Lexer und Parser

Ein Lexer (oder: Scanner) liest eine *Folge von Zeichen* ein, such darin nach Lexemen, erzeugt aus jedem Lexem ein Token und gibt eine *Folge von Token* aus.

Ein Parser liest eine *Folge von Token* ein und konstruiert daraus einen *Syntaxbaum*.

Ein Lexer führt *Typ-3-Syntaxprüfungen* durch. Er erkennt z.B.
- nicht-erlaubte Zeichen,
- falsch geschriebene Schlüsselworte,
- Literale, die nicht den Regeln für Literale entsprechen,
- Bezeichner, die nicht den Regeln für Bezeichner entsprechen etc.

Ein Parser führt *Typ-2-Syntaxprüfungen* durch. Er erkennt z.B.
- fehlende oder überzählige Klammern,
- Vereinbarungen, die nicht den Regeln für Vereinbarungen entsprechen,
- Ausdrücke, die nicht den Regeln für Ausdrücke entsprechen,
- Anweisungen, die nicht den Regeln für Anweisungen entsprechen etc.

Ein Lexer führt nur relativ einfache Prüfungen durch, ist dabei aber sehr schnell.

Ein Parser führt kompliziertere Prüfungen durch, die aufwendiger sein können.

Wenn der Gentle-Compiler ein Quellprogramm übersetzt, übergibt er alle `token`-Prädikate (und alle in `phrase`-Prädikaten vorkommenden String-Literale wie `"mother"` oder `"while"` oder `")"` etc.) einem *Lexer-Generator*, der daraus einen Lexer generiert. Die `phrase`-Prädikate übergibt der Gentle-Compiler einem *Parser-Generator*, der daraus einen Parser generiert. Die übrigen Teile des Quellprogramms (die Typ-Vereinbarungen, `proc`-Prädikate und `condition`-Prädikate etc.) übersetzt der Gentle-Compiler selbst.

Bestimmte Probleme kann man (zumindest "im Prinzip") wahlweise vom Lexer oder vom Parser erledigen lassen. Wenn man dabei nicht genau beachtet, dass der Lexer "primitiver arbeitet" als der Parser, können einem leicht Fehler unterlaufen, die man schwer versteht.

Die folgenden beiden Gentle-Programme sollen dieselbe (ganz simple) Sprache erkennen. Das eine Programm läßt wichtige Arbeiten vom *Parser* erledigen (es enthält nur `phrase`-Prädikate, keine `token`-Prädikate). Das andere Programm enthält `token`-Prädikate, und läßt damit wichtige Arbeiten vom *Lexer* erledigen.

Die beiden Programme sollen (anhand eines einfachen Beispiels) einen Einblick vermitteln, wie verschieden Lexer und Parser arbeiten.

Warnung: Die folgenden beiden Programme sind Beispiele dafür, "wie man es *nicht* machen sollte". Das eine enthält einen Fehler, den man kennen sollte (um ihn zu vermeiden oder schnell zu erkennen, wenn er einem doch mal unterläuft). Das andere benutzt den Parser für Arbeiten, für die der Lexer besser geeignet ist. Aus der Tatsache, dass die Version *ohne* `token`-Prädikaten hier besser funktioniert als die *mit* `token`-Prädikaten sollte man auf keinen Fall schließen, dass man "`token`-Prädikate meiden sollte". Man sollte sie allerdings richtig verwenden, und nicht so falsch wie im Programm `token05.g`.

