendes enthält: `123`, `-45`, `007`.
- Sie enthält dagegen nicht: `+89`, `--5`, `23-42`, `0.56`.

Syntax-Diagramme (12)

- Ein Syntax-Diagramm darf auch ein Rechteck mit dem eigenen Namen enthalten.

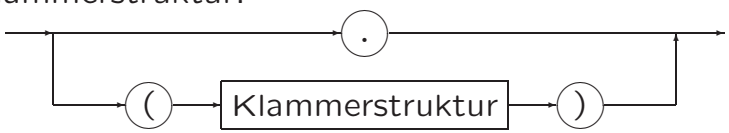
  Man sagt dann "es ist rekursiv". Zwei (oder mehr) Diagramme können sich auch gegenseitig benutzen.

- Man kann in diesem Fall natürlich nicht mehr vorab alle Rechtecke durch ihre Diagramme ersetzen (das würde ja nicht aufhören), aber man kann ein Rechteck immer dann, wenn man es tatsächlich durchläuft, ersetzen.

  Irgendwann muß man einen anderen Pfad wählen, um das Diagramm zu verlassen. Das gilt ja auch bei einem gewöhnlichen Zyklus.

Syntax-Diagramme (13)

- Klammerstruktur:



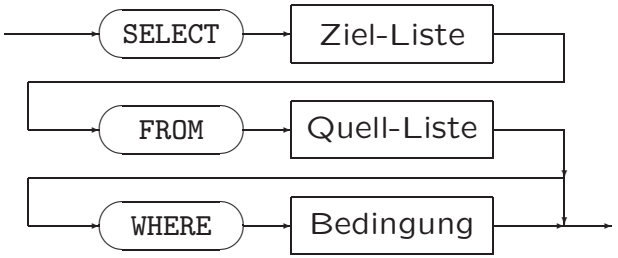
- Diese (sinnlose) Sprache enthält z.B.
  - ◇ `.`
  - ◇ `(.)`

    Wenn man dem unteren Pfad folgt und für "Klammerstruktur" den "." einsetzt, von dem man inzwischen weiß, daß er in der Sprache enthalten ist.

  - ◇ `((.))`, `(((.)))`, u.s.w.

Syntax-Diagramme (14)

- Wenn man die lexikalische Syntax definiert, erscheinen einzelne Buchstaben in den Ovalen/Kreisen.
- Später werden die Wortsymbole (z.B. `SELECT`) dann als Basis-Einheiten verwendet:



Syntax-Diagramme (15)

- Das Oracle SQL Handbuch benutzt gerade umgekehrt
  - ◇ Rechtecke für explizit so zu schreibene Zeichen (syntaktische Basiseinheiten)
  - ◇ Ovale für Aufrufe anderer Syntaxdiagramme (syntaktische Kategorien)
- Ich bin der ursprünglichen Notation gefolgt, die z.B. im Pascal Sprachreport verwendet wird.

  Niklaus Wirth war jedenfalls einer der ersten, die Syntaxdiagramme verwendet haben. Ich weiß nicht, ob er sie auch erfunden hat.

Syntax-Diagramme (16)

- In Buch von Chamerlin über DB2 wird eine etwas kompaktere Notation verwendet.

  Ovale werden für syntaktische Kategorien verwendet, die durch ein Syntax-Diagramm definiert sind, Großbuchstaben (ohne Kasten) für Schlüsselworte (explizit angegebene Worte/Zeichen), und Kleinbuchstaben für Wortsymbole wie "Spaltenname", die durch einen tatsächlichen Namen ersetzt werden können. Diagramme können sich über mehrere Zeilen erstrecken ohne einen expliziten Pfeil zurück (Start und Ziel sind mit expliziten Symbolen markiert. Wenn ein Pfeil einfach rechts aus dem Diagramm zeigt, setze man ihn in der nächsten Zeile fort.) diagrams can extend over multiple lines without explicit backward arrow Schließlich gibt es noch eine spezielle Notation für Optionen, die in beliebiger Reihenfolge angegeben werden können.

Aufgabe

- Entwickeln Sie Syntaxdiagramme für die Kommandosprache eines Textadventurespiels.
- Typische Kommandos bestehen aus Verb und Objekt, z.B. "`Nimm Lampe`".

  Verben z.B. "`nimm`", "`untersuche`". Objekte z.B. "`Lampe`", "`Seil`".

- Man kann optional einen Artikel benutzen (braucht nicht zu passen): "`Nimm die Lampe`".
- Man kann ein Verb mit mehreren Objekten verwenden: "`Nimm die Lampe und das Seil`".

  Wird intern behandelt wie "`Nimm Lampe`", "`Nimm Seil`".