Software Analysis and Verification - Fall 2025

Ruzica Piskac

October 2, 2025

The deadline for Project 2 is October 21, 11:59PM.

You are encouraged to work with a partner.

There is no starter code provided for this assignment, but you may be able to reuse some components from Project 1 such as the lexer and parser. You may use any programming language you wish.

1 Simplex Algorithm

Implement the simplex algorithm as described in lecture 14. Your implementation should take as input a set of linear equalities and inequalities and check if they are satisfiable. If they are, then print the solution, and if they are not, then print "UNSAT".

The input will be provided in SMT-LIB format, but restricted to a sub-language as follows:

$$
\begin{aligned}
\langle script \rangle &:= \langle command \rangle^+ \\
\langle command \rangle &:= \texttt{(assert } \langle formula \rangle \texttt{)} \\
\langle formula \rangle &:= \texttt{(and } \langle atom \rangle^+ \texttt{)} \mid \langle atom \rangle \\
\langle atom \rangle &:= \texttt{(}\langle boolop \rangle\ \langle term \rangle\ \langle term \rangle\texttt{)} \\
\langle boolop \rangle &:= \texttt{<} \mid \texttt{>} \mid \texttt{>=} \mid \texttt{<=} \mid \texttt{=} \\
\langle term \rangle &:= \texttt{(+ } \langle term \rangle^+ \texttt{)} \\
&\mid \texttt{(- } \langle term \rangle^+ \texttt{)} \\
&\mid \texttt{(* } \langle c \rangle\ \langle term \rangle \texttt{)} \\
&\mid \langle c \rangle \\
&\mid \langle var \rangle
\end{aligned}
$$

A variable $\langle var \rangle$ is a string containing only alphanumeric characters and starting with a letter. A constant $\langle c \rangle$ is a rational number (eg., `-1` , `-1/2` , `0` , `1/3` , `2`). The semantics of the `-` operator should be as follows: if one argument is provided, perform negation; if more than one argument is provided, perform subtraction. Negative rational numbers may be written as `(-4)` (with parentheses and a space) or `-4` (with neither parentheses nor a space). Additional whitespace around tokens may be present at any time.

For example, for input formula $x \geq 1 \wedge 2 * x \leq 1$, the input to your tool could be:

```
(assert (and (>= x 1) (<= (* 2 x) 1)))
```

Equally, it could be written:

```
(assert (>= x 1))
(assert (<= (* 2 x) 1))
```

The desired output for unsatisfiable formulae should be formatted exactly as below:

```
UNSAT
```

The desired output for satisfiable formulae should be newline-separated equations between variables and constants. For example, the input formula of $x \geq 1 \wedge 2 * x + y \leq 1$ should produce the below output (in any order of x, y):

```
x=1
y=-1
```

2 Grading

Grading will primarily be based on performance on a set of benchmark constructed by the teaching staff. All the benchmarks will be publicly posted after grades are released. These benchmarks are worth 70 points.

You will receive 20 points for producing reasonable output showing the operation of your tool (as determined by manual inspection.)

You will receive 10 points for submitting a README. This should include instructions on how to build and run your project. Try to minimize the number of steps required to build, for example, include a make script that does as much as possible automatically. Please include the exact command line steps required to build your tool on a Unix-like system (such as the Zoo.)

If you would like to extend your tool to find integer solutions (and determine unsatisfiability for integer solutions) you will receive up to 20 bonus points. The default of your tool should be to find rational solutions, and use a command line flag (which should be specified in the README) to compute integer solutions instead.