Class Information

• **Special permission numbers.**
  
  Unfortunately, it is not possible to accommodate everyone. I sent out a second round of SP numbers. If you wanted section 5 or 6, emails will go out sometime today.

  Please put your name down again on the SP request list.

• Recitation will start next week.

• Next Tuesday is a Rutgers Monday, i.e., a Monday schedule applies. Next class is on Friday, September 11

• Office hours will be posted early next week.
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 \( \Sigma \) of (terminal) symbols: \( L \subseteq \Sigma^* \)

Examples:

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

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?

Syntactic representation of “values”

What do the following syntactic expressions have in common?

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

*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** $1 \# \Rightarrow \rightarrow A$
- **rule 6** $\& \# \Rightarrow \rightarrow B$

Replace patterns using the rules as often as you can. When you cannot replace a pattern any more, stop.
Syntax and Semantics: How does it work?

example input:

$ 0 0 \#$

$0\ 0\ # \text{ is rewritten as } 0\ 0\ # \text{ by rule 2}$

$0\ 0\ # \text{ is rewritten as } 0\ 0\ # \text{ by rule 2}$

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

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?

Syntax without semantics is not useful!

Two problems on rewrite systems in the first homework.
Front end of a compiler

Parser: syntax & semantic analyzer, \textit{il} code generator
(syntax-directed translator)

Front End Responsibilities:

- recognize legal programs
- report errors
- produce \textit{il}
- preliminary storage map
- shape the code for the back end

\textit{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 andSyntactic 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), 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), we use **regular grammars/expressions** for specification and **finite (state) automata** for recognition.
Lexical Analysis (Scott 2.1, 2.2)

character sequence

\[ \text{if } \text{id}< \text{a} \leq \text{b} \text{then id } \text{c} := \text{1} \]

\[ \text{scanner} \]

\[ \text{if } \rightarrow \text{id<a} \rightarrow \leq \rightarrow \text{id<b} \]
\[ \rightarrow \text{then } \rightarrow \text{id<c} \rightarrow := \rightarrow \text{num<1} \]

\[ \text{token sequence} \]

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 ...
Lexical Analysis (cont.)

Identifiers

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

Keywords

- Special identifiers which represent tokens in the language
- May be reserved (\textit{reserved words}) or not
  - E.g., Pascal: \texttt{“if”} is reserved.
  - E.g., FORTRAN: \texttt{“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 \texttt{‘\char 38’})
    - \texttt{DO I = 1.5} same as \texttt{DOI=1.5}
  - Most languages have more delimiters such as \texttt{‘\char 38’}, 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>$\varepsilon$</td>
<td>${\varepsilon}$</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>(any number of r’s concatenated)</td>
<td></td>
</tr>
<tr>
<td>$r^*$</td>
<td>${\varepsilon} \cup L(r) \cup L(rr) \cup L(rrr) \cup \ldots$</td>
</tr>
<tr>
<td>($r^* = r^+</td>
<td>\varepsilon$)</td>
</tr>
</tbody>
</table>

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

⇒ **Note:** Inductive definition!
## Examples of Expressions

<table>
<thead>
<tr>
<th>RE</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>bc</td>
</tr>
<tr>
<td>(a</td>
<td>b)c</td>
</tr>
<tr>
<td>aε</td>
<td></td>
</tr>
<tr>
<td>a*</td>
<td>b</td>
</tr>
<tr>
<td>ab*</td>
<td></td>
</tr>
<tr>
<td>ab*</td>
<td>c+</td>
</tr>
<tr>
<td>(a</td>
<td>b)*</td>
</tr>
<tr>
<td>(0</td>
<td>1)*1</td>
</tr>
</tbody>
</table>
### Examples of Expressions - Solution

<table>
<thead>
<tr>
<th>RE Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>(a</td>
</tr>
<tr>
<td>a\epsilon</td>
</tr>
<tr>
<td>a*</td>
</tr>
<tr>
<td>ab*</td>
</tr>
<tr>
<td>ab*</td>
</tr>
<tr>
<td>(a</td>
</tr>
<tr>
<td>(0</td>
</tr>
</tbody>
</table>
Finite state machines (NFA and DFA) and their implementations in a scanner

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

Things to do:

• First homework will be posted next Tuesday. Please check our web site; Homework will be due on Friday, September 18, before class (10 minutes grace period).

• read Scott, Ch. 2.3 - 2.5 (skip 2.3.3 Bottom-up Parsing)