Programming in Propositional Logic or Reductions: Back to the Roots (Satisfiability)

Hermann Stamm-Wilbrandt
Institut für Informatik III
Universität Bonn
hermann@holmium.informatik.uni-bonn.de

Abstract

In this paper, NP-complete and polynomial solvable problems are reduced to the SATISFIABILITY problem. We call this process "programming" in propositional logic. On the one hand, the programs (propositional formulas) derived by this process build a rich pool of easy and hard (non-random) formulas for SATISFIABILITY-solving heuristics. On the other hand, the implementations (programs) give rise to new heuristics for solving SATISFIABILITY.

Contents

Chapter 1

Introduction

In this paper we will consider the SATISFIABILITY problem, which is the question whether or not a propositional formula in conjunctive normal form is satisfiable. A propositional formula consists of boolean variables and the connectives $\wedge$ (conjunction), $\vee$ (disjunction) and $\neg$ (negation). It is satisfiable if, and only if, there is an assignment of values *true* and *false* to the variables such that the whole formula becomes true. A formula is said to be in *conjunctive normal form* (CNF) if, and only if, it is a conjunction of clauses. A *clause* is a disjunction of literals, and a *literal* is a variable or a negation of a variable.

SATISFIABILITY (SAT) was the first [2] problem to be shown NP-complete [6]. Its NP-completeness forces us to consider special cases of it or to look for good heuristics, given that P$\neq$NP [6]. NP-complete restrictions of SATISFIABILITY are 3-SAT (SAT with clauses of at most 3 literals) [6], SAT(3) (SAT with each literal occurring at most 3 times in the formula) [5] and PLANAR SATISFIABILITY (SAT with planar clause-graph (see chapter 6.2.1)) [10]. Polynomial solvable restrictions of SAT are Horn-formulas and formulas in 2-CNF (see chapter 7.1 and 7.2) and *nested formulas* [8], which are formulas with a hierarchical structure.

Because practical instances of SAT are normally not part of the polynomial solvable classes of SAT, we have to look for good heuristics to solve them. Heuristics for SAT are frequently compared using random formulas. In this paper we present a method to generate structured and thus possibly more practical examples for these comparisons.

The NP-completeness of SAT assures that any other NP-complete problem can be reduced [6] to SAT by simulating any Turing machine solving this problem. We will speak of "programming in propositional logic" if we give a reduction of a problem to SAT. The most surprising result is that implementations of problems in Satisfiability are derived much easier directly from arbitrary problems to SAT than via Turing machines.

Since P$\subseteq$NP, all polynomial solvable problems can also be reduced to SAT. Thus, we have a rich pool of easy and hard formulas for testing SAT-solving heuristics. The implementations give also rise to new heuristics for solving SAT.

The remainder of this paper is organized as follows: In the second chapter we introduce some useful transformations that simplify programming in propositional logic. In the third chapter a technique for transfering arbitrary propositional formulas to equivalent formulas in CNF is developed. In the fourth chapter different problems are implemented: polynomial solvable problems are described, followed by implementations of NP-complete problems starting with the first ten problems from the famous table in [6]. In the fifth chapter techniques are developed for implementing even problems dealing with numbers and for reducing the number of clauses for the problems presented in the fourth chapter. In the sixth chapter some heuristics are presented and their behavior on the basic formulas proposed in chapter 2 is analyzed. The seventh chapter discusses the expressiveness of restrictions of SAT, and in the eighth chapter important techniques for improving some of the basic formulas described in chapter 2 are shown. The summary follows in the ninth chapter together with three "little" open problems.

Chapter 2

Useful formulas

The formulas in this paper are all in conjunctive normal form (CNF), because this seems to be a natural form of representation, and instances of SATISFIABILITY are normally considered to be in this form.

