CS 320 : Formal Grammars

Marco Gaboardi

MSC 116

gaboardi@bu.edu

Announcements

- The third programming assignment is posted.
- Due Sunday October 25th 11:59pm.

Plan for today

- Intro to Part 2
- Formal grammars

https://powcoder.com

Assignment Project Exam Help

Add WeChat powcoder

PART II

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

What is the area of Programming Languages?

> Language Design
  • Programming Constructs, Abstractions

> Formal mechanisms to reason about and specify programs / languages
  • Type Systems, Verification

> Compiler Design
  • Optimizations, Program Analysis, JIT

> Language Runtime Design
  • Virtual Machines, Garbage Collection, JIT, Interpreters
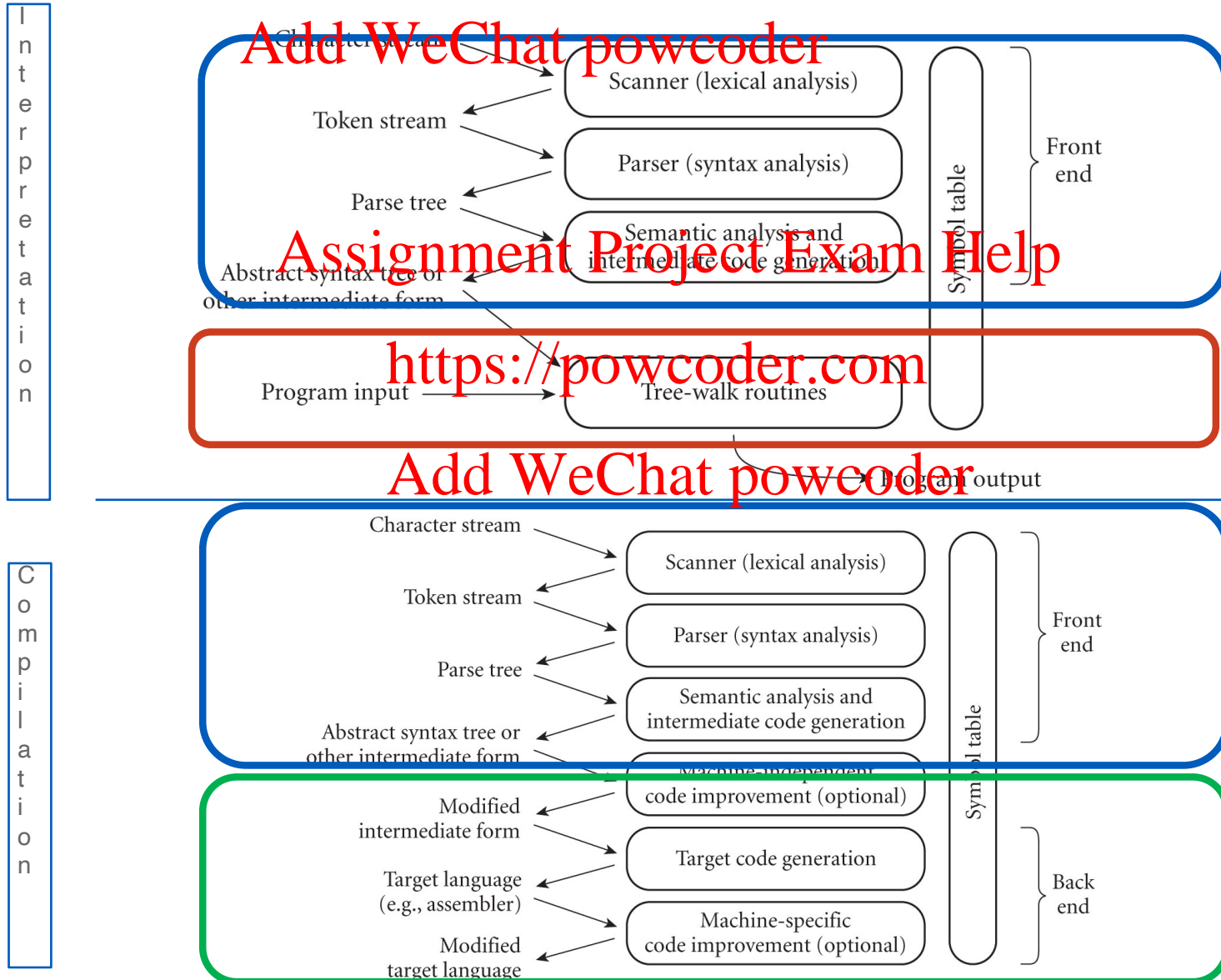
PL

What we have focused on so far.

What we will do for the rest of the class.

Commonalities and differences

Interpretation

Character stream

Scanner (lexical analysis)

Token stream

Parser (syntax analysis)

Parse tree

Semantic analysis and intermediate code generation

Abstract syntax tree or other intermediate form

Symbol table

Front end

Program input → Tree-walk routines

Program output

Compilation

Character stream

Scanner (lexical analysis)

Token stream

Parser (syntax analysis)

Parse tree

Semantic analysis and intermediate code generation

Abstract syntax tree or other intermediate form

Machine-independent code improvement (optional)

Modified intermediate form

Target code generation

Target language (e.g., assembler)

Machine-specific code improvement (optional)

Modified target language

Symbol table

Front end

Back end

TopHat Q1-Q3

Formal Grammars

BNF - Backus Normal/Naur Form

- A grammar is defined by a set of terminals (tokens), a set of nonterminals, a designated nonterminal start symbol, and a finite nonempty set of rules

```
<sentence>        ::= <noun-phrase><verb-phrase>.
<noun-phrase>     ::= <article><noun>
<article>         ::= a | an | the
<noun>            ::= man | apple | worm | penguin
<verb-phrase>     ::= <verb> | <verb><noun-phrase>
<verb>            ::= eats | throws | sees | is
```

Generator vs Recognizer

```
<program> ::= <stmts>
<stmts> ::= <stmt> | <stmt> ; <stmts>
<stmt> ::= <var> = <expr>
<var> ::= a | b
<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
```

https://powcoder.com

Assignment Project Exam Help

Add WeChat powcoder

Assignment Project Exam Help

https://powcoder.com

TopHat Q4-Q8

Add WeChat powcoder

Some examples

```
<sentence>    ::= <noun-phrase><verb-phrase>.
<noun-phrase> ::= <article><noun>
<article>     ::= a | an | the
<noun>        ::= man | apple | worm | penguin
<verb-phrase> ::= <verb> | <verb><noun-phrase>
<verb>        ::= eats | throws | sees | is
```

How do we generate the following sentence?

```
the penguin sees.
```

Some examples

```
<sentence>    ::= <noun-phrase><verb-phrase>.
<noun-phrase> ::= <article><noun>
<article>     ::= a | the
<noun>        ::= man | apple | worm | penguin
<verb-phrase> ::= <verb> | <verb><noun-phrase>
<verb>        ::= eats | throws | sees | is
```

How do we generate the following sentence?

```
a worm eats an apple.
```

Some examples

```
<sentence>      ::= <noun-phrase><verb-phrase>.
<noun-phrase> ::= <article><noun>
<article>       ::= a | an | the
<noun>          ::= man | apple | worm | penguin
<verb-phrase> ::= <verb> | <verb><noun-phrase>
<verb>          ::= eats | throws | sees | is
```

How do we recognize the following sentence?

```
a man throws a penguin.
```

https://powcoder.com

Assignment Project Exam Help

Add WeChat powcoder

Assignment Project Exam Help

https://powcoder.com

TopHat Q9-Q14

Add WeChat powcoder