4.9. Konstruktion der Steuertabellen

4.9.1. Benötigte Tabellen

1. Action-Tabelle
   action[state][token] enthält eine von 4 möglichen Aktionen:
   a) SHIFT newState
   b) REDUCE withRule
   c) ACCEPT
   d) ERROR
2. sizeofRHS-Tabelle
   sizeofRHS[rule] enthält die Anzahl der Grammatiksymbole auf der rechten Seite der Produktion "rule".
3. NontermLHS-Tabelle
   nontermLHS[rule] enthält die Nummer des Nichtterminalsymbols auf der linken Seite der Produktion "rule".
4. Goto-Tabelle
   goto[state][nonterminal] enthält den Folgezustand nach einer Reduktion. "state" ist der aufgedeckte Zustand nach der Stackreduktion, "nonterminal" die linke Seite der angewendeten Produktion.
5. Es ist zweckmäßig, eine "Default Reduction"-Tabelle anzulegen. Wenn in einer Zeile der Action-Tabelle außer Fehlereinträgen nur Reduktionen zu derselben Produktion stehen, kann diese Reduktion in jedem Fall durchgeführt werden! Dabei werden Fehler etwas später (aber zuverlässig) entdeckt. Beim Anwenden einer Default Reduction muß nicht einmal ein Token gelesen werden.

4.9.2. Tabellenkonstruktion

1. Action-Tabelle
   a) Die SHIFT-Einträge ergeben sich direkt aus dem Handle-Finding-Automaten.
   b) Die REDUCE-Einträge für einen Zustand ergeben sich aus denjenigen LR-Items, die den Zustand bilden und den Punkt ganz rechts haben. Beim Eintragen werden Shift/Reduce- und Reduce/Reduce-Fehler entdeckt, gemeldet und ggf. aufgelöst.
   c) Eine Reduktion mit Regel 0 bedeutet ACCEPT.
   d) Alle anderen Einträge sind ERROR.
2. SizeofRHS-Tabelle – trivial
3. NontermLHS-Tabelle – trivial
4. Goto-Tabelle
   Deren Einträge werden (wie die SHIFT-Einträge der Action-Tabelle) direkt dem Automaten entnommen. Konflikte kann es dabei nicht geben.
5. Default-Reduction-Tabelle
   Default Reductions kann man in der vorher ausgefüllten Action-Tabelle durch Vergleich der Einträge in einer Zeile leicht entdecken.

4.10. Treiber

4.10.1. Aufgaben

- Aufruf des Scanners, wenn ein Token benötigt wird
- Interpretation der Steuertabellen
  (d.h. Ausführen des "Handle Finding Automaton")
- Verwalten des Parse-Stacks
- Verwalten des dazu parallelen Semantik-Stacks
- Ausführen des User-Codes bei Reduktionen
- Erkennen von Fehler- und Ende-Zustand

4.10.2. Algorithmus

```
initParser;
while (1) {
    if (no default reduction for this state) {
        if (! token_read) {
            readToken;
            token_read = true;
        }
        action = actionTable [state] [token];
    } else {
        action = defaultReduction [state];
    }
    switch (action) {
        case SHIFT newState:
            update parse and semantic stack;
            state = newState;
```

```
                token_read = false;
                break;
            case REDUCE withRule:
                update parse stack;
                state = gotoTable [state][lhs(withRule)];
                execute user code;
                update semantic stack;
                break;
            case ACCEPT:
                return 0;
            case ERROR:
                return 1;
        }
    }
```

Bem.: Der Parser-Zustand "state" kann entweder im obersten Stackeintrag oder (besser) in einer lokalen Variablen gehalten werden. Dementsprechend ist "update parse stack" zu implementieren.