Programming Language Concepts

Introduction

Spring 2020

Important Information About Me

- James Heliotis
- jehics@rit.edu
- 585-475-6133
- GOL 3515
- My schedule
- Office Hours
  - Monday, Wednesday, Thursday 2:00 PM

Resources

- Instructor Information
  - www.cs.rit.edu/~jeh
- Main Course Web Page
  - www.cs.rit.edu/usr/local/pub/jeh/courses/344
- mycourses
  - Grades
  - Discussion Forums
- PLCC: The course Software [Prof. Timothy Fossum]
  - http://www.cs.rit.edu/usr/local/pub/jeh/PLCC

Let's Review…

- The syllabus
- The schedule

Let's Begin…

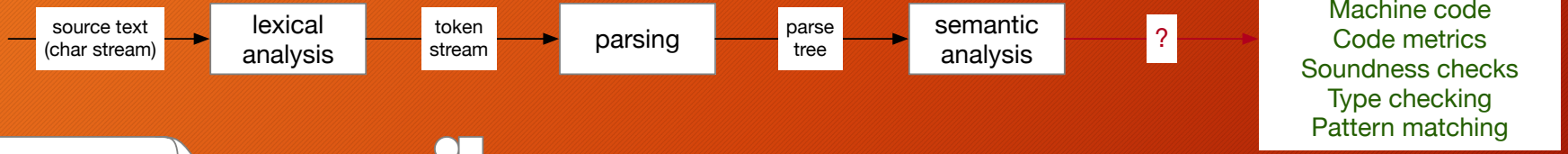
Syntax versus Semantics

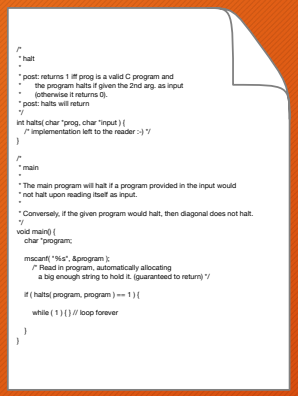
- Can anyone suggest an English sentence that is grammatically well-structured but would be semantically nonsensical?
- Do all natural languages have similar grammars?
- Are all natural languages capable of conveying the same ideas?
- What kind of elements would you talk about if writing a grammar for a natural language like English?

Tokenizing

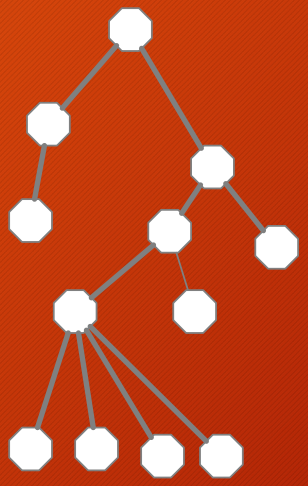
- In programming languages we speak of source text as being composed of tokens.
- `structure Token { token type; lexeme string }`
- State some token types of your favorite programming languages.
  - Give sample lexemes.

The Essence of Language Processing





```
/*
 * halt
 *
 * post: returns 1 iff prog is a valid C program and
 *       the program halts if given the 2nd arg. as input
 *       (otherwise it returns 0).
 * post: halts will return
 */
int halts( char *prog, char *input ) {
    /* implementation left to the reader :-) */
}

/*
 * main
 *
 * The main program will halt if a program provided in the input would
 * not halt upon reading itself as input.
 *
 * Conversely, if the given program would halt, then diagonal does not halt.
 */
void main() {
    char *program;

    mscanf( "%s", &program );
      /* Read in program, automatically allocating
         a big enough string to hold it. (guaranteed to return) */

    if ( halts( program, program ) == 1 ) {

        while ( 1 ) { } // loop forever

    }
}
```



```
/*
 * halts
 *
 * post: returns 1 iff prog is a valid C program and
 *       the program halts if given the 2nd arg. as input
 *       (otherwise it returns 0).
 */
int halts( char *prog, char *input ) {
    /* implementation left to the reader :-) */
}

/*
 * main
 *
 * The main program will halt if a program provided in the input would
 * not halt upon reading itself as input.
 *
 * Conversely, if the given program would halt, then main does not halt.
 */
void main() {
    char *program;

    mscanf( "%s", &program );
        /* Read in program, automatically allocating
           a big enough string to hold it. (guaranteed to return) */

    if ( halts( program, program ) == 1 ) {

        while ( 1 ) { } // loop forever

    }
}
```

Break down this program into its semantic elements.

- Consider both tokens and syntactic structure.

```
public class Div {
    public static void main(String [] args) {
        System.out.println(18/5);
    }
}
```

A Programming Language's Specification

- A programming language spec should include two things:
  - syntax rules (tokens, grammar)
  - semantics

Is Syntax Enough?

- Sometimes you can figure out the semantics from syntax
  - operator +
  - `if` statement
- But what about...
  - Operator precedence?
  - The semantics of generics in Java, or templates in C++?
  - Primitive data types?
  - Behavior of a procedure call?
  - The meaning of keywords like `None`, `null`, `nil`, or `Void`?

