Technical Notes on Using a SAT Solving in Python

Ulle Endriss, September 2019

A SAT solver is a highly optimised tool that can be used to automatically check the satisfiability of a propositional formula (which must be provided in CNF). In these notes I explain how you can use such a solver from within a simple Python program. This is useful if you want to try out some of the stuff presented in my course on automated reasoning for social choice theory.

Make sure you have Python3 installed on your machine. If you are not sure how to do this, probably this will work:

https://www.python.org/downloads/

Then install pylgl, a Python interface for Lingeling, one of the best performing SAT solvers available:

https://pypi.python.org/pypi/pylgl/

You can use this tool to check whether a given formula in CNF is satisfiable. If it is, one of the models satisfying the formula will be returned. You can also generate the list of all models that satisfy your formula (this might be slow for formulas with very many models). Clauses are represented as lists of positive and negative integers, representing positive and negative literals. A formula in CNF then is represented as a list of such lists. Thus, the formula (P v Q) ^ (-P v -Q v R) would be represented as [[1,2], [-1,-2,3]].

After you have started up Python3, run the following command to make the functions solve() and itersolve() available to you:

```
>>> from pylgl import solve, itersolve
```

Now you can use the function solve() to confirm that the formula (P v Q) ^ (-P v -Q v R) is satisfiable and to compute one of its models (the first one the system happens to find):

```
>>> solve([ [1,2], [-1,-2,3] ])
[1, 2, 3]
```

Thus, if we set all three variable to TRUE, then our formula is satisfied. Note that in the input a list such as [-1,-2,3] denotes a disjunction, while in the output a list denotes a model (which you might think of as a conjunction of literals).

Suppose you want to compute a second model of the same formula. One way of doing this would be to add one further clause to the CNF that explicitly rules out the first model we found:

```
>>> solve([ [1,2], [-1,-2,3], [-1,-2,-3] ])
[1, -2, -3]
```

So another way of satisfying the formula (P v Q) ^ (-P v -Q v R) would be to set P to TRUE and the other two variables to FALSE.

The next example shows that the formula (P v Q) ^ (-P v Q) ^ (P v -Q) ^ (-P v -Q) is not satisfiable:

```
>>> solve([ [1,2], [-1,2], [1,-2], [-1,-2] ])
'UNSAT'
```

You can use the function itersolve() compute all of the models of a given formula. Note that you need to use the function list() to transform the output into a list that can be displayed:

```
>>> list( itersolve([ [1,2], [-1,-2,3] ]) )
[[1, 2, 3], [1, -2, 3], [-1, 2, 3], [-1, 2, -3], [1, -2, -3]]

>>> list( itersolve([ [1,2], [-1,-2,3], [-1,-2,-3] ]) )
```

```
[[1, -2, -3], [-1, 2, -3], [-1, 2, 3], [1, -2, 3]]

>>> list( itersolve([ [1,2], [-1,2], [1,-2], [-1,-2] ]) )
[]
```

If you are not used to working with Python, try to find time to practice a bit before attending my class. The most advanced concepts we are going to use are list comprehension and lambda expressions.