```
anNF (p@(LetterP _)) = p
anNF (NotP p) = NotP (anNF p)
anNF (AndP p q) = AndP (anNF p) (anNF q)
anNF (OrP p q) = NotP (AndP (NotP (anNF p)) (NotP (anNF q)))
anNF (ImpliesP p q) = NotP (AndP (anNF p) (NotP (anNF q)))
anNF _ = error "anNF"
```

Figure 6: And-Negation-Normal Form

that SAT problem would solve the acceptance problem for the non-deterministic Turing machine. This theorem, now known as the Cook-Levin theorem, is a very direct reduction showing that SAT is complete for $NP$. That means that if you can solve SAT in $P$, then $P = NP$.

After 40 years of intense effort we have not found a polynomial time algorithm for SAT (although this week one has been claimed to exist). We have also not been able to prove a non-polynomial lower bound.

With an efficient complete solver highly unlikely, we must look at incomplete solvers if we want to be fast. The past decade has seen tremendous progress on SAT solving algorithms that work well in practice. All of these algorithms are either incomplete (meaning they don't solve all instances of the problem) or are very expensive in some cases.

## 6.1 Simple, Incomplete SAT solvers

Huth and Ryan sketch a couple of simple algorithms that are inspired by ideas in Stalmarck's method, first published in 1990. The algorithms are called linear and cubic; the names describe achievable complexity of the implementations. We will discuss Haskell implementations of these algorithms. The Haskell implementation may not achieve these bounds exactly, but the bounds are achievable.

### 6.1.1 Conceptual Description

Both algorithms use the same representation of propositions as a directed acyclic graph (dag). The dag maximizes sharing, achieving a compact representation and giving an efficient structure for propagating information. It has the very nice property that every any consistent labeling of the graph with True and False will correspond to a satisfying assignment.

The dag representation is based on the adequate set of operators $\wedge$ and $\neg$, so the first step is rewriting the formula into and-negation-normal-form. In this normal form we only have proposition letters, and and not. Simple code to do this conversion is given in Figure 6. This code does not deal with constants for true and false.

For example, consider the formula $p \wedge \neg(q \vee \neg p)$, which is Huth and Ryan example 1.48. We can calculate its normal form as follows:

```
*SimpleSetExamples> ex1_48
```

```
p /\ ~(q \/ ~p)
*SimpleSetExamples> anNF it
p /\ ~~(~q /\ ~~p)
```

The conversion to a dag will yield a graph with the full term represented by the dag's unique source (in-degree 0) and the proposition letters exactly the set of sinks (out-degree 0). The dag for the example is:

```
8     0 /\ 7
7     ~ 6          8
6     ~ 5          7
5     2 /\ 4       6
4     ~ 3          5
3     ~ 0          4
2     ~ 1          5
1     q            2
0     p            8,3
```

In this representation the left hand column is node number. The center column shows the operator labeling the node and the node numbers of its children. The right hand column is the set of parents of the node.

The algorithm solves the sat problem by building a truth assignment for every node in the graph. This truth assignment is called a *marking*. If a complete, consistent marking can be constructed the algorithm terminates with success, declaring the formula satisfiable. The linear algorithm makes only logically necessary markings "forced" by marking the root as True. If while making a necessary marking the algorithm attempts to mark a True node as False or a False node as True the algorithm terminates declaring the formula unsatisfiable. If the algorithm fails to construct a complete assignment then it terminates declaring the formula undetermined. It is because the third outcome is a possibility that this is called an incomplete method. It does not solve all instances of SAT.

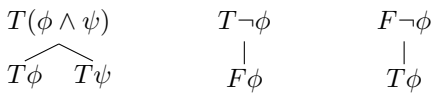
The marking algorithm begins at the root, which it marks True. When an $\wedge$ node is marked True, its children will be marked True as well. In the example, the root (8) is marked True. This propagates to 0, which is the proposition letter $p$, and to 7, which is a negation. When a negation node is marked, the complement of the mark is passed to its child. This marks 6 as False. Since 6 is a negation, its child is marked True. Thus 5 is marked True. Since 5 is an $\wedge$ node, its children are marked True. The remaining nodes are all negations, which force their children. The final assignment and an animation of its construction is given here:

```
State = Satisfiable
8     0 /\ 7                    TT          +
7     ~ 6          8            T  T        +
6     ~ 5          7            F   F       +
5     2 /\ 4       6            T    T      +
4     ~ 3          5            T       T   +
```

```
3     ~ 0          4              F         F e+
2     ~ 1          5              T     T     +
1     q            2              F      F    +
0     p            8,3            T T        = +
```

