Lecture 7:

- Formal Grammars (cont)

Announcements:

- HW-1 out
- Quiz 2 Friday – Lexical analysis, grammars

Using Parentheses: Can use parentheses to simplify rules

$S \rightarrow (\,\texttt{ab}\,)^* \;\mid\; (\,\texttt{ba}\,)^*$

Check In: What is the language of this grammar rule?

Check In: How can the above be rewritten so it doesn't use parantheses?

$S \rightarrow T^* \;\mid\; U^*$

$T \rightarrow \texttt{ab}$

$U \rightarrow \texttt{ba}$

Note: alternation has lower precedence than other "operators"

- The rule: $S \rightarrow\; \texttt{a}\;\texttt{b}^*\;\texttt{c} \;\mid\; \texttt{d}^*\;\texttt{e}$
- Is the same as: $S \rightarrow (\texttt{ab}^*\texttt{c}) \;\mid\; (\texttt{d}^*\texttt{e})$

Check In: What is the language of this grammar rule?

$S \rightarrow (\texttt{a} \mid \texttt{b})^* \;\mid\; (\texttt{d} \mid \texttt{e})^*$

The language consists of the empty string, all combinations of **a** and **b**, and all combinations of **d** and **e**

Recursion

Either directly when used in same rule, or indirectly ...

Direct Example: $S \to \texttt{a}\, S\, \texttt{b} \mid \epsilon$ ... S occurs (directly) in S rule

- S yields the strings $a^i\, b^i$ for $i \geq 0$
- note this is not possible to express using * (Kleene star)
- however, * can be implemented using recursion (w/ the empty string ...)

Indirect Example:

$S \to T \mid \epsilon$

$T \to \texttt{a}\, S\, \texttt{b}$

Derivations: can help decipher language of grammars, especially with recursion

- A derivation starts with a <u>single</u> non-terminal (e.g., S)
- Repeatedly replaces <u>one</u> non-terminal until only terminals remain
- Each "step" in the replacement is denoted by $\Rightarrow$

Example using the Indirect recursive grammar above:

$$S \Rightarrow T \Rightarrow \texttt{a}\, S\, \texttt{b} \Rightarrow \texttt{a}\, T\, \texttt{b} \Rightarrow \texttt{aa}\, S\, \texttt{bb} \Rightarrow \texttt{aabb}$$

Check In: Give a derivation of abcd starting from S using grammar:

$S \rightarrow \mathtt{a}\, T\, U\, \mathtt{d}$

$T \rightarrow \mathtt{b}\, T \mid \epsilon$

$U \rightarrow U\, \mathtt{c} \mid \mathtt{c}$

$$S \Rightarrow \mathtt{a}\, T\, U\, \mathtt{d} \Rightarrow \mathtt{a}\, \mathtt{b}\, T\, U\, \mathtt{d} \Rightarrow \mathtt{a}\, \mathtt{b}\, U\, \mathtt{d} \Rightarrow \mathtt{a}\, \mathtt{b}\, \mathtt{c}\, \mathtt{d}$$

MyPL Literals

We can use grammar rules to define a PL's literal values

Note that we use BNF below ...

- where `::=` used instead of $\rightarrow$
- and non-terminals as <*name*>

BOOL_VAL ::= 'true' | 'false'

INT_VAL ::= <pdigit> <digit>* | '0'

DOUBLE_VAL ::= INT_VAL '.' <digit> <digit>*

STRING_VAL ::= '"' <character>* '"'

ID ::= <letter> (<letter> | <digit> | '_')*

<letter> ::= 'a' | ... | 'z' | 'A' | ... | 'Z'

<pdigit> ::= '1' | ... | '9'

<digit> ::= '0' | <pdigit>

... where <**character**> is any symbol (letter, number, etc.) except '"'

Terminology and Next Steps

A **regular** language is one that can be defined only using:

- concatenation, alternation, and Kleene star (plus simple rules $S \to \texttt{a}$)
- but no recursion (except for Kleene star)

A **context free** language is one that can be defined using:

- any of the constructs (including recursion)
- but cannot have terminals on the left-hand-side of rules

A **context sensitive** language allows terminals on the left-hand side of rules

- e.g., $\texttt{a}\,A \to \texttt{a}\,\texttt{b}\,B$ substrings $\texttt{a}A$ replaced by $\texttt{ab}B$
- this rule is matched only when a string has an `a` before `A`
- the initial `a` serves as context for when to apply the rule

PL syntax is defined using context-free grammars

- but typically not enough to prohibit all invalid programs
- which is a reason for semantic analysis
- we will talk later about additional issues in grammars (e.g., ambiguity)

Some example syntax rules: ... use EBNF or variants

- For Java: `https://docs.oracle.com/javase/specs/jls/se7/html/jls-18.html`
- For Python: `https://docs.python.org/3/reference/grammar.html`
- Summary of C++: `https://alx71hub.github.io/hcb/`

Summary – Things to Know

1. Basic rules, concatenation, alternation, kleene star
2. How to rewrite a rule to remove alternation
3. How recursion (direct, indirect) generally works with grammar rules
4. How to rewrite Kleene Star using recursion
5. Basic idea of a derivation, how to do basic derivations