Example:
Tokens Without Grammatical Context

- Python: the `int` class has a method `__add__`.
- When you write

  ```
  a + b
  ```

  it is automatically converted to

  ```
  a.__add__( b ).
  ```

- Then why does this not work?

```
5.__add__( 8 )
```

Ensuring Correct Programs

- What can you do to specify what your code is supposed to do, and have it checked automatically?

- In the end, any formal notation is subject to errors and misunderstandings by humans.

How to Describe a Set of Strings for a Token

- Regular Expressions (*regexes*)
  - Strings that represent sets of strings
    ⇒ Some characters are special: . * [ ] etc.
  - For example, Python:
  - https://docs.python.org/3/library/re.html
- Examples follow.

Regular Expression	String	Matches?
	(Python syntax)	
'hell'	'Well hello world!'	YES
'Hell'	'Well hello world!'	NO
'hello|goodbye'	'Well hello world!'	YES
'[0-9]'	When I'm 64 ...	YES
'^[0-9]$'	When I'm 64 ...	NO
'^[0-9]$'	64	NO
'^[0-9]$'	6	YES
'^\d$'	6	YES
r'^\d$'	6	YES
r'^\d*$'	2525	YES
r'^\d*$'	2525a	NO
r'\d*'	2525a	YES
r'\d*'	a	YES
r'\d+'	a	NO
r'\D+'	2525a	YES
r'\D+'	2525	NO

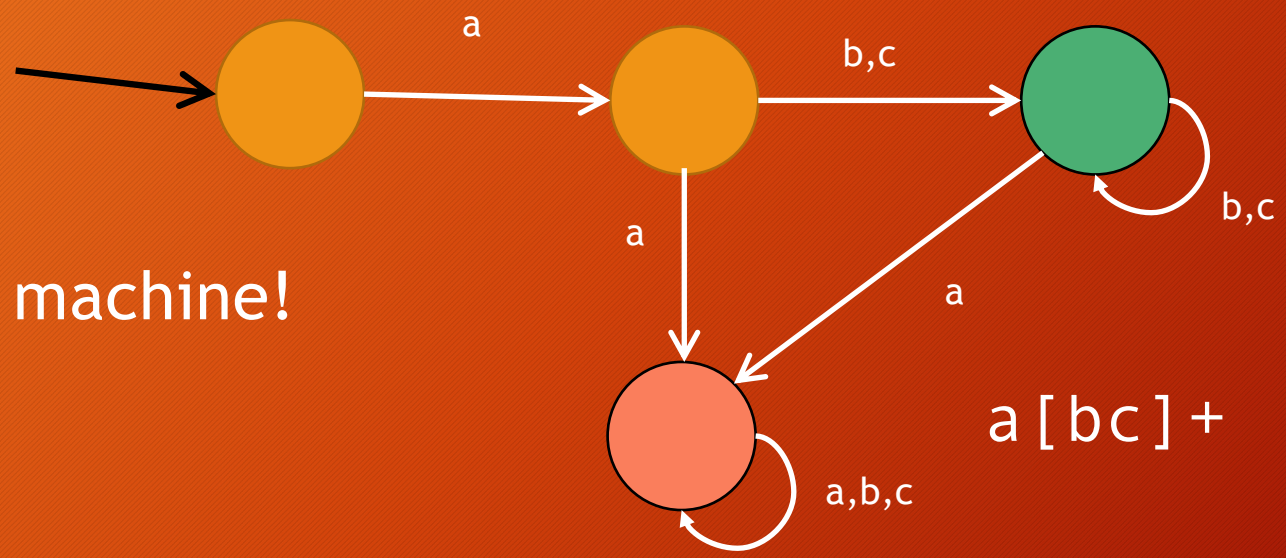
Regexes are Incredibly Useful

- Example: text editors
  - Vim
  - Jetbrains source editor
  - … many more
- Search for calls to method foo:
  - `\.foo(.*)`
- Change brackets to braces in LaTeX hyperref invocations:
  - `s/href\[\(.*\)\]/href{\1}/`

Exercise

- Propose a regular expression that matches floating point constants
  - `5.86`
  - `.83`
  - `19e5`
  - `2.62E+23`
  - `1e-9`
  - `-83.`
- You should not match integers.
- Exponents are at most two (decimal) digits.

How to Match Regular Expressions

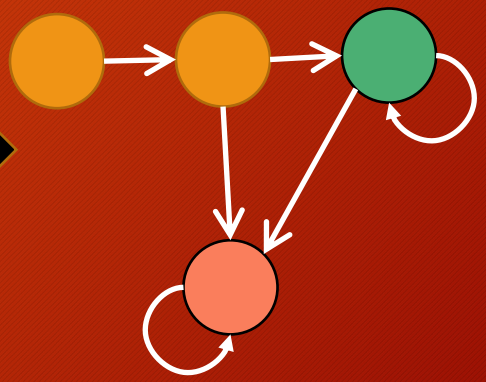


A state machine!

