Prüfung Compiler Construction

Datum : 21.1.2011, 8:15 Uhr
Bearbeitungszeit : 90 Minuten
Hilfsmittel : Alle eigenen auf Papier, keine elektronischen Hilfsmittel
Erreichbare Punkte : 31

Name :
Vorname :
Studiengruppe :
Matrikelnummer :
Unterschrift :
Aufgabe 1: (9 Punkte)

Given the following Transition Diagram:

a) This Transition Diagram defines a language. Give three examples of strings that belong to this language and three examples of strings that do not belong to this language.

b) Construct a deterministic Transition Diagram defining the same language. Make it as simple as possible.

c) Give a Regular Expression defining the same language.

Aufgabe 2: (10 Punkte)

Consider expressions from a programming language (e.g. C) consisting only of Numbers, Variables, Assignments “=” and array access using brackets “[“ “]”. Examples are: “x = 1”, “x[y[z]] = 1”, “x[1] = y = 50”, “b[x = y[a]] = z”, “x = y[a = y]”, or “a[x = 1][y = 2] = 3”.

The expressions are described by the following rules:

- “=” is a binary right-associative operator.
- All expressions have a value (even assignments) but not all expressions are memory references.
- Variables are memory references but numbers are not.
- An expression that is a memory reference can be indexed by using brackets and an index value. The result is again a memory reference.
- An expression on the left hand side of an assignment must be a memory reference.

Invalid are the following strings: “x[= y]” (= is binary), “5 = x” (no assignment to a number), “x[]= z” (Index missing), “[x] = z” (Array missing), “x = 5[a]” (Indexing of a number), “a[=b = [x = y]]” (extra closing bracket).
Write a lex/yacc program to parse expressions as described above. Hint: Only parsing is required, not code generation.

a) Write a lex program, extracting suitable tokens from the input, ignoring valid but insignificant characters, and producing error messages for invalid characters.

b) Write a well structured yacc program, using the above lex program, to parse expressions as described above.

**Aufgabe 3:** (12 Punkte)

Given the following yacc program:

```
%token ID OPEN CLOSE NUMBER

%%
expr: atomic;
expr: cast expr;

atomic: ID;
atomic: NUMBER;
atomic: OPEN expr CLOSE;

cast: OPEN ID CLOSE;
```

a) Construct the start state of the SLR(1) automaton.

b) Construct the state that is reached from the start state after shifting first the token OPEN and then the token ID. Hint: This state will have conflicts.

c) Describe the conflicts in this state precisely.

d) How will a yacc generated SLR(1) parser resolve this conflict?

e) Which of the following valid strings will produce a parse error?

```
“OPEN ID CLOSE NUMBER -f”,
“OPEN NUMBER CLOSE -f”,
“OPEN ID CLOSE -f”,
“OPEN ID CLOSE OPEN ID CLOSE -f”.
```