EDAN65: Compilers
Computer Science
Lund University

September 08, 2014

E04: Context-free grammars

E04-1: Suppose there is a nonterminal *stmt* for statements. Construct an EBNF grammar for a sequence of

(a) zero or more statements, with a semicolon after each statement.

(b) one or more statements, with a semicolon between statements.

(c) zero or more statements, with a semicolon between statements.

E04-2: Translate the grammars in the previous problem to canonical form.

E04-3: The following grammar generates a language on the alphabet `{ "(", ")" }`.

```
S → "(" S ")"
S → S S
S → "(" ")"
```

(a) Which strings with length 6 belong to the language?

(b) The grammar is ambiguous. Which is the shortest string in the language with at least two parse trees?

E04-4: The following grammar is ambiguous. Construct an unambiguous grammar accepting the same language.

```
S → "(" S ")"
S → S S
S → ε
```

E04-5: The following grammar for logical expressions is ambiguous.

```
E → "!" E
E → E "&&" E
E → E "||" E
```

$\texttt{E} \rightarrow \texttt{ID}$

Assume that `!` has higher precedence than `&&` which in turn precedes over `||`. Construct an unambiguous EBNF grammar that respects the precedences describing the same language.

E04-6: Construct a canonical grammar that is equivalent to the following EBNF rule.

$\texttt{CallStmt} \rightarrow \texttt{ID "(" (}\ \epsilon\ \texttt{| Expr ("," Expr)*) ")"}$

E04-7: Every language that is described by a regular expression can be described by a *right regular grammar*, where all productions have one of the forms

$\texttt{A} \rightarrow \texttt{a B}$
$\texttt{A} \rightarrow \texttt{a}$
$\texttt{A} \rightarrow \epsilon$

where `A` and `B` are non-terminals and `a` is a terminal. Note that right recursion is allowed, but not left recursion.[1]

Construct a right regular grammar for the language described by the regular expression

`(a* b) | (b a*)`

Is it possible to do this without using productions on the form $\texttt{A} \rightarrow \texttt{a}$?

E04-8: Suppose you would like to write a parser that can parse basic regular expressions over the alphabet `{"a", "b"}`. Some short strings in this language are: `"a"`, `"b"`, `"a*"`, `"ab"`, `"(abb)*"`, `"(a|b)"`.

(a) What is the alphabet of this regular expression language?

(b) Construct an EBNF grammar for this language, and that respects the normal precedences of the regular expression operators.

[1] Equivalently, each regular expression can be described by a *left regular grammar* which only allows left recursion.