The logic reduction rules of Davis-Putnam

Terminology

~ is negation (NOT)
+ is disjunction (OR)
* is conjunction (AND)
F is the CNF formula we want to satisfy - it is a conjunction of clauses.
A clause is a disjunction of literals.
A literal is either some atom *p* or its negation *~p*

Rule I: Elimination of one-literal clauses

(a) If *F* contains the one-literal clause *p* and also the one-literal clause *~p*, then *F* is 0 (UNSAT)
(b) If (a) doesn't apply, and if the literal *p* appears as a one-literal clause in *F*, delete all clauses that contain the literal *p* and delete all the literals *~p* from other clauses
(c) If (a) doesn't apply, and if the literal *~p* appears as a one-literal clause in *F*, delete all clauses that contain the literal *~p* and delete all the literals *p* from other clauses

Note: if after applying (b) or (c), *F* becomes empty, it is SAT

Rule II: Affirmative-Negative rule

If the atom *p* appears in *F* only affirmatively (always *p*, never *~p*) or if *p* appears only negatively (always *~p*, never *p*), then all clauses that contain *p* or *~p* may be deleted. If the resulting formula is empty, *F* is SAT.

Rule III: Rule for eliminating atomic formulas

Note: This procedure is popularly called **resolution**.

Let some given formula be of the form *(A + p) * (B + ~p) * C* , where *A*, *B* and *C* are free of *p*. Then the formula can be reduced to *(A + B) * C*

This can be done on parts of the big formula. For example, the following *F* :
*(a + b + ~c) * (c + ~r + ~h + k) * (t + ~s) * (r + u)*

By resolution of the first two clauses, can be reduced to:
*(a + b + ~r + ~h + k) * (t + ~s) * (r + u)*

Note: *F* can now be further reduced by resolution of clauses 1 and 3

The DPLL procedure

The idea: using some form of the logic rules stated above, a recursive procedure is given to find a satisfying variables assignment for a CNF formula. The procedure is exponential in time (unavoidable – SAT is NP complete), but polynomial in memory.

First, the **DPLL_reduce** procedure is presented. Its aim is to get rid of an assigned literal from a formula.

Algorithm: **DPLL_reduce**

Inputs: formula *F*; literal *L*

Output: formula *F'* with the literal *L* reduced

Operation: Uses resolution to reduce *L* from *F*. This is exactly Rule I of Davis-Putnam, assuming that a single-literal clause (*L*) is a part of the formula. Thus:

- Clauses containing *L* are removed
- All instances of ~*L* in other clauses are removed

Pseudo-code:

```
DPLL_reduce (F, L)
     F' = {}
     foreach clause C in F do
          if C contains L
               skip
          elsif C contains ~L
               C' = C with ~L deleted
               add C' to F'
          else
               add C to F'

     return F'
```

The main algorithm, **DPLL_satisfy**, relies on **DPLL_reduce** to shorten the formula during its recursion step.

Algorithm: **DPLL_satisfy**

Inputs: formula *F*

Output: TRUE if *F* is SAT, FALSE if *F* is UNSAT

Operation: Given a formula, first tries to propogate boolean constraints – forced resolutions, using the Davis-Putnam rules I (to reduce one-literal

clauses) and II (to reduce "pure" literals). When this is no longer possible, uses a heuristic to pick a variable to split, and recursively tries to satisfy both choices (TRUE and FALSE for this variable), in a DFS manner.
Note: "pure" literals are literals that appear only affirmatively or only negatively in the (rule II)

<u>Pseudo-code:</u>

```
DPLL_satisfy (F)
     if F is empty
          return TRUE
     if F contains an empty clause
          return FALSE
     if exists a pure literal L in F
          return DPLL_satisfy(DPLL_reduce(F, L))
     if exists a unit clause {L} in F
          return DPLL_satisfy(DPLL_reduce(F, L))

     v = pick_variable(F)

     return
          DPLL_satisfy(DPLL_reduce(F, L)) or
          DPLL_satisfy(DPLL_reduce(F, ~L))
```

Eli Bendersky, June 2004