CIS 552: Advanced Programming

- Home
- Schedule
- Resources
- Style guide
- Syllabus

SAT Solving with the DPLL algorithm

The Davis-Putnam-Logemann-Loveland algorithm is an algorithm for deciding the satisfiablility of propositional logic formulae. Although the SAT problem is NP-complete, it is still remarkably amenable to automation, and modern high-performance SAT-solvers are regularly used in software verification, constraint solving and optimization.

Your task for this problem will be to implement the DPLL algorithm and verify its correctness using QuickCheck.

```
{-# OPTIONS -Wall -fwarn-tabs #-}
module Sat where

import Data.Map (Map)
import qualified Data.Map as Map


import Test.QuickCheck
import Test.HUnit
```

The DPLL algorithm works on formulae that are in Conjunctive Normal Form, i.e. formula that are a conjunction of clauses, where each clause is a disjunction of literals, i.e. positive or negative propositional variables.

```
-- | An expression in CNF, the conjunction of clauses
newtype CNF = CNF [ Clause ] deriving (Eq, Ord, Show)
unCNF :: CNF -> [ Clause ]
unCNF (CNF cs) = cs

-- | A clause -- the disjunction of a number of literals
type Clause = [ Lit ]

-- | A literal, either a positive or negative variable
type Lit    = Int

-- | invert the polarity of a literal
invert :: Lit -> Lit
invert a = negate a
```

For example, we can represent the formula `A ^ (B v ~A)` by numbering the propositional variables:

```
example_formula :: CNF
example_formula = CNF [[1],[2,-1]]
```

This formula is satisfiable as long as `A` or (AKA literal 1) is assigned to `True`.

```
example_assignment :: Map Lit Bool
example_assignment = Map.fromList [(1, True), (2, True)]
```

Note that not every map from literals to booleans is valid. If `(1,True)` is in the map, then `(-1, True)` had better not be.

```haskell
valid :: Map Lit Bool  -> Bool
valid m = Map.foldrWithKey (\ lit b1 val ->
                              case Map.lookup (invert lit) m of
                                Just b2 -> b1 == not b2 && val
                                Nothing -> True && val) True m

test :: Test
test =  valid
   (Map.fromList ( [(-2,True),(-1,True),(2,True)])) ~?= False
```

Given a valid assignment for literals, we can determine the truth value of the formula.

```haskell
interp :: (Map Lit Bool) -> CNF -> Bool
interp m (CNF f) = all (any (findLit m)) f where
  findLit m' k =
     case (Map.lookup k m', Map.lookup (invert k) m') of
          (Just b1, Just b2) | b1 == not b2 -> b1
          (Just _, Just  _)  -> error "invalid map"
          (Just b, Nothing)  -> b
          (Nothing, Just b)  -> not b
          (Nothing, Nothing) -> True
```

The DPLL algorithm takes a formula in conjunctive normal form and produces an assignment for the variables if the formula is satisfiable. Your job is to complete this implementation based on the description of the algorithm on the Wikipedia page. Note that the pseudocode given is fairly imperative. You should think about how to define a *functional* version of the algorithm, based on what we have learned in this course.

```haskell
dpll :: CNF -> Maybe (Map Lit Bool)
dpll = undefined
```

As a hint, you might think about some of the steps of the algorithm individually. In particular, consider the following functions.

```haskell
-- | Given a list of literals, create the trivial assignment
-- that satisfies that list (if one exists).
satisfy :: Clause -> Maybe (Map Lit Bool)
satisfy = undefined

-- | If a propositional variable occurs with only one polarity in the
-- formula, it is called pure. Pure literals can always be assigned in
-- a way that makes all clauses containing them true. Thus, these
-- clauses do not constrain the search anymore and can be deleted.
-- This function collects all pure literals from the formula and
-- returns the assignment paired with the refactored formula
-- that reflects that assignment.
pureLitAssign :: CNF -> (Map Lit Bool, CNF)
pureLitAssign = undefined

-- | If a clause is a unit clause, i.e. it contains only a single
-- unassigned literal, this clause can only be satisfied by assigning
-- the necessary value to make this literal true. This function collects
-- all unit clauses from the formula and returns the assignment paired
-- with the refactored formula that reflects that assignment.
unitPropagate :: CNF -> (Map Lit Bool, CNF)
unitPropagate = undefined
```

You will know that you have finished the assignment when the following property is satisfied by quick check. Note: You may wish to instrument this property (using collect/classify) to make sure that you are testing your solution with meaningful tests.

```
prop_dpll :: CNF -> Property
prop_dpll c =

  case dpll c of
    Just m -> if valid m then
       (property (interp m c))
      else property False
    Nothing ->  (property True)
```

News :

Welcome to CIS 552!
See the home page for basic information about the course, the schedule for the lecture notes and assignments, the resources for links to the required software and online references, and the syllabus for detailed information about the course policies.

Links :

- Piazza
- Haskell.org
- GHC manual
- Library documentation
- Hackage