# CS536: Homework 4

Keith Funkhouser

October 6th, 2015

For this homework you will define a syntax-directed translation for the CFG given below, which defines a very simple programming language.

```
program → MAIN LPAREN RPAREN LCURLY list RCURLY

list → list oneItem
     | epsilon

oneItem → decl
        | stmt

decl → BOOL ID SEMICOLON
     | INT ID SEMICOLON

stmt → ID ASSIGN exp SEMICOLON
     | IF LPAREN exp RPAREN stmt
     | LCURLY list RCURLY

exp → exp PLUS exp
    | exp LESS exp
    | exp EQUALS exp
    | ID
    | BOOLLITERAL
    | INTLITERAL
```

## 1

Write a syntax-directed translation for the CFG given above, so that the translation of an input program is the *set* of variables used somewhere in the program. Note: here the term *use* is in contrast to *declaration*; any appearance of a variable that is not a variable declaration counts as a *use* of that variable. For the example code in Question 2, the translation should be { `x, a, b` }.
Your translation rules should use the following notation:

- { } is an empty set
- { `ID.value` } is a set containing the variable whose name is the value associated with this `ID` token
- `S1` ∩ `S2` is the intersection of sets
- `S1` ∪ `S2` is the union of sets `S1` and `S2`
- `S1 - S2` is the set of all items that are in `S1` but not in `S2`

Note that you should not try to use something like "{ `a, b` }" to mean a set with two elements; instead, use set union to combine two sets that each contain one element.
Use the notation that was used in class and in the on-line readings; i.e., use `nonterminal.trans` to mean the translation of a nonterminal, and `terminal.value` to mean the value of a terminal. Assume that `ID.value` is a `String` (the name of the identifier). Use subscripts for translation rules that include the same nonterminal or the same terminal more than once.

In order of the CFG production rules given above, the following translation rules apply:

