1 Administrivia

- Problem Set 2 due last night
- Project 2 out, due Oct. 21 and Nov. 1
- Lab 2 this Friday

2 Satisfiability

2.1 Problem Definition

The satisfiability problem, or SAT, is the problem of finding, for a given boolean expression consisting of one or more variables, an assignment that causes the expression to evaluate to true. For example:

\[ x + y + z' \]

has for one of its solutions the assignment \((x, y, z) = (T, T, T)\).

In most cases, we will be studying a special version of SAT known as 3-SAT, which is actually provably the same problem as the general case. The difference is that we require our equations to be in conjunctive normal form, which looks like:

\[(x_1 + x_2 + x_3) \cdot (x'_1 + x_2 + x_5) \cdot (x_2 + x'_4 + x'_6) \cdot \ldots\]

Each clause in this case is the boolean disjunction (OR) of three literals, which can either be a variable of the negation of a variable. The formula itself is then defined as the conjunction (AND) of all of the clauses.

2.2 DPLL

The DPLL, or Davis-Putnam-Logemann-Loveland, algorithm is a simple backtracking-based routine for solving instances of 3-SAT. It has a few basic steps for solving a given formula, \(\phi\). The pseudocode for DPLL(\(\phi\)) looks like:

1. If \(\phi\) is consistent and completely assigned, return true
2. If any clause in \(\phi\) is empty, return false
3. If there are any unit clauses (clauses with only one literal), apply unit propagation to them such that for the single variable in the clause, \(x\):
   - \(x\) is replaced with \(T\) throughout \(\phi\)
   - \(x'\) is replaced with \(F\) throughout \(\phi\)
4. Pick the next unassigned literal, \(y\), in \(\phi\)
   - Arbitrarily assign either \(T\) or \(F\) to \(y\) and recurse.
   - If that fails, try the other option.

3 Grammars

It is often useful to define a data type in terms of a grammar. This is a relatively simple abstraction technique that we can use to define fairly complex data structures. Here is a simple recursive grammar that defines a linked list:

\[
\text{List} <E> = \text{Empty} \mid \text{Cons} <E> \\
\text{Cons} <E> = \text{Cons}(\text{first}: E, \text{rest}: \text{List}<E>)
\]