We start by defining some useful formulas which will make "programs" more readable. The formula *at_most_one* (*at_least_one*) defines the property that at most one (at least one) literal of the arguments is true:

$$at\_most\_one\{l_1, \ldots, l_x\} := \bigwedge_{1 \leq i < j \leq x} (\overline{l_i} \vee \overline{l_j})$$

$$at\_least\_one\{l_1, \ldots, l_x\} := (l_1 \vee l_2 \vee \ldots \vee l_x)$$

With these formulas it is easy to define *exactly_one*, which is true if, and only if, exactly one of its arguments is true:

$$exactly\_one\{l_1, \ldots, l_x\} := at\_most\_one\{l_1, \ldots, l_x\} \wedge at\_least\_one\{l_1, \ldots, l_x\}$$

Efficient implementations of *at_least_one* and *exactly_one* are given in chapter 8. In this paper, implications of the form

$$(a_1 \wedge a_2 \wedge \ldots \wedge a_k) \Longrightarrow (b_1 \vee b_2 \vee \ldots \vee b_l)$$

are also considered to be representations in CNF even if they contain formulas of the types given above, because an implication $a \Longrightarrow b$ is equivalent to $\overline{a} \vee b$ in CNF.

Thus, it makes sense to define the formula *none (all)*, which is true if, and only if, none (all) of the arguments is (are) true:

$$none\{l_1, \ldots, l_x\} := (\overline{l_1} \wedge \overline{l_2} \wedge \ldots \wedge \overline{l_x})$$

$$all\{l_1, \ldots, l_x\} := (l_1 \wedge l_2 \wedge \ldots \wedge l_x)$$

Chapter 3

Useful techniques

3.1 Removing terms of the form "$\ldots \Longrightarrow \bigwedge_{\ldots} \ldots$"

This is a simple task:

$$\left(F \Rightarrow \bigwedge_{1 \leq i \leq k} X_i\right) \Longleftrightarrow \left(\overline{F} \vee \left(\bigwedge_{1 \leq i \leq k} X_i\right)\right) \Longleftrightarrow \left(\bigwedge_{1 \leq i \leq k} (\overline{F} \vee X_i)\right) \Longleftrightarrow \left(\bigwedge_{1 \leq i \leq k} (F \Rightarrow X_i)\right)$$

3.2 Removing terms of the form "$\ldots \bigvee_{\ldots} \bigwedge_{\ldots} (\ldots)$"

The problem and its solutions will be demonstrated by the following example:

$$F = F' \wedge \left(\bigvee_{1 \leq k \leq n} (a_k \wedge b_k)\right)$$

The right subformula can be transformed into CNF by introducing n new variables $[1], \ldots, [n]$ and reformulating it as follows:

$$F'' = F' \;\wedge\; \mathit{at\_least\_one}\,\{\,[k]\,|\,1 \leq k \leq n\,\} \;\wedge\; \bigwedge_{1 \leq k \leq n} ([k] \Rightarrow (a_k \wedge b_k))$$

It is easy to see that this transformation implies $F \equiv F''$. Now F'' can be transformed to F''' in CNF further by the previous technique:

$$F''' = F' \;\wedge\; \mathit{at\_least\_one}\,\{\,[k]\,|\,1 \leq k \leq n\,\} \;\wedge\; \bigwedge_{1 \leq k \leq n} ([k] \Rightarrow (a_k)) \;\wedge\; \bigwedge_{1 \leq k \leq n} ([k] \Rightarrow (b_k))$$

3.3 Removing terms of the form "$\ldots \Longleftrightarrow \ldots$"

This can be easily done by replacing "$X \Longleftrightarrow Y$" with " $(X \Longrightarrow Y) \wedge (Y \Longrightarrow X)$", which can be transformed further by the techniques described above, if necessary.

