CMP682 –Artificial Intelligence – Fall 2021

Homework 2

Goal:

Solving constraint satisfaction problems.

Theorem Provers to solve CSP:

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 satsfiability 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.

Use of Z3 and SMT-LIB

In this homework, you will utilize Z3 theorem prover to solve CSP. 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 paerser 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 s special case of SAT, formulas are required to be in conjunctive normal form (CNF).

Consider the formula

$(x_1 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_3) \land \neg x_1$

The formula is satisfiable, because if we choose x_1 = FALSE, and x_2 = FALSE,

For any x_3

(FALSE ∨ ¬FALSE) ∧ (¬FALSE ∨ FALSE ∨ x_3) ∧ ¬FALSE is equivalent to

(FALSE ∨ TRUE) ∧ (TRUE ∨ FALSE ∨ x_3) ∧ TRUE, and it is equivalent to

TRUE ∧ TRUE ∧ 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

(AVBVC) ∧ (DVEVF)

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

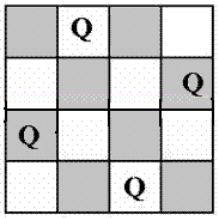
We can reduce unrestricted SAT problem -which is converted into CNF- to 3-SAT problem., simply by transforming each clause $l_1 \lor \cdots \lor l_n$ to a conjunction of $n - 2$ clauses as

$$(l_1 \lor l_2 \lor x_2) \land$$
$$(\neg x_2 \lor l_3 \lor x_3) \land$$
$$(\neg x_3 \lor l_4 \lor x_4) \land \cdots \land$$
$$(\neg x_{n-3} \lor l_{n-2} \lor x_{n-2}) \land$$
$$(\neg x_{n-2} \lor l_{n-1} \lor l_n)$$

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

4-Queen Problem:

In chess the quenn may move horizontally. vertically or diagonally. Assume that, we have a chess borad of size 4x4. Our goal in 4-Queen problem is to place 4 queens on the 4x4 board so that non of them will attack another. An example solution is given below.



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

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

We have therefore, 16 variables

p11 p12 p13 p14
p21 p22 p23 p24
p31 p32 p33 p34
p41 p42 p43 p44

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 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
pi1 V pi2 V pi3 V pi4

How can we represent at most one queen in a row?
If we take any two varaibles, both of them cannot be true
We can represent it as:
~pij V ~pik for all j<k

This is the same for columns
For every column:
p1j V p2j V p3j V p4j should be true
and
~pij V ~pkj 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 pij and pi'j' where i+j = i' + j' should not have a queen at the same time
If we consider the diagonals in the direction of \ , then
Two varibles pij and pi'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
~pij V ~pi'j'

Task 1: (10 pts) 3-SAT with Z3
In the first part of this homework, you are required to solve 3-SAT problem using theorem provers as described above
Requirements : Write a propositinal formula in CNF and find the satisfying values.

Task 2: (20 pts) Unrestricted
In the second part, you are required to show that unrestricted formulas can be converted into restricted forms, and solution for both is the same.
Requirements:
Write an unrestricted Propositional formula (by using all possible operators)
Convert it into CNF and then into 3-SAT (by hand)
Find the values satisfying the original and the final formula with Z3 to show that they are equisatisfiable.

Task 3: (30 pts) N-Queens Problem
In the third part, you are required to formulate N-Queens problem with Z3.
Requirements
Solve the 4-Queens problem by writing the corresponding rules
Generalize it to N-Queens problem.

Task 4: (40 pts) Layout Problem
Your last task is to place N rectangular blocsks with different sizes into a grid, by formulating it as a CSP propblem.
The blocks should not overlap with each other, they can touch on the edges, and there can be empty places.
Requirements
Place 4 rectangular blocks of size 2x2 into a 8x8 grid
Vary the number of blocks, the sizes of the blocks, and the size of the grid.

Submission

Submit your homework to gradescope in 4 different parts

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
-The second task should be names as USAT-org and USAT-converted, where org is the unrestricted formula, and converted is the formula in 3-CNF

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

Similarly, submit your code for Task3, as 4-Queens and N-Queens, as well as inputs and outputs for your formulas

For the final part, together with your codes and input/outputs, also submit a report describing your approach and solution strategy.

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/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