Class Information

- Remember our course web page:
  
  https://www.cs.rutgers.edu/courses/314/classes/fall_2017_kremer/

- **Special permission numbers.**
  
  Please be patient since. We have a new system. You should have received in email from me if you requested an SP number

  Course **drop deadline**: Tuesday, Sept. 12.
  
  Course **add deadline**: Wednesday, Sept. 13.

- Recitations will start next week.

- Our sakai / piazza system up! First homework will be posted next week.

- TA office hours will be posted next week.

  My office hours are **Mondays, noon - 1:30pm, CoRE 305** (small conference room)
Syntax & Semantics of Prog. Langs.

Syntax:
Describes what a legal program looks like

Semantics:
Describes what a correct (legal) program means

A formal language is a (possibly infinite) set of sentences (finite sequences of symbols) over a finite alphabet Σ of (terminal) symbols: \( L \subseteq \Sigma^* \)

Examples:

- \( L = \{ \text{identifiers of length 2} \} \) with \( \Sigma = \{a, b, c\} \)
- \( L = \{ \text{strings of only 1s or only 0s} \} \)
- \( L = \{ \text{strings starting with \$ and ending with \#}, \text{ and any combination of 0s and 1s inbetween} \} \)
- \( L = \{ \text{all syntactically correct Java programs} \} \)

Claim: The larger the language, the harder it is to formally specify the language. In other words, it get’s harder for each \( i: L_1 \subset L_2 \subset L_3 \ldots \subset L_i \subset \ldots \). True or false?
Syntax and Semantics: How does it work?

Example: Syntactic representation of “values”

What do the following syntactic expressions have in common?

XI
1011
B
\lambda fx. (f(f(f(f(f(f(f(f(f(f(x))))))))))))
$ \text{#} $
3 + 20 - (2 \times 6)$
Syntax and Semantics: How does it work?

Example: Syntactic representation of “values”

What do the following syntactic expressions have in common?

\[ \text{XI} \]
\[ 1011 \]
\[ B \]
\[ \lambda f x. (f(f(f(f(f(f(f(f(f(f(f(x)))))))))))) ) \]
\[ $ ||||||||| \# \]
\[ 3 + 20 - (2 \times 6) \]

**Answer:** They are possible representations of the integer value “11” (written as a decimal number)

What is computation?

**Possible answer:** A (finite) sequence of syntactic manipulations of value representations ending in a “normal form” which is called the result. Normal forms cannot be manipulated any further.
Syntax and Semantics: How does it work?

Here is a “game” (rewrite system):

**input**: Sequence of characters starting with $ and ending with #, and any combination of 0s and 1s inbetween.

**rules**: You may replace a character pattern $X$ at any position within the character sequence on the left-hand-side by the pattern $Y$ on the right-hand-side: $X \Rightarrow Y$:

- **rule 1**: $1 \Rightarrow 1 \&$
- **rule 2**: $0 \Rightarrow 0$
- **rule 3**: $\& 1 \Rightarrow 1$
- **rule 4**: $\& 0 \Rightarrow 0 \&$
- **rule 5**: $\# \Rightarrow \rightarrow A$
- **rule 6**: $\& \# \Rightarrow \rightarrow B$

Replace patterns using the rules as often as you can. Application of rules is “atomic”, i.e., can only be done one at a time. There is no order constraint among sets of rules that can be applied (non-deterministic selection). When you cannot replace a pattern any more, stop.
Syntax and Semantics: How does it work?

example input:

$ 0 0 #$

$0 0 # is rewritten as $0 \$ 0 # by rule 2

$0 0 # is rewritten as $0 0 \$ # by rule 2

$0 0 \$ # is rewritten as $0 0 \rightarrow A$ by rule 6

no more rules can be applied (STOP)

More examples:

$0 1 1 0 1 #$

$1 0 1 0 0 #$

$1 1 0 0 1 #$

Questions

• Can we get different “results” for the same input string?

• Does all this have a meaning (semantics), or are we just pushing symbols?
Syntax without Semantics?

There will be rewrite systems problems in the first homework.
Front end of a compiler

Front End: syntax & (static) semantics analyzer, il code generator (syntax-directed translation)

Front End Responsibilities:

• recognize legal programs
• report errors
• produce il (intermediate language / representation)
• preliminary storage map
• shape the code for the back end

Much of front end construction can be automated
Syntax and Semantics of Prog. Languages

The syntax of programming languages is often defined in two layers: *tokens* and *sentences*.