In this presentation of the algorithm, the first three major columns present the dag, and the fourth presents the animation. Within the animation, the first column is the final assignment constructed. After that, moving left to right, we see how the algorithm proceeded. In the tenth step the `=` indicates that a redundant marking of node 0 was called for. The eleventh step, indicated with an `e`, revisited node 3 because its child was marked. Finally the `+` mark indicates that a verification of the complete marking completed successfully.

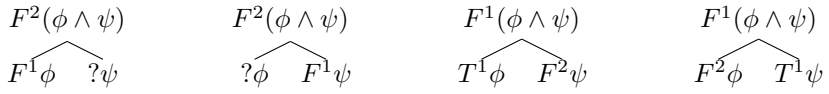
In this example we have basically used three rules:




In the example we applied these rules "top to bottom". This kind of propagation is sometimes called forcing, since marking the upper node forces (functionally determines) the mark of the lower node.

The rules can also be read "bottom to top." That is, if the children of an $\wedge$ node are both marked true that $\wedge$ node can be marked True. Similarly markings can propagate up through a $\neg$ node.

Other relationships that can be exploited include the following:




Here markings superscripted with 1 force those markings superscripted with 2. The algorithm attempts to propagate information by these rules whenever a child is marked.

For example, consider the formula $p \wedge \neg(p \wedge q)$.

```
State = Satisfiable
4     0 /\ 3                      TT       +
3     ~ 2          4              T  T     +
2     0 /\ 1       3              F   Fe e +
1     q            2              F     F =+
0     p            4,2            T T      +
```

The "top down" forcing marks nodes 4, 3, 2 and 0. But since node 2 is an $\wedge$ marked with False it does not force node 1. However, since node 0 was marked False, nodes 0 and 2 together force node 1 to False, completing the marking of the graph. In the animations the `e` symbol indicates that a node is being explored to see if is participating in a non-top-down forcing relationship. In

this case the first exploration marks 1 False. The second exploration, which was triggered because node 2 is parent of node 0, which was marked in step 2, makes a redundant marking of node 1.

Of course, one possible outcome of the algorithm is to discover that a proposition is not satisfiable. The simplest such proposition is $p \wedge \neg p$. It's animation is:

```
State = Unsatisfiable Exploring 1
2     0 /\ 1                    TT
1     ~ 0         2             T  T
0     p           2,1           T T X
```

This is marked unsatisfiable. The symbol `X` in the animation shows the step at which the contradiction was discovered. In particular, after marking node 1 True, it was attempting to mark node 0 False, but it was previously marked True.

While the linear algorithm does very well with $\wedge$ nodes marked True, it does very little with $\wedge$ nodes marked False. In particular, it fails on the very simple example $\neg(p \wedge q)$, which is satisfiable by any valuation in which one of $p$ or $q$ is marked False:

```
State = Unknown
3     ~ 2                       TT  -
2     0 /\ 1      3             F Fe-
1     q           2                 -
0     p           2                 -
```

Note here the outcome is unknown. The symbol `-` in the animation indicates that the test of the completeness of the markings failed. (The test actually tests both completeness and consistency of the markings; in this case the marking is incomplete.)

One reason the linear algorithm has problems with this example, which essentially encodes an $\vee$ operation, is that it only assigns markings that are required; it never guesses or backtracks. Contrast this with the complete tautology checker you implemented in earlier assignments. That did complete search of all possible assignments. Also consider the tableau proof technique which would branch the path to explore disjunctions. Both of these complete algorithms are potentially exponential, partially because they can layer guess upon guess and may explore an exponential number of alternatives.

