SPL LL(1) Grammar Specification

Students' Programming Language - Refined Context-Free Grammar

This document presents the complete LL(1) compatible grammar for the Students' Programming Language (SPL), properly left-factored to eliminate parsing conflicts.

Complete Grammar Rules

Program Structure

```
SPL_PROG → 'glob' '{' VARIABLES '}'
           'proc' '{' PROCDEFS '}'
           'func' '{' FUNCDEFS '}'
           'main' '{' MAINPROG '}'
```

Variable Declarations

```
VARIABLES → VAR VARIABLES | ε
VAR       → id
NAME      → id
```

Procedure Definitions

```
PROCDEFS  → PDEF PROCDEFS | ε
PDEF      → id '(' PARAM ')' '{' BODY '}'
```

Function Definitions

```
FUNCDEFS  → FDEF FUNCDEFS | ε
FDEF      → id '(' PARAM ')' '{' BODY ';' 'return' ATOM '}'
```

Body and Parameters

```
BODY      → 'local' '{' MAX3 '}' ALGO
PARAM     → MAX3
```

Parameter Lists (≤3 identifiers, LL(1)-factored)

```
MAX3      → ε | id MAX3_2
MAX3_2    → ε | id MAX3_1
MAX3_1    → ε | id
```

Main Program

```
MAINPROG  → 'var' '{' VARIABLES '}' ALGO
```

Atoms

```
ATOM      → id | number
```

Algorithms (one-or-more instructions separated by semicolons)

```
ALGO      → INSTR ALGO'
ALGO'     → ';' INSTR ALGO' | ε
```

Instructions

```
INSTR     → 'halt'
          | 'print' OUTPUT
          | id INSTR_AFTER_ID
          | LOOP
          | BRANCH
```

Instruction Disambiguation (after identifier)

```
INSTR_AFTER_ID → '(' INPUT ')'          # procedure call
               → '=' ASSIGN_RHS         # assignment
```

Assignment Right-Hand Side

```
ASSIGN_RHS     → id ASSIGN_RHS_ID'
               → number
               → PARENS_TERM

ASSIGN_RHS_ID' → '(' INPUT ')' | ε     # function call or plain id
```

Parenthesized Terms

```
PARENS_TERM    → '(' UNOP TERM ')'
               → '(' TERM BINOP TERM ')'
```

Terms

```
TERM      → ATOM | PARENS_TERM
```

Operators

```
UNOP      → 'neg' | 'not'
BINOP     → 'eq' | '>' | 'or' | 'and' | 'plus' | 'minus' | 'mult' | 'div'
```

Output

```
OUTPUT    → ATOM | string
```

Input Arguments (0..3 actual arguments, LL(1)-factored)

```
INPUT     → ε | ATOM INPUT1
INPUT1    → ε | ATOM INPUT2
INPUT2    → ε | ATOM
```

Control Structures

```
LOOP      → 'while' TERM '{' ALGO '}'
          | 'do' '{' ALGO '}' 'until' TERM

BRANCH    → 'if' TERM '{' ALGO '}' BRANCH'
BRANCH'   → 'else' '{' ALGO '}' | ε
```

Terminal Symbols (Lexical Rules)

Keywords

- `glob`, `proc`, `func`, `main`, `var`, `local`, `return`
- `halt`, `print`, `while`, `do`, `until`, `if`, `else`
- `neg`, `not`, `eq`, `or`, `and`, `plus`, `minus`, `mult`, `div`

Operators and Punctuation

- `=`, `>`, `(`, `)`, `{`, `}`, `;`

Lexical Patterns

- **`id`**: User-defined identifier `[a-z][a-z0-9]*` (cannot be keywords)
- **`number`**: `0` or `[1-9][0-9]*` (no leading zeros)
- **`string`**: `"[a-zA-Z0-9]{0,15}"` (alphanumeric, max 15 characters)

LL(1) Properties

Key LL(1) Features

1. **Single Token Lookahead**: All parsing decisions made with exactly one token of lookahead
2. **No Left Recursion**: All recursive productions use right recursion
3. **Left Factoring**: Common prefixes properly eliminated
4. **Predictive Parsing**: Each production has disjoint FIRST sets

