Der Parser-Generator Accent

1 Vorgeschichte

Kurz vor 1960 kamen (mit der Sprache Algol60) die *kontextfreien Grammatiken* in die Informatik. Mitte der 1975-er Jahre wurden zahlreiche *Parser-Generatoren* entwickelt. Das sind Programme, die als Eingabe eine (kontextfreie) Grammatik erwarten und daraus einen (in einer Programmiersprache wie C oder Java oder ... formulierten) Parser erzeugen. Aus heute nicht mehr so leicht nachvollziehbaren Gründen (großer Stolz auf die entwickelten Werkzeuge?) war es üblich, diese Parser-Generatoren als "Compiler Compiler" zu bezeichnen, obwohl sie keine Compiler compilieren, sondern nur ("abstrakte") Grammatiken in ("konkrete") Parserprogramme übersetzen.

Der heute mit Abstand bekannteste und am weitesten verbreitete Parser-Generator ist der `yacc` (yet another compiler compiler, auf Deutsch etwa "noch so ein Compiler Compiler", eine scherzhafte Anspielung auf seine zahlreichen Vorgänger und Konkurrenten). Auch heute noch steht einem der `yacc` auf praktisch jedem Unix-System zur Verfügung, und eine Weiterentwicklung namens `bison` gibt es unter Unix und unter Windows.

Anmerkung: Das ähnlich wie `yacc` klingende englische Wort yak bezeichnet ein tibetanisches Hochlandrind (Bos grunniens). Vermutlich deshalb wurde die Weiterentwicklung des `yacc` nach einem anderen Rindvieh (Bison bison) benannt.

Als die erste Version des `yacc` entstand (in den 1975-er Jahren) waren verbreitete Computer erheblich langsamer als PCs aus den 2000-er Jahren (etwa um den Faktor eine Million). Deshalb wurde der `yacc` auf sogenannte *LR-Grammatiken* eingeschränkt. Dabei ist LR eine ziemlich komplizierte und (für viele Menschen) nicht ganz leicht zu erkennende oder nachzuprüfende Eigenschaft. Wenn eine Grammatik diese Eigenschaft hat, kann der `yacc` daraus einen besonders schnellen (tabellengesteuerten, LR-) Parser erzeugen. Man sagt auch:
Der `yacc` erzeugt aus einer LR-Grammatik einen LR-Parser.

Wendet man den `yacc` auf eine Grammatik an, die *nicht LR* ist, bekommt man Fehlermeldungen wie `"Shift reduce conflict ..."` oder `"Reduce reduce conflict ..."`. Auch geübte Compilerbauer fürchten diese Fehlermeldungen, weil es häufig unklar und schwierig ist zu erkennen, ob und wie man sie (durch einen "Umbau der Grammatik in eine LR-Grammatik") beseitigen kann.

Anmerkung: Bei der Definition der Sprache HTML unterlief den Entwicklern ein historisch bedeutsamer Fehler. Um das Schreiben von HTML-Seiten zu erleichtern, führten sie eine harmlos klingende Regel ein, die es erlaubt, bestimmte geöffnete Klammern ("Tags niedriger Stufe") implizit ("ohne Schreibarbeit") dadurch zu schließen, dass man eine andere Klammer ("einen Tag höherer Stufe") explizit schloss. Wegen dieser Regel ist die Grammatik für HTML nicht LR und man kann mit `yacc` keinen Parser daraus erzeugen. Statt von Hand Parser zu schreiben, die genau dem HTML-Standard entsprachen, begannen wichtige Browser-Hersteller einen Wettbewerb, wer den "am meisten akzeptierenden" (und damit am wenigsten standardkonformen) Parser und Browser schreiben kann. Einer verbreiteten Ansicht entsprechend hat Microsoft mit dem InternetExplorer diesen Wettbewerb gewonnen, aber dadurch den (auch vorher schon "kränkelnden") HTML-Standard geschwächt. XHTML ist eine Variante von HTML ohne die fragwürdige Klammer-Spar-Regel. Die Grammatik von XHTML ist LR.

Accent (**a c**ompiler **c**ompiler for the **ent**ire class of contextfree grammers) ist ein Parser-Generator, der für **jede** kontextfreie Grammatik einen Parser erzeugen kann (nicht nur für LR-Grammatiken oder andere Teilmengen). Eine Eingabe für den `accent` sieht genau so aus, wie eine Eingabe für den `yacc`. Auch der `accent` wird (um `yacc`-Fans nicht zum Lernen ungewohnter Bezeichnungen zu zwingen) als *Compiler Compiler* bezeichnet. Es folgt ein simples Beispiel für eine Eingabe, die der `accent` (oder der `yacc`) in einen Parser übersetzen kann.