The cubic algorithm extends the linear algorithm by allowing some speculation, but it only speculates on one graph label at a time. It never explores multiple dependent guesses. Speculation is managed by distinguishing between *permanent marks*, to which the algorithm is fully committed, and *temporary marks*, that are dependent upon the speculative mark (the guess). In the $\neg(p \wedge q)$ example, the cubic algorithm explores what happens if node 0 is marked True. In that case, since node 2 is marked False, this forces node 1 to be marked False. It then tests this marking, and discovers that this satisfies all nodes, confirming that the graph is satisfiable. This is animated:

```
State = Satisfiable
3     ~ 2                        TT  -     +
2     0 /\ 1      3              F Fe-   e +
1     q           2              F   -w   F+
0     p           2              T   -wtT  +
```

In this animation we see that after the failed test, the two unmarked nodes, 1 and 0, are placed on a worklist (note symbol `w`). Node 0 is then speculatively marked True (note symbol `t`). Since its child has been marked, node 2 is explored, causing node 1 to be marked. This marking is complete and consistent, indicated by the symbol `+`. Whenever a complete and consistent marking is created, even if it involves speculation, the algorithm terminates with the outcome satisfiable.

If marking node 0 True had not yielded a satisfying assignment the algorithm would have explored marking node 0 False instead. To do this it would roll-back the temporary marks and start again with a new temporary marking. If both speculative marks yield non-contradictory and inconclusive markings, then the intersection of the two markings are adopted as permanent marks. For example, consider the animation of the partial run of the algorithm on $\neg((q \wedge \neg q) \wedge r \wedge (p \wedge \neg p)$:

```
State = SpeculateTrue
9     ~ 8                        TT  -          -         -
8     2 /\ 7      9              F Fe-         e-    e    -
7     3 /\ 6      8                  -w         -         -
6     4 /\ 5      7                  -w         -         -
5     ~ 4         6                  -w         -         -
4     p           6,5                -w         -         -
3     r           7                  -w         -         -
2     0 /\ 1      8              F   -w  e  eF -  eF   e-!F
1     ~ 0         2                  -w   eF   -     eT -
0     q           2,1                -wtT      -fF      -m
```

Here we see that setting node 0 to be True forced nodes 1 and 2 to be False. Similarly, setting node 0 to be False forced node 2 to be False and node 1 to be true. Going forward, the algorithm does not commit node 0 to have any mark, but it does commit node 2 to be marked False. The symbol `m` marks the node forcing the merge assignment. The symbol `!` flags the nodes marked by merging the two markings.

Later on in the execution of this example, the cubic algorithm explores the consequences of setting node 4 to be True and False. Again both outcomes are inconclusive, but they agree on the markings of nodes 6 and 7:

```
9     ~ 8                        T           -          -
8     2 /\ 7      9              F          e-      e   -
7     3 /\ 6      8              F        eF -    eF    -!F
6     4 /\ 5      7              F  e  eF   -  eF     e-!F
5     ~ 4         6                  eF     -       eT -
```

```
4     p           6,5              tT        -fF        -m
3     r           7                          -          -
2     0 /\ 1      8               F          -          -
1     ~ 0         2                          -          -
0     q           2,1                        -          -
```

Ultimately the cubic algorithm fails to find a marking for this graph, even though it is true under all truth assignments.

If one of the speculative markings yields a contradiction then its complement mark is added to the permanent marking. This can be seen in the animation of Huth and Ryan's unsatisfiable equation (1.11):

$$(p \vee q \vee r) \wedge (p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p) \wedge (\neg p \vee \neg q \vee \neg r)$$

The initial phase of the linear algorithm leaves several nodes unmarked:

```
28    10 /\ 27                    TT                  -
27    13 /\ 26    28              T   T               -
26    16 /\ 25    27              T      T            -
25    19 /\ 24    26              T         T         -
24    ~ 23        25              T            T      -
23    17 /\ 22    24              F             Fe    -
22    ~ 21        23                                  -w
21    ~ 20        22                                  -w
20    11 /\ 14    21                                  -w
19    ~ 18        25              T           T       -
18    5 /\ 17     19              F            F   e  -
17    ~ 1         23,18                               -w
16    ~ 15        26              T       T           -
15    3 /\ 14     16              F        F       e  -
14    ~ 5         20,15                               -w
13    ~ 12        27              T    T              -
12    1 /\ 11     13              F     F           e -
11    ~ 3         20,12                               -w
10    ~ 9         28              T T                 -
9     1 /\ 8      10              F  F               e-
8     ~ 7         9                                   -w
7     ~ 6         8                                   -w
6     3 /\ 5      7                                   -w
5     ~ 4         18,14,6                             -w
4     r           5                                   -w
3     ~ 2         15,11,6                             -w
2     q           3                                   -w
1     ~ 0         17,12,9                             -w
0     p           1                                   -w
```

