BU CS320 Assignment 6: Context Free Grammars

November 6, 2023

1. Given the following grammar where $\langle expr \rangle$ is the starting symbol:

$$
\begin{array}{lll}
\langle id \rangle & ::= & \text{a} \mid \text{b} \mid \text{c} \mid \ldots \mid \text{z} \\
\langle dig \rangle & ::= & 0 \mid 1 \mid 2 \mid \ldots \mid 9 \\
\langle expr \rangle & ::= & () \mid \langle dig \rangle \mid \langle id \rangle \\
 & \mid & \text{let } \langle id \rangle = \langle expr \rangle \text{ in } \langle expr \rangle \\
 & \mid & \langle expr \rangle \ ; \ \langle expr \rangle \\
 & \mid & \text{begin } \langle expr \rangle \text{ end}
\end{array}
$$

Demonstrate the grammar above is ambiguous.

consider let a = b in c ; d

① <expr>
=> let <id> = <expr> in <expr>
=> let a = <id> in <expr>
=> let a = b in <expr>
=> let a = b in <expr> ; <expr>
=> let a = b in <id> ; <expr>
=> let a = b in c ; <expr>
=> let a = b in c ; <id>
=> let a = b in c ; d

② <expr>
=> <expr> ; <expr>
=> let <id> = <expr> in <expr> ; <expr>
=> let a = <expr> in <expr> ; <expr>
=> let a = <id> in <expr> ; <expr>
=> let a = b in <expr> ; <expr>
=> let a = b in <id> ; <expr>
=> let a = b in c ; <expr>
=> let a = b in c ; <id>
=> let a = b in c ; d

It's ambiguous because we can get the string "let a = b in c ; d"
by expanding <expr> in 2 different structure.

2. Modify the grammar (reproduced below) to be unambiguous. Hint: There is not just one way.

⟨*id*⟩	::=	a | b | c | ... | z
⟨*dig*⟩	::=	0 | 1 | 2 | ... | 9
⟨*expr*⟩	::=	() | ⟨*dig*⟩ | ⟨*id*⟩
	|	let ⟨*id*⟩ = ⟨*expr*⟩ in ⟨*expr*⟩
	|	⟨*expr*⟩ ; ⟨*expr*⟩
	|	begin ⟨*expr*⟩ end

<expr> ::= <simple_expr> | <compound_expr>

<simple_expr> ::= () | <dig> | <id>

<compound_expr> ::= <let_expr> | <block_expr> | <sequence_expr>

<let_expr> ::= let <id> = <expr> in <expr>

<block_expr> ::= begin <expr> end

<sequence_expr> ::= <expr> ; <sequence_tail>

<sequence_tail> ::= <expr> | <expr> ; <sequence_tail>

3. Demonstrate your modified grammar fixes the previously shown ambiguity.

let a = b in c ; d

<expr>

=> <compound_expr>

=> <let_expr>

=> let <id> = <expr> in <expr>

=> let a = <expr> in <expr>

=> let a = <simple_expr> in <expr>

=> let a = <id> in <expr>

=> let a = b in <expr>

=> let a = b in <compound_expr>

=> let a = b in <sequence_expr>

=> let a = b in <expr> ; <sequence_tail>

=> let a = b in <simple_expr> ; <sequence_tail>

=> let a = b in <id> ; <sequence_tail>

=> let a = b in c ; <sequence_tail>

=> let a = b in c ; <expr>

=> let a = b in c ; <simple_expr>

=> let a = b in c ; <id>

=> let a = b in c ; d