

CMP682 – Artificial Intelligence - Fall 2020

Homework 1

Goal:

Using theorem provers to solve constraint satisfaction problems.

Introduction:

The problem of determining whether there exists an interpretation that satisfies a given sentence in propositional logic is called as **Satisfiability Problem (SAT)**. The variables of a given boolean formula are consistently assigned to the values TRUE or FALSE, to see whether there is any assignment that makes the formula to evaluate to TRUE. In that case the formula is called satisfiable. If there is no possible assignments that makes the formula true, then it is unsatisfiable.

DPLL (Davis–Putnam–Logemann–Loveland) algorithm is a complete algorithm to decide whether a sentence in conjunctive normal form (CNF) is satisfiable. DPLL employs a systematic backtracking search procedure to explore the (exponentially sized) space of variable assignments looking for satisfying assignments

Although SAT is NP-complete, current SAT solvers are successful for many long formulas. They implement and optimize DPLL.

Satisfiability modulo theories (SMT) generalizes boolean satisfiability (SAT) where propositional variables are replaced with formulas of another mathematical theory. SMT can be thought of as a form of the constraint satisfaction problem and SMT solvers (also known as automated theory provers) enable wide range of applications. Many modern SMT solvers use DPLL(T) -which extends the original SAT-solving DPLL algorithm with the ability to reason about an arbitrary theory T - for determining the satisfiability of SMT problems.

There are many SMT solvers, such as Z3, YICES, and CVC4. Most of them have their own syntax, but they all accept the syntax of the **SMT- LIB** standard.

Z3 Theorem Prover is an efficient SMT Solver developed at Microsoft Research, and is used in various software verification and analysis applications. It is freely available.

Short Description:

In this homework you will utilize Z3 theorem prover to solve two problems: 3-SAT and 4-Queens. You will just specify the problems in a formal way using SMT-LIB standard, and then let the SMT solver Z3 to find the solution automatically for you.

This homework is inspired from the “Automated Reasoning: Satisfiability” course at Coursera thought by Hans Zantema (<https://www.coursera.org/learn/automated-reasoning-sat>)

Before starting the homework you should be familiar with SMT-LIB and Z3. The related materials can be found in the following links:

Tutorial on Programming with Z3:

<https://theory.stanford.edu/~nikolaj/programmingz3.html>

SMT-LIB (The Satisfiability Modulo Theories Library):

<http://smtlib.cs.uiowa.edu/>

<http://smtlib.github.io/jSMTLIB/SMTLIBTutorial.pdf>

Construction of a file in SMT-LIB2 format:

Important Note: Use **version 2.0** since it is commonly used and many resources can be found (<http://smtlib.cs.uiowa.edu/language.shtml>)

An SMT-LIB file includes the following main elements:

1) Logics:

```
set-logic <symbol>
```

Allows you to choose among a list of logics provided in

<http://smtlib.cs.uiowa.edu/logics.shtml>

```
ex: (set-logic QF-UF)
```

QF-UF should be sufficient for this homework

(There is also `set-options` but you will probably not need it)

2) Declarations:

```
declare-const
```

```
declare-fun
```

ex:

```
(declare-const p Bool)
```

```
(declare-fun f (Int) Int)
```

3) Formulas

`(assert ...)`

... corresponds to the actual formula

All formulas should be written in **prefix form**

Ex:

`(assert (and p (not p)))`

`(assert (= (- x y) (+ x (- y) 1)))`

4) Doing the actual SAT solving – checking satisfiability

`(check-sat)`

5) To show the satisfying assignment in case of satisfiability

`(get-model)`

6) `(exit)`

Examples:

(from <http://smtlib.cs.uiowa.edu/examples.shtml>)

```
; Basic Boolean example
(set-option :print-success false)
(set-logic QF_UF)
(declare-const p Bool)
(assert (and p (not p)))
(check-sat) ; returns 'unsat'
(exit)
```

(from <https://www.cs.upc.edu/~erodri/webpage/cps/lab/sat/tutorial-smt-solvers-slides/slides.pdf>)

```
(assert
  (and
    (or
      (<= (+ x 3) (* 2 u) )
      (>= (+ v 4) y)
      (>= (+ x y z) 2)
    )
    (= 7
      (+
        (ite (and (<= x 2) (<= 2 (+ x 3 (- 1))))) 3 0)
        (ite (and (<= u 2) (<= 2 (+ u 3 (- 1))))) 4 0)
      )
    )
  )
)
```