Critical Disambiguation Points

1. Procedure Call vs Assignment

After seeing an identifier in instruction context:

- If next token is `(` → procedure call
- If next token is `=` → assignment

2. Function Call vs Plain Identifier in Assignment

After `id =` in assignment:

- If next token after second identifier is `(` → function call

- Otherwise → plain identifier

3. Unary vs Binary Operations in Parentheses

After `(` in parenthesized term:

- If next token is unary operator → unary operation
- Otherwise → binary operation (first operand)

FIRST Sets (Key Examples)

- FIRST(VARIABLES) = {ε, id}
- FIRST(INSTR) = {halt, print, id, while, do, if}
- FIRST(ASSIGN_RHS) = {id, number, (}
- FIRST(TERM) = {id, number, (}

FOLLOW Sets (Key Examples)

- FOLLOW(VARIABLES) = {}, proc, local, while, do, if, halt, print, id}
- FOLLOW(ALGO) = {}, until, else}
- FOLLOW(TERM) = {), }, ;, until, else, and, or, plus, minus, mult, div, eq, >}

Language Constraints

Parameter Limits

- Maximum 3 parameters per procedure/function
- Maximum 3 arguments per procedure/function call

String Limits

- String literals maximum 15 characters (excluding quotes)
- Only alphanumeric characters allowed in strings

Identifier Rules

- Must start with lowercase letter
- Can contain lowercase letters and digits
- Cannot be identical to any keyword

Number Format

- Either single digit `0` or starts with non-zero digit
- No leading zeros allowed (except for `0` itself)

ALL

```
SPL_PROG → 'glob' '{' VARIABLES '}'
           'proc' '{' PROCDEFS '}'
           'func' '{' FUNCDEFS '}'
           'main' '{' MAINPROG '}'

VARIABLES → VAR VARIABLES | ε
VAR       → id
NAME      → id

PROCDEFS  → PDEF PROCDEFS | ε
PDEF      → id '(' PARAM ')' '{' BODY '}'

FUNCDEFS  → FDEF FUNCDEFS | ε
FDEF      → id '(' PARAM ')' '{' BODY ';' 'return' ATOM '}'

BODY      → 'local' '{' MAX3 '}' ALGO
PARAM     → MAX3

# up to 3 identifiers, LL(1)-factored
MAX3      → ε | id MAX3_2
MAX3_2    → ε | id MAX3_1
MAX3_1    → ε | id

MAINPROG  → 'var' '{' VARIABLES '}' ALGO

ATOM      → id | number

# one-or-more INSTR separated by semicolons
ALGO      → INSTR ALGO'
ALGO'     → ';' INSTR ALGO' | ε

INSTR     → 'halt'
          | 'print' OUTPUT
          | id INSTR_AFTER_ID
          | LOOP
          | BRANCH

# after an identifier, decide call vs assignment by one token of lookahead
```

```
INSTR_AFTER_ID → '(' INPUT ')'         # procedure call
               → '=' ASSIGN_RHS        # assignment

# RHS of assignment (function call allowed only here, per sheet)
ASSIGN_RHS     → id ASSIGN_RHS_ID'
               → number
               → PARENS_TERM
ASSIGN_RHS_ID' → '(' INPUT ')' | ε     # function call or plain id

PARENS_TERM    → '(' UNOP TERM ')'
               → '(' TERM BINOP TERM ')'

TERM      → ATOM | PARENS_TERM

UNOP      → 'neg' | 'not'
BINOP     → 'eq' | '>' | 'or' | 'and' | 'plus' | 'minus' | 'mult' | 'div'

OUTPUT    → ATOM | string

# 0..3 actual arguments, LL(1)-factored
INPUT     → ε | ATOM INPUT1
INPUT1    → ε | ATOM INPUT2
INPUT2    → ε | ATOM

LOOP      → 'while' TERM '{' ALGO '}'
          | 'do' '{' ALGO '}' 'until' TERM

BRANCH    → 'if' TERM '{' ALGO '}' BRANCH'
BRANCH'   → 'else' '{' ALGO '}' | ε
```