Chapter 4

Different Reductions

4.1 Problems from P

4.1.1 SHORTEST PATH $\propto$ SAT

INSTANCE: Graph $G = (V, E)$, $s, t \in V$, $K > 0$.
QUESTION: $\exists$ path of length$\leq$K between s and t ?

$$
\begin{aligned}
X &= \{\, [v, i] \mid v \in V, 1 \leq i \leq K \,\} \\
F &= \bigwedge_{2 \leq i \leq K} \bigwedge_{v \in V} (([v, i]) \Longrightarrow at\_least\_one \, \{\, [w, i-1] \mid w \in N(v) \,\}\,) \\
&\quad \wedge \quad at\_least\_one \, \{\, [t, i] \mid 1 \leq i \leq K \,\} \ \wedge \bigwedge_{v \in (V - N(s))} \left(\overline{[v, 1]}\right)
\end{aligned}
$$

This implementation makes the polynomial solvability of SHORTEST PATH obvious, because the above formula is in Co-Horn-form (i.e., at most one negative literal per clause)! The first part ensures, that a vertex v is reachable from s within i stages if, and only if, there is any neighbor of v reachable in $i-1$ stages. The second part ensures that t is reachable in at most K stages. The last part tells us, that no vertex which is not a neighbor of s is reachable from s in 1 stage.

4.1.2 BOOLEAN MATRIX MULTIPLICATION $\propto$ SAT

INSTANCE: $A, B \in \{0,1\}^{n^2}$
QUESTION: $\exists C \in \{0,1\}^{n^2} : C = A * B$, that is $\forall 1 \leq i, j \leq n : c_{ij} = \bigvee_{1 \leq k \leq n} (a_{ik} \wedge b_{kj})$?

$$
\begin{aligned}
X &= \{\, [x, i, j] \mid x \in \{a, b, c\}, 1 \leq i, j \leq n \,\} \\
F &= \bigwedge_{1 \leq i,j \leq n}^{a_{i,j}=1} ([a, i, j]) \wedge \bigwedge_{1 \leq i,j \leq n}^{a_{i,j}=0} (\overline{[a, i, j]}) \wedge \bigwedge_{1 \leq i,j \leq n}^{b_{i,j}=1} ([b, i, j]) \wedge \bigwedge_{1 \leq i,j \leq n}^{b_{i,j}=0} (\overline{[b, i, j]}) \\
&\wedge \bigwedge_{1 \leq i,j \leq n} \bigwedge_{1 \leq k \leq n} (([a, i, k] \wedge [b, k, j]) \Longrightarrow ([c, i, j])) \\
&\wedge \bigwedge_{1 \leq i,j \leq n} \left(([c, i, j]) \Longrightarrow \bigvee_{1 \leq k \leq n} ([a, i, k] \wedge [b, k, j]) \right)
\end{aligned}
$$

The above formulation as decision-problem is given for formal reasons only, because the above formula is clearly always satisfiable. The "result" is given here as the contents of [c,i,j] for any satisfying truth-assignment of the formula. The CNF-implementation is thus the following:

$$
\begin{array}{rcl}
X' & = & X \cup \{\, [k,i,j] \mid 1 \leq k,i,j \leq n \,\} \\
F' & = & \bigwedge_{1\leq i,j\leq n}^{a_{i,j}=1} ([a,i,j]) \wedge \bigwedge_{1\leq i,j\leq n}^{a_{i,j}=0} (\overline{[a,i,j]}) \wedge \bigwedge_{1\leq i,j\leq n}^{b_{i,j}=1} ([b,i,j]) \wedge \bigwedge_{1\leq i,j\leq n}^{b_{i,j}=0} (\overline{[b,i,j]}) \\
& \wedge & \bigwedge_{1\leq i,j\leq n} \bigwedge_{1\leq k\leq n} (([a,i,k] \wedge [b,k,j]) \Longrightarrow ([c,i,j])) \\
& \wedge & \bigwedge_{1\leq i,j\leq n} (([c,i,j]) \Longrightarrow \mathit{at\_least\_one}\, \{\, [k,i,j] \mid 1 \leq k \leq n \,\}) \\
& \wedge & \bigwedge_{1\leq i,j\leq n} \bigwedge_{1\leq k\leq n} (([k,i,j]) \Longrightarrow ([a,i,k])) \wedge \bigwedge_{1\leq i,j\leq n} \bigwedge_{1\leq k\leq n} (([k,i,j]) \Longrightarrow ([b,k,j]))
\end{array}
$$