Here `ite` stands for `if-then-else`

It has three arguments: (1) condition which is a Boolean, (2) result in the case of condition being true, (3) result in the case of condition being false

`(ite a b c)`

is the expression of sort `S` equal to: `b` if `a` holds, `c` if `a` does not hold

Building and Running Z3:

Please refer to the following pages for building and running Z3

<https://github.com/Z3Prover/z3>

<https://github.com/Z3Prover/z3/wiki>

More on Z3 can be found at:

<https://rise4fun.com/z3/tutorial>

<https://www.cs.upc.edu/~erodri/webpage/cps/lab/sat/tutorial-smt-solvers-slides/slides.pdf>

Usage:

`Z3 [<options>] <input>`

Some options

`-smt2`: use parser for SMT-LIB 2 input format

`-h`: help, shows all options

We will use

`Z3 -smt2 file`

Where `file` is your file in SMT-LIB2 format

3- SAT Problem:

A propositional formula is composed of Boolean variables and operators (negation, disjunction, conjunction, implication, and biconditional). A propositional formula is defined as satisfiable (SAT) if it is possible to give values to the variables in such a way that the formula yields true. As a special case of SAT, formulas are required to be in conjunctive normal form (CNF).

Consider the formula

$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_1$

The formula is satisfiable, because if we choose $x_1 = \text{FALSE}$, and $x_2 = \text{FALSE}$,

For any x_3

$(\text{FALSE} \vee \neg \text{FALSE}) \wedge (\neg \text{FALSE} \vee \text{FALSE} \vee x_3) \wedge \neg \text{FALSE}$ is equivalent to

$(\text{FALSE} \vee \text{TRUE}) \wedge (\text{TRUE} \vee \text{FALSE} \vee x_3) \wedge \text{TRUE}$, and it is equivalent to $\text{TRUE} \wedge \text{TRUE} \wedge \text{TRUE}$ which is **TRUE**.

3-SAT is the problem of determining the satisfiability of a formula in CNF where each clause is limited to at most three literals (3-CNF). Note that unrestricted SAT problems can be transformed into 3-SAT.

Consider the Boolean expression in 3-SAT form

$(\text{A} \vee \text{B} \vee \text{C}) \wedge (\text{D} \vee \text{E} \vee \text{F})$

It has 2 clauses, and each clause contains 3 literals.

3-SAT tries to find whether there are such values of A-F that make the Boolean expression **TRUE**.

We can write the formula above in SMT-LIB2 format as

```
(declare-const A Bool)
(declare-const B Bool)
(declare-const C Bool)
(declare-const D Bool)
(declare-const E Bool)
(declare-const F Bool)
(assert
  (and
    (A or B or C)
    (D or E or F)
  ))
(check-sat)
(get-model)
```

Reducing Unrestricted SAT (USAT) into 3-SAT

We can reduce unrestricted SAT problem -which is converted into CNF- to 3-SAT problem., simply by transforming each clause $l_1 \vee \dots \vee l_n$ to a conjunction of $n - 2$ clauses as

$$\begin{aligned} & (l_1 \vee l_2 \vee x_2) \wedge \\ & (\neg x_2 \vee l_3 \vee x_3) \wedge \\ & (\neg x_3 \vee l_4 \vee x_4) \wedge \dots \wedge \\ & (\neg x_{n-3} \vee l_{n-2} \vee x_{n-2}) \wedge \\ & (\neg x_{n-2} \vee l_{n-1} \vee l_n) \end{aligned}$$

where x_2, \dots, x_{n-2} are newly introduced variables not occurring elsewhere.

4-Queen Problem:

In chess the queen may move horizontally, vertically or diagonally. Assume that, we have a chess board of size 4x4. Our goal in 4-Queen problem is to place 4 queens on the 4x4 board so that none of them will attack another. An example solution is given below.

	Q		
			Q
Q			
		Q	

We will now see, how we can formulaize this problem in propositional logic.

