Unit Propagation (Scheme+Haskell - 7+7 Points)

June 8, 2022

DPPL algorithm is the core of SAT solvers nowadays. It takes a propositional formula in CNF (conjunctive normal form) and returns true if, and only if, the formula is satisfiable. A formula in CNF is usually represented as a set of clauses. A *clause* is represented as a set of literals. A *literal* is either a propositional variable (e.g. x) or its negation (e.g. $\neg x$). For instance, a formula

$$\varphi = (a \vee b \vee \neg c \vee \neg f) \wedge (b \vee c) \wedge (\neg b \vee e) \wedge \neg b \tag{1}$$

is represented as

$$\varphi = \{\{a, b, \neg c, \neg f\}, \{b, c\}, \{\neg b, e\}, \{\neg b\}\}. \tag{2}$$

One of the subroutines of the DPLL algorithm is the unit propagation simplifying the input formula. A *unit* is a clause containing a single literal, e.g. $\{\neg b\}$. It is obvious that a satisficing evalution for a formula in CNF must evaluate all units to true. This allows simplifying the input formula. Assume that a set of clauses $\varphi = \{c_1, \ldots, c_n\}$ has a unit, i.e., $c_k = \{u\}$ for some k and literal u, then φ can be simplified by the following rules:

1. if $u \in c_i$, then c_i can be removed from φ,
2. if $\neg u \in c_i$, then $\neg u$ can be removed from c_i.

For example, the formula φ in (2) has a unit $\{\neg b\}$, so we can simplify to $\{\{a, \neg c, \neg f\}, \{c\}\}$. Note that by propagating the unit, a new unit was created. Thus we can continue and propagate the unit $\{c\}$ obtaining $\{\{a, \neg f\}\}$. The resulting set of clauses has no unit.
Your task is to implement the unit propagation for a given formula φ in CNF, i.e., eliminate all possible unit clauses. See the following pseudocode.

```
while there is a unit clause {u} in φ do
      φ <- unit-propagate(u, φ);
```

Task 3 - Scheme

In Scheme, implement a function `(propagate-units cls)` that accepts a list of clauses and returns a list of clauses after the unit propagation. A clause is represented as a list of literals. Positive and negative literal is represented, respectively by the following structures:

```
(struct pos (variable) #:transparent)
(struct neg (variable) #:transparent)
```

As the resulting list of clauses returned by `propagate-units` should represent a set, remove all the duplicated clauses from the list.
Your task is to be called `task3.rkt` and must provide the `propagate-units` and both structures `pos` and `neg`. Hence, the head of your file should start with

```
#lang racket
(provide propagate-units (struct-out pos) (struct-out neg))

(struct pos (variable) #:transparent)
(struct neg (variable) #:transparent)

; your code here
```

Hint

To remove an element `v` from a list `lst`, you may want to use the function `(remove v lst)`. To remove duplicated elements from a list `lst`, call the function `(remove-duplicates lst)`.

Examples

The following shows the behaviour of the `propagate-units` function.
For $\varphi = \{\{\neg x\}\}$ we get

```
> (propagate-units (list (list (neg "x"))))
'()
```

For $\varphi = \{\{x\}, \{\neg x\}, \{y\}, \{\neg y\}\}$ we get

```
> (propagate-units (list (list (pos "x")) (list (neg "x")) (list (pos "y")) (list (neg "y"))))
'(())
```

For $\varphi = \{\{a, b, \neg c, \neg f\}, \{b, c\}, \{\neg b, e\}, \{\neg b\}\}$, we get

```
> (propagate-units (list (list (pos "a") (pos "b") (neg "c") (neg "f"))
                         (list (pos "b") (pos "c"))
                         (list (neg "b") (pos "e"))
                         (list (neg "b"))))
(list (list (pos "a") (neg "f")))
```

Task 4 - Haskell

In Haskell, implement a function `propagateUnits :: [Clause] -> [Clause]` that accepts a list of clauses and returns a list of clauses after the unit propagation. As the resulting list of clauses returned by `propagateUnits` should represent a set, remove all the duplicated clauses from the list. Literals and clauses are represented as follows:

```
type Variable = String
data Literal = Neg { variable :: Variable }
             | Pos { variable :: Variable } deriving (Eq, Ord)

type Clause = [Literal]
```

and for your convenience, you are provided with the instance of `Show` for literals:

```
instance Show Literal where
  show (Neg x) = "-" ++ x
  show (Pos x) = x
```

Your task is to be called `Task4.rkt` and must export the `propagateUnits` function and the `Literal` data type. Hence, the head of your file should read

```
module Task4 ( propagateUnits, Literal (..) ) where
import Data.List -- for delete, nub functions
```

Hint

To remove an element from a list, you can use the function `delete :: Eq a => a -> [a] -> [a]`. To remove duplicated elements from a list, call the function `nub :: Eq a => [a] -> [a]`. Both functions are located in the module `Data.List`

Examples

The following shows the behaviour of the `propagateUnits` function.
For $\varphi = \{\{\neg x\}\}$ we get

```
> propagateUnits [[Neg "x"]]
[]
```

For $\varphi = \{\{x\}, \{\neg x\}, \{y\}, \{\neg y\}\}$ we get

```
> propagateUnits [[Pos "x"], [Neg "x"], [Pos "y"], [Neg "y"]]
[[]]
```

For $\varphi = \{\{a, b, \neg c, \neg f\}, \{b, c\}, \{\neg b, e\}, \{\neg b\}\}$, we get

```
> propagateUnits [[Pos "a", Pos "b", Neg "c", Neg "f"], [Pos "b", Pos "c"], [Neg "b", Pos "e"], [Neg "b"]]
[[a,-f]]
```