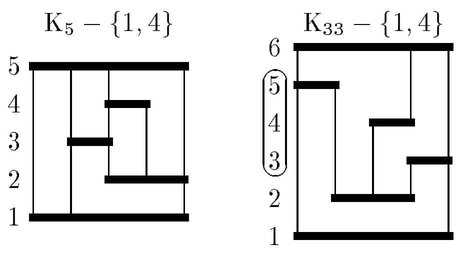
If the heuristic used to solve SATISFIABILITY has the property that clauses of length 1 are satisfied directly by setting the corresponding variable correctly (see also chapter 6), then the above formula is solved directly. This is because the first 4 parts force all [c,i,j] to be set to $true$, for which the result is $true$ by the fifth part, and all remaining [c,i,j] are set to $false$ by the last parts directly, because there will exist always newly generated clauses with only 1 literal.

4.1.3 PLANARITY $\propto$ SAT

INSTANCE: Graph $G = (V, E)$.
QUESTION: Is G planar ?

$$
\begin{array}{rcl}
X & = & \{\, [v,i] \mid v \in V, 1 \leq i \leq |V| \,\} \cup \{\, [x,y] \mid 1 \leq x \leq 2|V|-4, 1 \leq y \leq |V| \,\} \\
F & = & \bigwedge_{v\in V} \mathit{exactly\_one}\, \{\, [v,i] \mid 1 \leq i \leq |V| \,\} \wedge \bigwedge_{1\leq i\leq |V|} \mathit{exactly\_one}\, \{\, [v,i] \mid v \in V \,\} \\
& \wedge & \bigwedge_{1\leq y\leq |V|} \bigwedge_{1\leq x<2|V|-4} \Big(([x,y] \wedge \overline{[x+1,y]}) \Rightarrow \mathit{none}\, \{\, [z,y] \mid x < z \leq 2|V|-4 \,\} \Big) \\
& \wedge & \bigwedge_{\{u,v\}\in E} \bigwedge_{1\leq i,j\leq |V|}^{i\neq j} \left(\begin{array}{l} ([u,i] \wedge [v,j]) \Longrightarrow \\ \bigvee_{1\leq x\leq 2|V|-4} \left([x,i] \wedge \mathit{none} \left\{ [x,k] \;\middle|\; \begin{array}{l} \text{i<k<j, if i<j} \\ \text{j<k<i, if i>j} \end{array} \right\} \wedge [x,j] \right) \end{array} \right)
\end{array}
$$

The underlying idea for this implementation is the so called *horvert* representation of planar graphs, [7]. Here vertices are represented as horizontal segments and edges are drawn as vertical segments. The vertices are drawn at different levels, and the segment representing an edge is a vertical segment connecting the corresponding segments of the adjacent vertices. The most important fact is that a graph is planar if, and only if, it has a horvert representation. It is clear that there are exactly $|V|$ levels for the vertices, and it can be shown that $2|V|-4$ columns are sufficient for any horvert representation [7].



Examples for horvert-representations

The vertices have to be numbered in a special way to allow a vertex with number i to be placed on the $i-th$ level (generalized st-number). Thus the first two parts specify a correct numbering of the vertices, the third part specifies a correct placement of the vertex-segments, and the last part ensures, that all edge-segments could be drawn without problem. Therefore the test for planarity is implemented by the above formula. The CNF-implementation of PLANARITY follows:

$$
\begin{aligned}
X' &= X \cup \{\, [e,x] \mid e \in E, 1 \le x \le 2|V|-4 \,\} \\
F' &= \bigwedge_{v \in V} exactly\_one\,\{\, [v,i] \mid 1 \le i \le |V| \,\} \;\wedge \bigwedge_{1 \le i \le |V|} exactly\_one\,\{\, [v,i] \mid v \in V \,\} \\
&\wedge \bigwedge_{1 \le y \le |V|} \; \bigwedge_{1 \le x < 2|V|-4} \; \bigwedge_{x < z \le 2|V|-4} \Big(([x,y] \wedge \overline{[x+1,y]}) \Rightarrow (\overline{[z,y]}) \Big) \\
&\wedge \bigwedge_{e=\{u,v\} \in E} at\_least\_one\,\{\, [e,x] \mid 1 \le x \le 2|V|-4 \,\} \\
&\wedge \bigwedge_{e=\{u,v\} \in E} \; \bigwedge_{1 \le i \le |V|} \; \bigwedge_{1 \le x \le 2|V|-4} (([u,i] \wedge [e,x]) \Rightarrow ([x,i])) \\
&\wedge \bigwedge_{e=\{u,v\} \in E} \; \bigwedge_{1 \le j \le |V|} \; \bigwedge_{1 \le x \le 2|V|-4} (([v,j] \wedge [e,x]) \Rightarrow ([x,j])) \\
&\wedge \bigwedge_{e=\{u,v\} \in E} \; \bigwedge_{\substack{i \ne j \\ 1 \le i,j \le |V|}} \; \bigwedge_{1 \le x \le 2|V|-4} \; \bigwedge_{min\{i,j\} < k < max\{i,j\}} \Big(([u,i] \wedge [v,j] \wedge [e,x]) \Rightarrow (\overline{[x,k]}) \Big)
\end{aligned}
$$

Remark: If a planar graph can be embedded in the plane such that all vertices lie on the *outer face* (the one of infinite area) it is called *outerplanar*. In a horvert representation a vertex lies on the outer face if, and only if, it touches the left or right border. Thus OUTERPLANARITY (i.e., the test whether a given graph is outerplanar) can be implemented by the above CNF-formula for PLANARITY with the additional formulas:

$$
F'' = F' \wedge \bigwedge_{1 \le y \le |V|} at\_least\_one\{\, [1,y], [2|V|-4, y] \,\}
$$

4.1.4 STRONG CONNECTIVITY $\propto$ SAT

INSTANCE: Directed graph G=(V,A).
QUESTION: Is G strongly connected ?

$$
\begin{aligned}
X &= \{\, [i,v] \mid -(|V|-1) \le i \le |V|-1, v \in V \,\} \\
F &= ([0,v_0]) \wedge none\,\{\, [0,v] \mid v \in (V - \{v_0\}) \,\} \\
&\wedge \bigwedge_{v \in V} at\_least\_one\,\{\, [i,v] \mid 0 \le i \le |V|-1 \,\} \\
&\wedge \bigwedge_{0 < i \le |V|-1} \; \bigwedge_{v \in V} (([i,v]) \Longrightarrow at\_least\_one\,\{\, [j,w] \mid 0 \le j < i, (w,v) \in A \,\}) \\
&\wedge \bigwedge_{v \in V} at\_least\_one\,\{\, [i,v] \mid -(|V|-1) \le i \le 0 \,\} \\
&\wedge \bigwedge_{-(|V|-1) \le i < 0} \; \bigwedge_{v \in V} (([i,v]) \Longrightarrow at\_least\_one\,\{\, [j,w] \mid i < j \le 0, (v,w) \in A \,\})
\end{aligned}
$$

A digraph is strongly connected if, and only if, there is a path from u to v for each ordered pair of vertices u and v. This can be characterized alternatively in the following way: A digraph is strongly connected if, and only if, any fixed vertex v_0 has all other vertices both as descendants and ascendents. The third and fourth part of the above formula ensure all other vertices as descendants, and the last two parts verify all other vertices as ascendents, both for the vertex $v_0 \in V$. Here again the formula is in Co-Horn-form, indicating the polynomial solvability of STRONG-CONNECTIVITY.

4.2 NP-complete problems from G&J

In this section the first ten problems and a collection from the many other areas of the table of [6] are presented. The first ten are chosen to force us to implement problems even if they turned out to be difficult to handle (e.g. ACHROMATIC NUMBER). We also present a collection of problems from different areas requiring different "techniques" in their implementations.