Every position on (i,j) will be represented as p_{ij} , corresponding to whether there is a queen on that position or not

We have therefore, 16 variables

p_{11} p_{12} p_{13} p_{14}
 p_{21} p_{22} p_{23} p_{24}
 p_{31} p_{32} p_{33} p_{34}
 p_{41} p_{42} p_{43} p_{44}

Note that, you can write each rule by specifying the exact indices. However, more generic rules will be better, so that you can generalize it to N-queens problem where N is very large. Let's see how we can write generic formulas. All of the below conditions should be satisfied.

We should have only a single queen on any row or any column.

But there should be at least one.

We can represent this for every row as

$p_{i1} \vee p_{i2} \vee p_{i3} \vee p_{i4}$

How can we represent at most one queen in a row?

If we take any two variables, both of them cannot be true

We can represent it as:

$\sim p_{ij} \vee \sim p_{ik}$ for all $j < k$

This is the same for columns

For every column:

$p_{1j} \vee p_{2j} \vee p_{3j} \vee p_{4j}$ should be true

and

$\sim p_{ij} \vee \sim p_{kj}$ should be true, for all $j < k$

On diagonals we only have the requirement to have at most one queen

If we consider the diagonals in the direction of $/$, then

Two variables p_{ij} and $p_{i'j'}$ where $i+j = i' + j'$ should not have a queen at the same time

If we consider the diagonals in the direction of \backslash , then

Two variables p_{ij} and $p_{i'j'}$ where $i-j = i' - j'$ should not have a queen at the same time

These should be true for every pair

So we write it as

For all i, j, i', j'

with (i,j) not equal to (i', j')

and satisfying

$i+j = i'+j'$

or

$i-j = i' - j'$

we should have

$\sim p_{ij} \vee \sim p_{i'j'}$

Requirements and Grading

In this homework you are required to complete the following tasks to get the corresponding points

1) (10 points)

Write 5 propositional formulas in CNF and find the satisfying values.

You should vary the number of clauses and number of variables.

The grading will be based on the complexity of the formulas.

2) (20 points)

Write 5 unrestricted Propositional formulas (by using all possible operators)

Convert them into CNF

Then, convert them into 3-SAT

Find the values satisfying the original and the final formula to show that they are equisatisfiable.

3) (20 points)

Solve the 4-Queens problem by writing the corresponding rules

4) (50 points)

Find another CSP problem, and formulate it as SAT. Then provide the solutions. Note that 20 points will be given based on the originality of the problem.

Submission

Send your homework in a zip file named as CMP682_Name_Surname.zip
Where Name and Surname is yours ☺

You should submit the following files

Your SMT-LIB2 files for each problem that you solved. Please give names which are easier to understand.

- The first task should be names as 3SAT_<id> where $1 \leq id \leq 5$
- The second task should be names as USAT_<id>-org and USAT_<id>-converted, where org is the unrestricted formula, and converted is the formula in 3-CNF, here $1 \leq id \leq 5$
- The third task should be named as 4-Queens
- The fourth task should be named as MyCSP.

For each of the SMT-LIB files above you should have also a solution file which will be named with the extension solution, by following the above requirements

Please send your homeworks to pinar@cs.hacettepe.edu.tr

In your header please include CMP682-HW1

Good luck

References:

Davis, Martin; Putnam, Hilary (1960). "A Computing Procedure for Quantification Theory". *Journal of the ACM*. **7** (3): 201–215. doi:10.1145/321033.321034

Davis, Martin; Logemann, George; Loveland, Donald (1962). "A Machine Program for Theorem Proving". *Communications of the ACM*. **5** (7): 394–397. doi:10.1145/368273.368557

Z3: An efficient SMT solver, L De Moura, N Bjørner - International conference on Tools and Algorithms, 2008

https://en.wikipedia.org/wiki/Boolean_satisfiability_problem

https://en.wikipedia.org/wiki/DPLL_algorithm

https://en.wikipedia.org/wiki/Satisfiability_modulo_theories

[https://en.wikipedia.org/wiki/DPLL\(T\)](https://en.wikipedia.org/wiki/DPLL(T))

https://en.wikipedia.org/wiki/Z3_Theorem_Prover

<https://www.coursera.org/learn/automated-reasoning-sat>

Helpful resources

<http://cs.tau.ac.il/~msagiv/courses/sp/lecture-sat.pdf>

<http://www.dis.uniroma1.it/~liberato/ar/dpll/dpll.html>

<https://courses.cs.washington.edu/courses/cse507/14au/slides/L2.pdf>