PL/0 Compiler

HuJinghui

November 3, 2013

1 Grammar for extended PL/0

program ::= *block* .

block ::= [*constdec*][*vardec*]{[*procdec*]|[*fundec*]} *compstmt*

constdec ::= **const** *constdef* {, *constdef* };

constdef ::= *ident* = *const*

const ::= [+|−] *unsign* | *character*

character ::= ′ *letter* ′|′ *digit* ′

string ::= “{*ASCII characters with decimal code number varys from* 32 *to* 126 *exclude* 34}”

unsign ::= *digit* { *digit* }

ident ::= *letter* { *letter* | *digit* }

vardec ::= **var** *vardef* ; { *vardef* ; }

vardef ::= *ident* {, *ident* } : *type*

type ::= *basictype* |**array**′[′ *unsign* ′]′**of** *basictype*

basictype ::= **integer**|**char**

procdec ::= *prochead* *block* {; *prochead* *block* };

fundec ::= *funhead* *block* {; *funhead* *block* };

prochead ::= **procedure** *ident* ′(′[*paralist*]′)′;

funhead ::= **function** *ident* ′(′[*paralist*]′)′ : *basictype* ;

paralist ::= [**var**] *ident* {, *ident* } : *basictype* {; *paralist* }

statement ::= *assignstmt* | *ifstmt* | *repeatstmt* | *callstmt*
| *compstmt* | *readstmt* | *writestmt* | *forstmt* | *nullstmt*

assignstmt ::= *ident* := *expression* | *funident* := *expression*
| *ident* ′[′ *expression* ′]′ := *expression*

funident ::= *ident*

expression ::= [+|−] *term* { *addop* *term* }

term ::= *factor* { *multop* *factor* }

factor ::= *ident* | *ident* ′[′ *expression* ′]′| *unsign* |′(′ *expression* ′)′| *callstmt*

callstmt ::= *ident* ′(′ *arglist* ′)′

arglist ::= *argument* {, *argument* }

argument ::= *expression*

addop ::= +|−

multop ::= $*|/$

condition ::= *expression* *relop* *expression*

relop ::= < | <= | > | >= | = | <>

ifstmt ::= **if** *condition* **then** *statement*
|**if** *condition* **then** *statement* **else** *statement*

repeatstmt ::= **repeat** *statement* **until** *condition*

forstmt ::= **for** *ident* := *expression* (**to**|**downto**) *expression* **do** *statement*

callstmt ::= *ident* ′(′[*arglist*]′)′

compstmt ::= **begin** *statement* {; *statement* }**end**

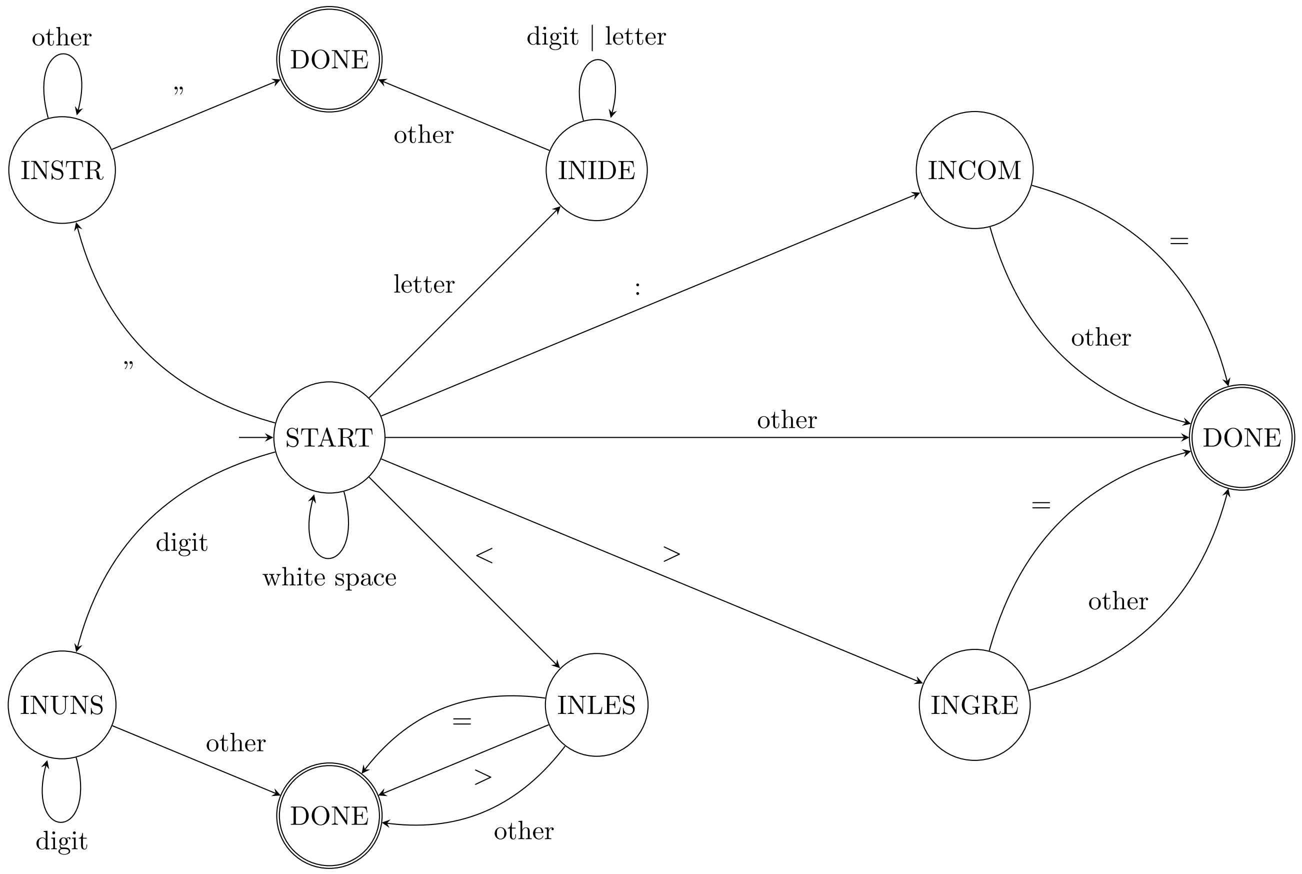
readstmt ::= **read**′(′ *ident* {, *ident* }′)′

writestmt ::= **write**′(′ *string* , *expression* ′)′|**write**′(′ *string* ′)′|**write**′(′ *expression* ′)′

letter ::= $a|b|c|...|z|A|B|C|...|Z$

digit ::= 0|1|2|3|...|9

2 State Machine for getToken


other
DONE
digit | letter
”
other
INSTR
INIDE
INCOM
=
letter
:
other
”
other
START
DONE
=
digit
<
>
white space
other
INUNS
INLES
INGRE
=
other
>
DONE
digit
other