CS 320 : Formal Grammars

Marco Gaboardi
CDS 1019
gaboardi@bu.edu

Announcement

Next Tuesday we will have quiz 1 on formal grammars.

Generator vs Recognizer

```
<program> ::= <stmts>
<stmts> ::= <stmt> | <stmt> ; <stmts>
<stmt> ::= <var> = <expr>
<var> ::= a | b | c | d
<expr> ::= <term> + <term> | <term> - <term>
<term> ::= <var> | const
```

Recognize a sentence

```
a = b + const
<var> = b + const
<var> = <var> + const
<var> = <term> + const
<var> = <term> + <term>
<var> = <expr>
<stmt>
<stmts> =:: <program>
```

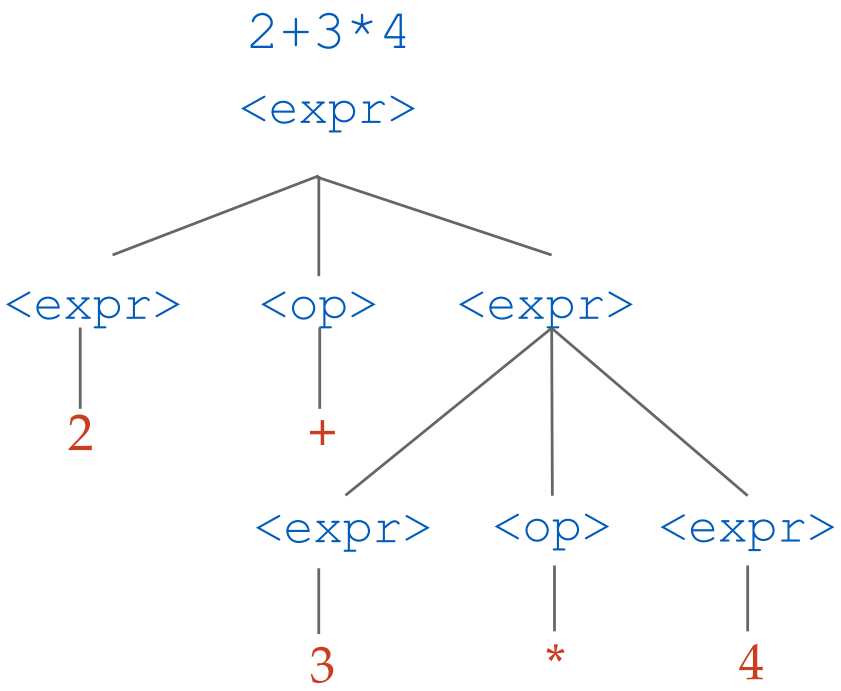
Generate a sentence

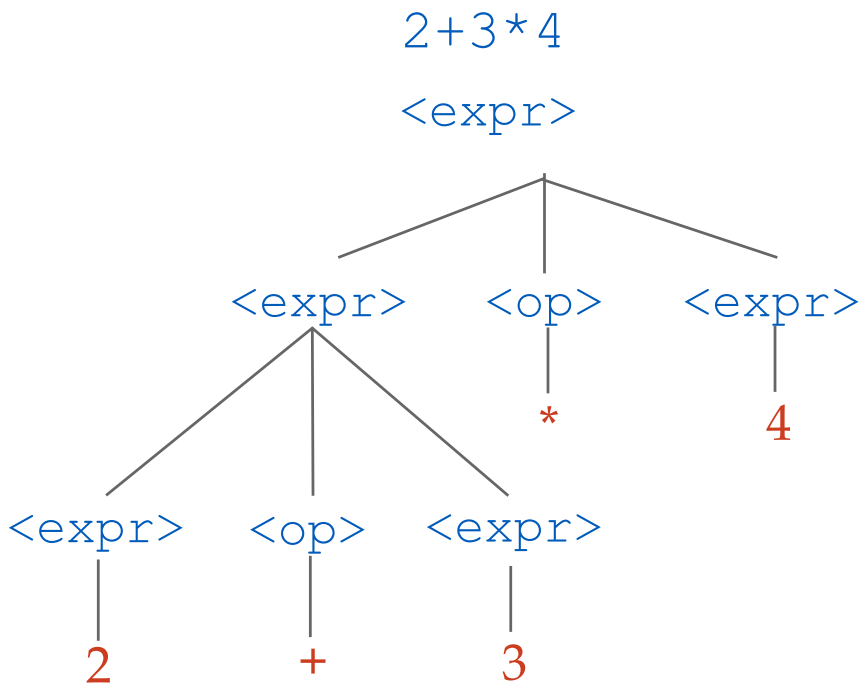
```
<program> ::= <stmts>
              <stmt>
              <var> = <expr>
              a = <expr>
              a = <term> + <term>
              a = <var> + <term>
              a = b + <term>
              a = b + const
```

Ambiguous Grammars

- A grammar is **ambiguous** if and only if it generates a sentential form that has two or more distinct parse trees.

```
<expr> ::= <expr> <op> <expr>
<expr> ::= 1|2|3|4|5|6|7|8|9|0
<op>   ::= +|-|*|/
```