Ein Parser mit "überlappenden" token-Prädikaten

```
 1 // ---------------------------------------------------------------------
 2 // Datei token05.g
 3 // ---------------------------------------------------------------------
 4 // Die AB-Sprache = {A, B} (wird vom token-Praedikat AB_Wort erkannt)
 5 // Die BC-Sprache = {B, C} (wird vom token-Praedikat BC_Wort erkannt)
 6 //
 7 // Die ABC-Sprache besteht aus allen Worten, die
 8 // aus einem AB-Wort gefolgt von einem BC-Wort oder
 9 // aus einem BC-Wort gefolgt von einem AB-Wort bestehen.
10 // Die ABC-Sprache = {AB, AC, BA, BB, BC, CA, CB}
11 // Der vorliegende Parser soll die ABC-Sprache erkennen.
12 // ---------------------------------------------------------------------
13 phrase MyStart
14    rule MyStart: AB_Wort BC_Wort
15    rule MyStart: BC_Wort AB_Wort
16
17 token AB_Wort   <<<A|B>>>
18 token BC_Wort   <<<B|C>>>
19
20 root
21    MyStart
22    "No parse error detected!\n"
23 // ---------------------------------------------------------------------
24 // Die folgende Ausgabe eines Testskripts zeigt, dass der vorliegende
25 // Parser nur 4 Worte der ABC-Sprache (AC, BC, CA, CB) akzeptiert
26 // und die restlichen 3 Worte (AB, BA, BB) ablehnt:
27 //
28 // Testfile1
29 // ----------------------------
30 // SRC AB
31 // Line 1: syntax error
32 // ----------------------------
33 // SRC AC
34 // No parse error detected!
35 // ----------------------------
36 // SRC BA
37 // Line 1: syntax error
38 // ----------------------------
39 // SRC BB
40 // Line 1: syntax error
41 // ----------------------------
42 // SRC BC
43 // No parse error detected!
44 // ----------------------------
45 // SRC CA
46 // No parse error detected!
47 // ----------------------------
48 // SRC CB
49 // No parse error detected!
50 // ----------------------------
51 // SRC AA
52 // Line 1: syntax error
53 // ----------------------------
54 // SRC CC
55 // Line 1: syntax error
56 // ----------------------------
57 // SRC XX
58 // Line 1: syntax error
59 // ---------------------------------------------------------------------
```

Ein Parser mit "überlappenden" phrase-Prädikaten

```
 1 // ------------------------------------------------------------------------
 2 // Datei phrase05.g
 3 // ------------------------------------------------------------------------
 4 // Die AB-Sprache = {A, B} (ableitbar aus AB_Wort)
 5 // Die BC-Sprache = {B, C} (ableitbar aus BC_Wort)
 6 //
 7 // Die ABC-Sprache besteht aus allen Worten, die
 8 // aus einem AB-Wort gefolgt von einem BC-Wort oder
 9 // aus einem BC-Wort gefolgt von einem AB-Wort bestehen.
10 // Die ABC-Sprache = {AB, AC, BA, BB, BC, CA, CB}
11 // Der vorliegende Parser soll die ABC-Sprache erkennen
12 // ------------------------------------------------------------------------
13 phrase MyStart
14    rule MyStart: AB_Wort BC_Wort
15    rule MyStart: BC_Wort AB_Wort
16
17 phrase AB_Wort
18    rule AB_Wort: "A"
19    rule AB_Wort: "B"
20
21 phrase BC_Wort
22    rule BC_Wort: "B"
23    rule BC_Wort: "C"
24
25 root
26    MyStart
27    "No parse error detected!\n"
28 // ------------------------------------------------------------------------
29 // Die folgende Ausgabe eines Testskripts zeigt, dass der vorliegende
30 // Parser alle Worte der ABC-Sprache akzeptiert (und 3 falsche Worte
31 // ablehnt):
32 //
33 // Testfile1
34 // ----------------------------
35 // SRC AB
36 // No parse error detected!
37 // ----------------------------
38 // SRC AC
39 // No parse error detected!
40 // ----------------------------
41 // SRC BA
42 // No parse error detected!
43 // ----------------------------
44 // SRC BB
45 // No parse error detected!
46 // ----------------------------
47 // SRC BC
48 // No parse error detected!
49 // ----------------------------
50 // SRC CA
51 // No parse error detected!
52 // ----------------------------
53 // SRC CB
54 // No parse error detected!
55 // ----------------------------
56 // SRC AA
57 // Line 1: syntax error
58 // ----------------------------
59 // SRC CC
60 // Line 1: syntax error
61 // ----------------------------
62 // SRC XX
63 // Line 1: syntax error
64 // ------------------------------------------------------------------------
```