- **tokens** – basic units of the language
  - Question: How to spell a token (word)?
  - Answer: regular expressions

- **sentences** – legal combination of tokens in the language
  - Question: How to build correct sentences with tokens?
  - Answer: (context-free) grammars (CFG) E.g.,

    Backus-Naur form (BNF) is a formalism used to express the syntax of programming languages.
Formalisms for Lexical and Syntactic Analysis

1. Lexical Analysis: Converts source code into sequence of tokens.

2. Syntax Analysis: Structures tokens into parse tree.

Two issues in Formal Languages:

- **Language Specification** → formalism to describe what a valid program (sentence) looks like.

- **Language Recognition** → formalism to describe a machine and an algorithm that can verify that a program is valid or not.

For (2. Syntax Analysis), we use context-free grammars to specify programming languages. Note: recognition, i.e., parsing algorithms using PDAs (push-down automata) will be covered in CS415.

For (1. Lexical Analysis), we use regular grammars for specification and finite (state) automata for recognition.
Lexical Analysis (Scott 2.1, 2.2)

character sequence

\[ i \_ f \_ a \_ \_ \_ \_ \_ \_ \_ = b \_ t \_ h \_ e \_ n \_ c : = 1 \]

\[ \text{scanner} \]

\[ \text{if} \rightarrow \text{id} <a> \rightarrow \Rightarrow \text{<=} \rightarrow \text{id} <b> \]

\[ \Rightarrow \text{then} \rightarrow \text{id} <c> \rightarrow \Rightarrow \text{:=} \rightarrow \text{num} <1> \]

token sequence

**Tokens** (Terminal Symbols of CFG, Words of Lang.)

- Smallest “atomic” units of syntax
- Used to build all the other constructs
- Example, Pascal:
  
  **keywords**: program begin if then ...
  
  = * / - < > = <= >= <>
  
  ( ) [ ] ; := . , ...
  
  **number** (Example: 3.14 28 ...)
  
  **identifier** (Example: b square addEntry ...)

CS 314

fall’17 lecture 2, page 11
Lexical Analysis (cont.)

Identifiers
- Names of variables, etc.
- Sequence of terminals of restricted form;
  Example, Pascal: $A31$, but not $1A3$
- Upper/lower case sensitive?

Keywords
- Special identifiers which represent tokens in the language
- May be reserved (reserved words) or not
  - E.g., Pascal: “if” is reserved.
  - E.g., FORTRAN: “if” is not reserved.

Delimiters – When does character string for token end?
- Example: identifiers are longest possible character sequence that does not include a delimiter
- Few delimiters in Fortran (not even ‘\t’)
  - DO I = 1.5 same as DOI=1.5
- Most languages have more delimiters such as ‘\t’, new line, keywords, ...
## Regular Expressions

A syntax (notation) to specify regular languages.

<table>
<thead>
<tr>
<th>RE $r$</th>
<th>Language $L(r)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>${a}$</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>${\epsilon}$</td>
</tr>
<tr>
<td>$r \mid s$</td>
<td>$L(r) \cup L(s)$</td>
</tr>
<tr>
<td>$rs$</td>
<td>${rs \mid r \in L(r), s \in L(s)}$</td>
</tr>
<tr>
<td>$r^+$</td>
<td>$L(r) \cup L(rr) \cup L(rrr) \cup \ldots$</td>
</tr>
<tr>
<td></td>
<td>(any number of $r$’s concatenated)</td>
</tr>
<tr>
<td>$r^*$</td>
<td>${\epsilon} \cup L(r) \cup L(rr) \cup L(rrr) \cup \ldots$</td>
</tr>
<tr>
<td></td>
<td>($r^* = r^+ \mid \epsilon$)</td>
</tr>
<tr>
<td>$(s)$</td>
<td>$L(s)$</td>
</tr>
</tbody>
</table>

(all left-assoc. in order of increasing precedence.)

⇒ **Note:** Inductive definition!
Next Lecture

More on regular expressions

Finite state machines

CFGs, BNF, derivations, parse tree, ambiguity, top-down parsing

Things to do:

• Please check our web site every other day for announcements etc.;
• read Scott, Ch. 2.3 - 2.5 (skip 2.3.3 Bottom-up Parsing)