Rewriting a grammar to avoid ambiguity

```
<expr> ::= <expr> <op> <expr>
<expr> ::= 1|2|3|4|5|6|7|8|9|0
<op>   ::= +|*|-|/
```

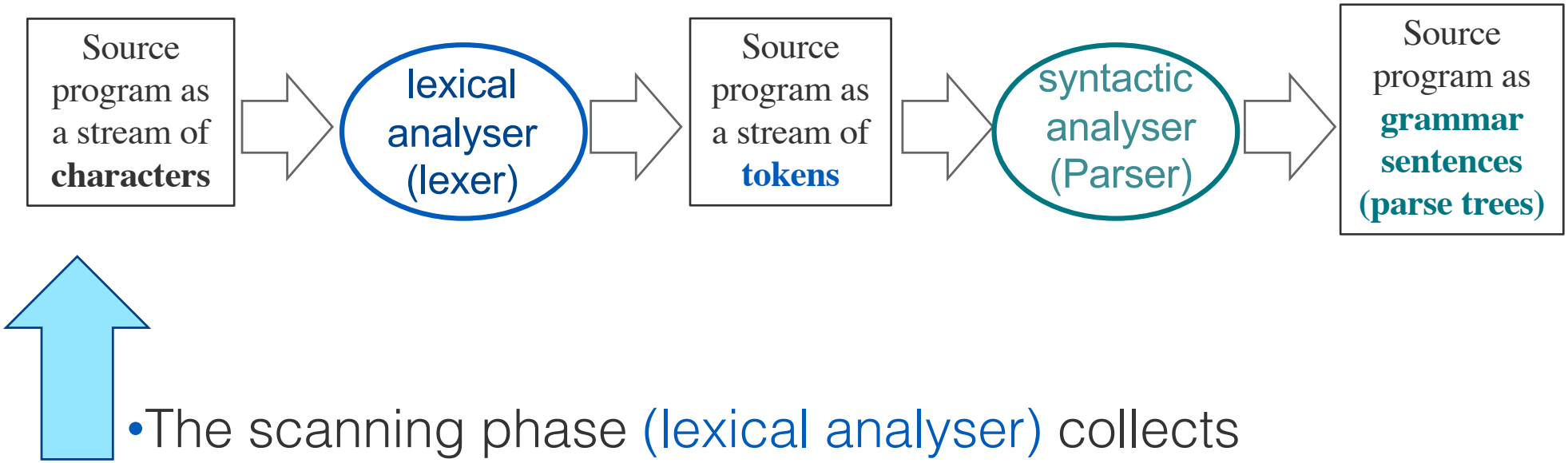
We can rewrite it to:

```
<expr> ::= <term> <addop> <expr> | <term>
<term> ::= <factor> <mulop> <term>
           | <factor>
<factor> ::= <const> | ( <expr> )
<const> ::= 1|2|3|4|5|6|7|8|9|0
<addop>   ::= + | -
<mulop>   ::= * | /
```

Learning Goals for today

- Regular Expressions

Syntactic Structure of Programming Languages



- The scanning phase (lexical analyser) collects characters into tokens (words)
- Parsing phase (syntactic analyser) determines the validity of grammar sentences.

Regular grammars

- A grammar is defined by a set of terminals, a set of nonterminals, a designated nonterminal start symbol, and a finite nonempty set of rules
- I regular grammars we have only three kinds of rules:

```
<non-terminal> ::= terminal
<non-terminal> ::= terminal<non-terminal>
<non-terminal> ::= empty
```

Regular grammars - example

```
<S> ::= a<S>
<S> ::= b<A>
<A> ::= ε
<A> ::= c<A>
```

Recognize a sentence

```
aabcε
aabc<A>
aab<A>
aa<S>
a<S>
<S>
```

Regular expressions

- A compact way to describe regular grammars:
  - A terminal is a regular expression
  - The or | of two expressions is a regular expression describing two alternatives
  - The grouping (-) of expressions is a regular expression describing sequencing of symbols
  - The quantification * of a regular expression is a regular expression describing zero or more occurrence of the same regular expression

Regular expressions - example

```
<S>  ::=  a<S>
<S>  ::=  b<A>
<A>  ::=  ε
<A>  ::=  c<A>
```

We can describe the grammar above by the following expression.

a*bc*

Regular expressions vs context free grammars

- Regular expressions cannot express everything we can express with context free grammar,
- A regular expression recognizer/generator is much simpler to implement than a parser,
- Regular expressions give potentially infinite vocabularies.

Regular Languages – Exercise

What regular expression corresponds to the following regular grammar?

<S> ::= a<A>
<A> ::= ε
<A> ::= c<A>
<A> ::= d<A>

1. a*c*d*
2. a(c*d*)*
3. ac*d*
4. a(cd)*

Regular Languages – Exercise

What regular expression corresponds to the following regular grammar?

<A> ::= a<B>
<A> ::= ε
<B> ::= b<C>
<C> ::= c<A>

1. a*b*c*
2. a(b*c*)*
3. (abc)*
4. a(cb)*