The cubic algorithm proceeds, speculating that $p$ (node 0) may be True. This eventually yields a contradiction at node 14. The algorithm then concludes that $p$ must be False:

```
28    10 /\ 27                T-
27    13 /\ 26    28          T-
26    16 /\ 25    27          T-
25    19 /\ 24    26          T-
24    ~ 23        25          T-
23    17 /\ 22    24          F-       e
22    ~ 21        23           -w       F
21    ~ 20        22           -w        T
20    11 /\ 14    21           -w         Fe     e
19    ~ 18        25          T-
18    5 /\ 17     19          F-            e
17    ~ 1         23,18        -w    eT
16    ~ 15        26          T-
15    3 /\ 14     16          F-                    e
14    ~ 5         20,15        -w              eT     X
13    ~ 12        27          T-
12    1 /\ 11     13          F-
11    ~ 3         20,12        -w                F
10    ~ 9         28          T-
9     1 /\ 8      10          F-
8     ~ 7         9            -w
7     ~ 6         8            -w
6     3 /\ 5      7            -w
5     ~ 4         18,14,6      -w            F
4     r           5            -w             T
3     ~ 2         15,11,6      -w                  T
2     q           3            -w                   F
1     ~ 0         17,12,9      -w  eF
0     p           1           F-wtT                    0F
```

Unfortunately, this also leads to a contradiction. It forces a contradictory marking of node 8:

```
28    10 /\ 27                T
27    13 /\ 26    28          T
26    16 /\ 25    27          T
25    19 /\ 24    26          T
24    ~ 23        25          T
23    17 /\ 22    24          F      e
22    ~ 21        23
21    ~ 20        22
20    11 /\ 14    21
19    ~ 18        25          T
18    5 /\ 17     19          F       e        e
17    ~ 1         23,18       F    eF           =
16    ~ 15        26          T
15    3 /\ 14     16          F            e
```

```
14    ~ 5          20,15          F             F
13    ~ 12         27             T
12    1 /\ 11      13             F        e
11    ~ 3          20,12          F         F
10    ~ 9          28             T
9     1 /\ 8       10             F                      e
8     ~ 7          9              T                    eT X
7     ~ 6          8              F                  eF
6     3 /\ 5       7              T                eT
5     ~ 4          18,14,6        T              T
4     r            5              F               F
3     ~ 2          15,11,6        T          T
2     q            3              F           F
1     ~ 0          17,12,9        T  eT
0     p            1              F0F
```

Having obtained contradictions for both possible markings of node 0, the algorithm concludes the proposition is unsatisfiable.

### 6.1.2 Discussion of complexity

Why is the linear algorithm linear? The following facts all contribute:

1. A mark can be assigned in constant time.
2. A propagating exploration operation can be performed in constant time.
3. Since the maximum out-degree of the dag is 2, the edge set of the graph is linear in the number of nodes.
4. Each edge will be traversed at most once to perform a "top-down" marking.
5. Each edge will be traversed at most once to trigger a propagating exploration.
6. The test of consistency and completeness of the marking can be performed in linear time and is performed exactly once.

What about the cubic algorithm? In a nutshell, the cubic algorithm as described invokes the linear algorithm twice per node in the graph. That suggests it should be the quadratic algorithm! This quadratic algorithm is sound, but it turns out to be very sensitive to the order of the worklist! Since information accumulates in the markings, it is possible for there to be dependencies between markings. There may be some critical sequence necessary to unlock all of the information necessary to discover a marking. However if we run the quadratic algorithm once for every (unmarked) node, we obtain a cubic algorithm that is insensitive to the order of the worklist.