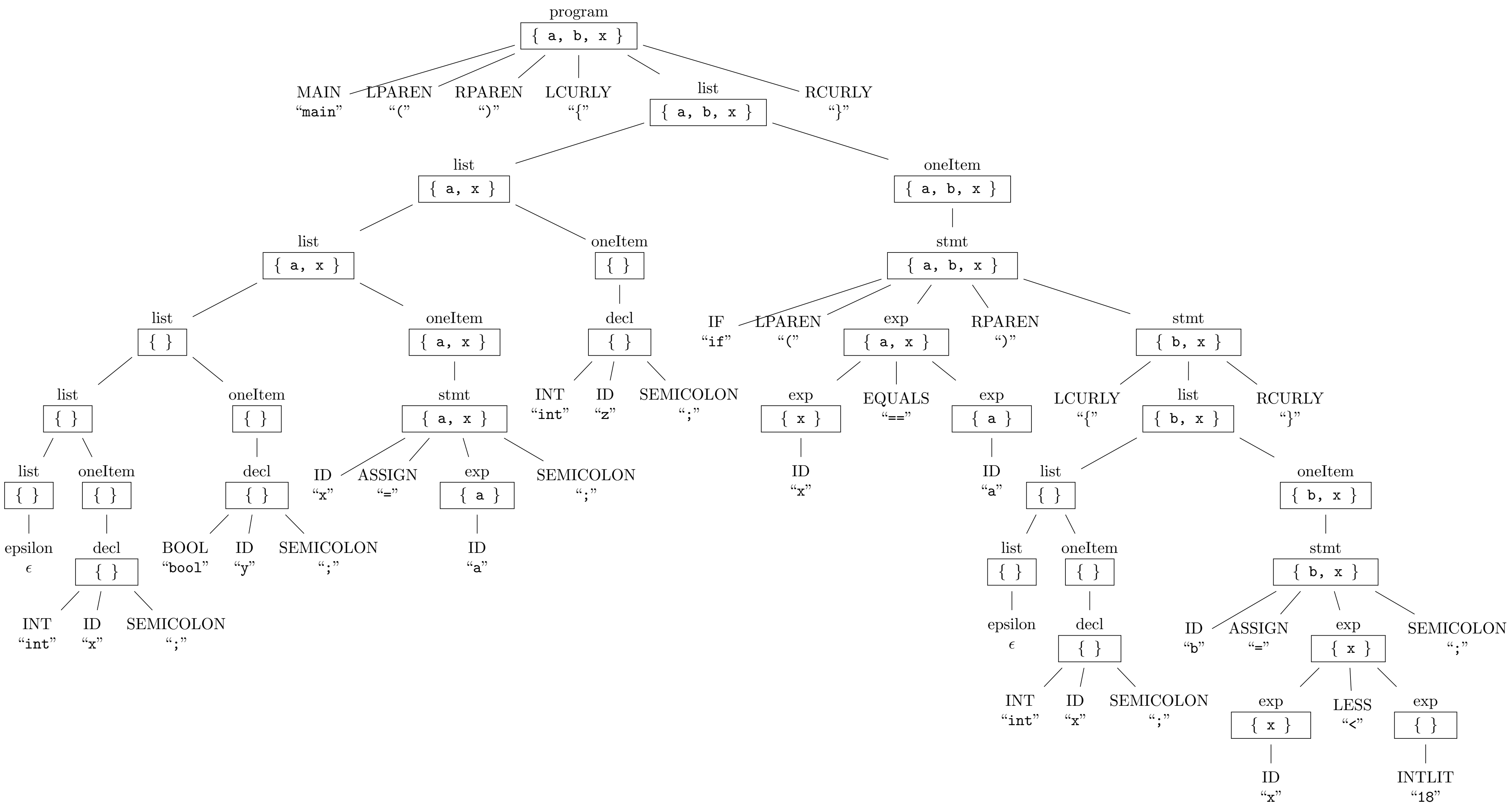
$$\begin{aligned}
\texttt{program.trans} &= \texttt{list.trans} \\
\texttt{list}_1\texttt{.trans} &= \texttt{list}_2\texttt{.trans} \cup \texttt{oneItem.trans} \\
\texttt{list.trans} &= \{\ \} \\
\texttt{oneItem.trans} &= \{\ \} \\
\texttt{oneItem.trans} &= \texttt{stmt.trans} \\
\texttt{decl.trans} &= \{\ \} \\
\texttt{decl.trans} &= \{\ \} \\
\texttt{stmt.trans} &= \{\ \texttt{ID.value}\ \} \cup \texttt{exp.trans} \\
\texttt{stmt}_1\texttt{.trans} &= \texttt{exp.trans} \cup \texttt{stmt}_2\texttt{.trans} \\
\texttt{stmt.trans} &= \texttt{list.trans} \\
\texttt{exp}_1\texttt{.trans} &= \texttt{exp}_2\texttt{.trans} \cup \texttt{exp}_3\texttt{.trans} \\
\texttt{exp}_1\texttt{.trans} &= \texttt{exp}_2\texttt{.trans} \cup \texttt{exp}_3\texttt{.trans} \\
\texttt{exp}_1\texttt{.trans} &= \texttt{exp}_2\texttt{.trans} \cup \texttt{exp}_3\texttt{.trans} \\
\texttt{exp.trans} &= \{\ \texttt{ID.value}\ \} \\
\texttt{exp.trans} &= \{\ \} \\
\texttt{exp.trans} &= \{\ \}
\end{aligned}$$

## 2

Draw a parse tree for the program given below and annotate each nonterminal in the tree with its translation.

```
main ( ) {
    int x;
    bool y;
    x = a;
    int z;
    if (x == a) {
        int x;
        b = x < 18;
    }
}
```

Consider the parse tree on the following page, where at each node the translation has been done for all of the nodes below it, according to the translation rules given above.



program
{ a, b, x }
MAIN "main"
LPAREN "("
RPAREN ")"
LCURLY "{"
list { a, b, x }
RCURLY "}"
list { a, x }
oneItem { a, b, x }
list { a, x }
oneItem { }
stmt { a, b, x }
list { }
oneItem { a, x }
decl { }
IF "if"
LPAREN "("
exp { a, x }
RPAREN ")"
stmt { b, x }
list { }
oneItem { }
stmt { a, x }
INT "int"
ID "z"
SEMICOLON ";"
exp { x }
EQUALS "=="
exp { a }
LCURLY "{"
list { b, x }
RCURLY "}"
list { }
oneItem { }
decl { }
ID "x"
ASSIGN "="
exp { a }
SEMICOLON ";"
ID "x"
ID "a"
list { }
oneItem { b, x }
epsilon ϵ
decl { }
BOOL "bool"
ID "y"
SEMICOLON ";"
ID "a"
list { }
oneItem { }
stmt { b, x }
INT "int"
ID "x"
SEMICOLON ";"
epsilon ϵ
decl { }
ID "b"
ASSIGN "="
exp { x }
SEMICOLON ";"
INT "int"
ID "x"
SEMICOLON ";"
exp { x }
LESS "<"
exp { }
ID "x"
INTLIT "18"