To-Do

Lecture 6 introduced context free grammars. Before the next lecture, do the following.

1. ***Continue Working on HW-1.*** You should be working on HW-1, which is due in one week. Please be sure to keep up to date with piazza and post questions that you have.
2. ***See additional information below.*** Regular and context-free grammars are described further below. In addition, a grammar is provided for specifying MyPL literals is given.
3. ***Do the practice questions below.*** You will need to understand how to create formal grammars, give derivations, and define the langauge of a grammar. Do the practice questions below before the next lecture.

Additional Information

Regular Grammars. A regular grammar does not involve recursive grammar rules. Concatenation, alternation, the empty string, and the star operator are each allowed.

Context-Free Grammars. Unlike a regular grammar, a context free grammar allows the use of recursion. Every regular grammar is also context free (i.e., the set of regular languages is a subset of the set of context free languages). The example of a language that is context-free but not regular is $\{\texttt{a}^i\texttt{b}^i \mid i \geq 0\}$, i.e., the set of strings with `a`'s followed by the same number of `b`'s. A context-free grammar for this language is:

$$S \rightarrow \texttt{a}\ S\ \texttt{b} \mid \varepsilon$$

It is not possible to simulate the recursion in this case using the star operator.

MyPL Literal Values. We can use a regular grammar to define the allowable literal values in MyPL. The rules are given in Fig. 1 using "BNF" notation, which differs from the notation used in class as follows.

- `::=` replaces $\rightarrow$
- non-terminals are written in <>'s
- terminals are in single quotes

Below non-terminals that are in all uppercase represent token types, and non-terminals that are in all lowercase are used solely for specifying the grammar (they do not represent token types). The `<character>` non-terminal represents any character.

```
  <BOOL_VAL>  ::=  'true' | 'false'
   <INT_VAL>  ::=  <pdigit> <digit>* | '0'
<DOUBLE_VAL>  ::=  ( <pdigit> <digit>* | '0' ) '.' <digit> <digit>*
<STRING_VAL>  ::=  '"' <character>* '"'
        <ID>  ::=  <letter> ( <letter> | <digit> | '_' )*
    <letter>  ::=  'a' | ... | 'z' | 'A' | ... | 'Z'
    <pdigit>  ::=  '1' | ... | '9'
     <digit>  ::=  '0' | <pdigit>
```

Figure 1: Formal grammar specifying literal values in MyPL.

Practice Questions

Design a *grammar* for each of the following languages. An answer key is given on the following page. Note that there are many different grammars for a given language. Your grammars should exactly capture the language, i.e., contain all well-formed strings and no strings that are not in the language.

1. The set of "balanced" curly braces (with alphabet symbols `{` and `}`). Examples of strings in the language are `"{}"`, `"{{}}"`, `"{}{}"`, `"{{}{}}{}"`, and so on.
2. The sequences of `a`'s and `b`'s that consist of an odd number of `a`'s followed by the same number of `b`'s.
3. The sequences of `a`'s followed by `b`'s that contain twice as many `a`'s as `b`'s. Note there must be at least one `b`.
4. All sequences of `a`'s followed by `b`'s that contain strictly more `a`'s than `b`'s. Note there can be zero `b`'s, but there must be at least one `a`.
5. All sequences of `a`'s and `b`'s that contain the same number of `a`'s and `b`'s in any order. Examples include `""`, `"ab"`, `"abab"`, `"aabb"`, `"baab"`, `"aabaabbb"`, and so on.
6. The set of strings for performing zero or more sequences of p addition operations over `0` and `1`. Examples include `"0"`, `"1"` , `"1 + 1"`, `"1 + 0 + 1"`, `"0 + 0 + 1 + 0"`, and so on.
7. The set of boolean expressions over `true` and `false`. Assume *and* (conjuction) is represented as $\wedge$, *or* (disjunction) as $\vee$, and *not* (negation) as $\neg$. Your grammar should allow expressions to be grouped using square brackets `[` and `]`. Examples:

   `"true"`
   `"true` $\wedge$ `false"`
   `"[false` $\vee$ `true]` $\wedge$ $\neg$ `false"`
   `"`$\neg$`[[false` $\vee$ `true]` $\wedge$ $\neg$`[true` $\wedge$ `false]]"`
   and so on

Answer Key

1. $S \to \texttt{\{\}} \mid \texttt{\{}S\texttt{\}} \mid SS$
2. $S \to \texttt{ab} \mid \texttt{aa}S\texttt{bb}$
3. $S \to \texttt{aab} \mid \texttt{aa}S\texttt{b}$
4. $S \to \texttt{a} \mid \texttt{a}S \mid \texttt{a}S\texttt{b}$
5. $S \to \texttt{a}S\texttt{b} \mid \texttt{b}S\texttt{a} \mid \texttt{ab}S \mid \texttt{ba}S \mid \varepsilon$
6. $S \to \texttt{0} \mid \texttt{1} \mid S + S$
7. $S \to \texttt{true} \mid \texttt{false} \mid S \wedge S \mid S \vee S \mid \texttt{[}S\texttt{]} \mid \neg S$