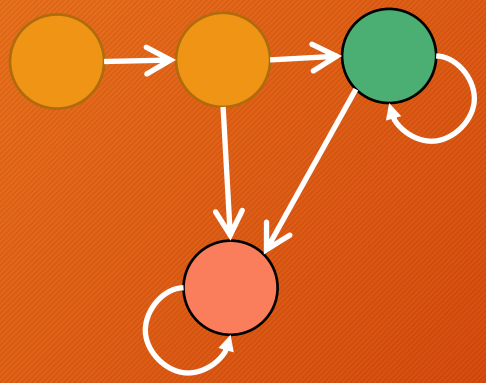
a[bc]+

Regex text can be automatically converted to a finite state machine.

a[bc]+



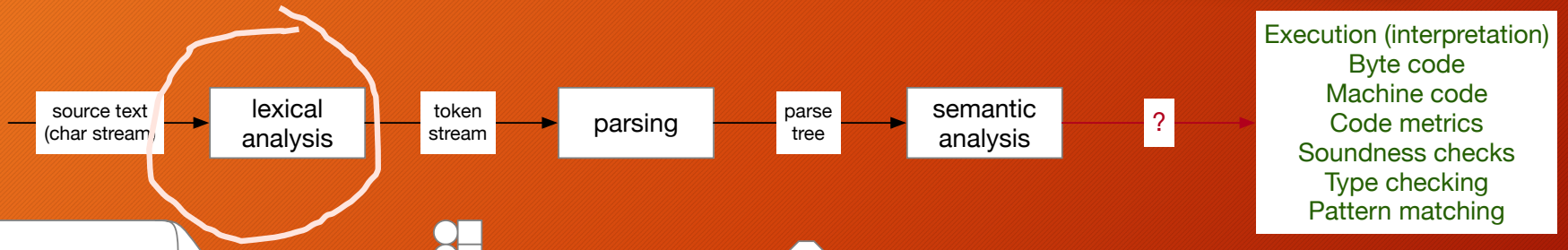
... and a finite state machine can be simulated in conventional software.



current state	a	b	c
S1	S2	S4	S4
S2	S4	S3	S3
S3	S4	S3	S3
S4	S4	S4	S4

Algorithm maintains current state and reads one character at a time to choose next state.

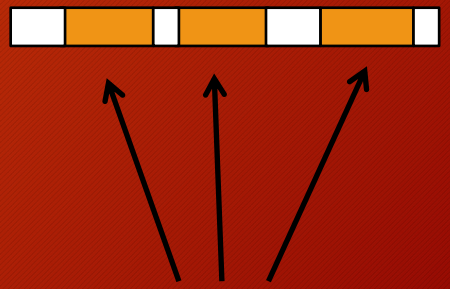
Scanners / Lexical Analyzers




```
/*
 * halt
 *
 * post: returns 1 iff prog is a valid C program and
 *       the program halts if given the 2nd arg. as input
 *       (otherwise it returns 0).
 * post: halts will return
 */
int halts( char *prog, char *input ) {
    /* implementation left to the reader :-) */
}

/*
 * main
 *
 * The main program will halt if a program provided in the input would
 * not halt upon reading itself as input.
 *
 * Conversely, if the given program would halt, then diagonal does not halt.
 */
void main() {
    char *program;

    mscanf( "%s", &program );
        /* Read in program, automatically allocating
           a big enough string to hold it. (guaranteed to return) */

    if ( halts( program, program ) == 1 ) {

        while ( 1 ) { } // loop forever

    }
}
```

How to Make Tokens

1. Open an input stream on the source text.
2. Scan for non-white-space.
3. Capture and remove non-white-space characters from the start of the stream. Put them in a string `lex`.
4. While the scan succeeded
    1. Find a token regex that matches `lex` in its entirety.
    2. Pass the token type and `lex` string to the parser.
    3. Skip white space in input stream.
    4. Capture next non-white-space sequence from stream in `lex`.



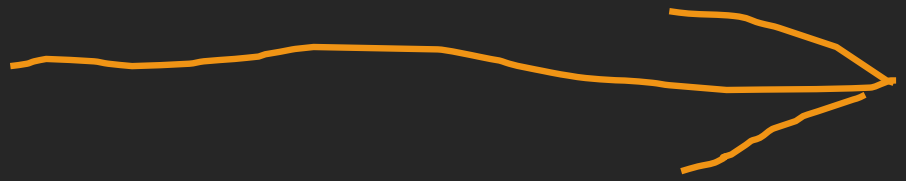
Match each separately.

Whitespace a Necessity?

- Most languages need whitespace, but punctuation can often help.
- How can you tokenize this?

```
ifx>0x=x-1elsey=2
```

Next...



- Build your own lexical scanner!
- Learn how to specify a lexical scanner with PLCC.