2 Ein Parser für die Sprache der geraden Binärzahlen

Eine Grammatik für die Sprache der geraden Binärzahlen sieht in (yacc-) accent-Notation etwa so aus:

```
1   start :
2     gerade { printf("OK\n"); }
3   ;
4
5   irgend :
6      '0'
7   |  '1'
8   |  '0' irgend
9   |  '1' irgend
10  ;
11
12  gerade :
13     '0'
14  |  irgend '0'
15  ;
```

In dieser Grammatik sind `start`, `irgend` und `gerade` Zwischensymbole. `'0'` und `'1'` sind Endsymbole. Das zuerst ("zuoberst") definierte Zwischensymbol ist automatisch das Startsymbol (hier: `start`).

Alle Regeln mit dem selben Zwischensymbol auf der linken Seite müssen nacheinander notiert werden. Die linke Seite (das Zwischensymbol) schreibt man nur einmal hin, trennt die rechten Seiten der Regeln durch senkrechte Striche `|` und schließt das Ganze mit einem Semikolon `;` ab. Im Beispiel gibt es nur eine Regel mit dem Zwischensymbol `start` auf der linken Seite (Zeile 1 bis 3). Für das Zwischensymbol `irgend` gibt es vier Regeln (Zeile 5 bis 10) und für `gerade` zwei Regeln (Zeile 12 bis 15).

Nach jeder Regel darf man eine C-Blockanweisung `{...}` notieren. Sie wird ausgeführt, wenn die entsprechende Regel zum Ableiten der Eingabe des Parsers benützt wird. Das Beispiel enthält nur eine einzige solche Blockanweisung (in Zeile 2, am Ende der einzigen Regel für `start`).

Ein Parser liest seine Eingabe in aller Regel mit Hilfe eines sogenannten Lexers (oder: Scanners) ein. Für (`yacc` und) `accent` kann man einen solchen Lexer mit dem Lexer-Generator `lex` (oder mit der Varianten `flex`) erzeugen. Für die obige `Accent`-Spezifikation kann eine passende `flex`-Spezifikation etwa so aussehen:

```
16  %{
17  #include "yygrammar.h"
18  %}
19  %%
20  "0"       { incColNr(); return '0'; }
21  "1"       { incColNr(); return '1'; }
22  " "       { incColNr();  /* skip blank */ }
23  "--".*\n  { incLineNr(); /* skip comment and adjust linenumber */ }
24  \n        { incLineNr(); /* skip newline and adjust linenumber */ }
25  .         { incColNr(); yyerror("illegal character");}
```

Diese `lex`- (`flex`-) Spezifikation legt fest:

Die Eingabe sollte im Wesentlichen nur aus den Zeichenketten `"0"` und `"1"` bestehen (Zeile 20, 21). Außerdem sind Blanks `" "` erlaubt (Zeile 22), Kommentare (Zeile 23) und `newline`-Zeichen `\n` (Zeile 24). Blanks, Kommentare und `newline`-Zeichen werden vom Lexer einfach überlesen (in den C-Blockanweisung in den Zeilen 22, 23 bzw. 34 steht kein `return`-Befehl, der die Kontrolle vom Lexer zurück an den Parser geben würde). Kommentare müssen mit `"--"` beginnen und reichen bis zum nächsten `newline`-Zeichen `\n`. Alle anderen Zeichen werden vom Lexer als unerlaubt betrachtet (Zeile 25, der Punkt `.` bezeichnet ein beliebiges Zeichen).

Immer wenn der Parser ein "weiteres Stück Eingabe" benötigt, ruft er den Lexer auf. Wenn der Lexer in der Eingabe das Lexem `"0"` (bzw. `"1"`) erkennt, liefert er dem Parser das Token `'0'` (bzw. `'1'`), mit der `return`-Anweisungen in Zeile 20 (bzw. 21). Blanks und Kommentare überliest der Lexer, ohne dass der Parser etwas davon erfährt.

Die Cyan-Versionen von Gentle verwenden standardmäßig den `accent` als Parsergenerator, können aber im Prinzip auch den `yacc` benutzen.

Weitere Informationen über den `accent` findet man unter http://accent.compilertools.net/ .