4.2.1 [GT1] VERTEX COVER $\propto$ SAT

INSTANCE: Graph $G = (V, E)$, positive integer $K \leq |V|$.
QUESTION: $\exists V' \subseteq V, |V'| \leq K, \forall \{u, v\} \in E : \{u, v\} \cap V' \neq \emptyset$?

$$\begin{aligned} X &= \{\, [v, i] \mid v \in V, 1 \leq i \leq K \,\} \\ F &= \bigwedge_{1 \leq i \leq K} \mathit{at\_most\_one} \{\, [v, i] \mid v \in V \,\} \\ &\wedge \bigwedge_{\{u,v\} \in E} \mathit{at\_least\_one} \{\, [u, i], [v, i] \mid 1 \leq i \leq K \,\} \end{aligned}$$

The first part implies, that at most K vertices are chosen (literals $[v, i]$ set to $true$), and the second part verifies that the chosen vertex-set is a vertex cover (VC) indeed.

4.2.2 [GT2] DOMINATING SET $\propto$ SAT

INSTANCE: Graph $G = (V, E)$, positive integer $K \leq |V|$.
QUESTION: $\exists V' \subseteq V, |V'| \leq K, \forall v \in V : (\{v\} \cup N(v)) \cap V' \neq \emptyset$?

$$\begin{aligned} X &= \{\, [v, i] \mid v \in V, 1 \leq i \leq K \,\} \\ F &= \bigwedge_{1 \leq i \leq K} \mathit{at\_most\_one} \{\, [v, i] \mid v \in V \,\} \\ &\wedge \bigwedge_{v \in V} \mathit{at\_least\_one} \{\, [w, i] \mid 1 \leq i \leq K, w \in (\{v\} \cup N(v)) \,\} \end{aligned}$$

The first part assures again that at most K vertices are chosen, and the second part verifies that the chosen vertex-set is a dominating set (DS). Here $N(v)$ denotes the set of vertices adjacent to vertex v.

4.2.3 [GT2a] EDGE DOMINATING SET $\propto$ SAT

INSTANCE: Graph $G = (V, E)$, positive integer $K \leq |E|$.
QUESTION: $\exists E' \subseteq E, |E'| \leq K, \forall e \in E, \exists f \in E' : e \cap f \neq \emptyset$?

$$\begin{aligned} X &= \{\, [e, i] \mid e \in E, 1 \leq i \leq K \,\} \cup \{\, [v] \mid v \in V \,\} \\ F &= \bigwedge_{1 \leq i \leq K} \mathit{at\_most\_one} \{\, [e, i] \mid e \in E \,\} \end{aligned}$$

$$\wedge \quad \bigwedge_{v \in V} (([v]) \Longrightarrow \mathit{at\_least\_one}\,\{\,[e,i] \mid v \in e \in E, 1 \le i \le K\,\})$$

$$\wedge \quad \bigwedge_{\{u,v\} \in E} \mathit{at\_least\_one}\{[u],[v]\}$$

The first part selects at most K edges, which are checked by the remaining two parts to be an edge dominating set (EDS). Here $v \in e \in E$ denotes all edges e from E, which have v as one of their end points.

4.2.4 [GT3] DOMATIC NUMBER $\propto$ SAT

INSTANCE: Graph $G = (V, E)$, positive integer $K \le |V|$.
QUESTION: $\exists V = V_1 \dot\cup \ldots \dot\cup V_k, k \ge K, \forall 1 \le i \le k : V_i \in DS(G)$?

$$\begin{aligned}
X &= \{\,[v,i] \mid v \in V, 1 \le i \le |V|\,\} \\
F &= \bigwedge_{v \in V} \mathit{exactly_one}\,\{\,[v,i] \mid 1 \le i \le |V|\,\} \ \wedge \bigwedge_{1 \le i \le K} \mathit{at_least_one}\,\{\,[v,i] \mid v \in V\,\} \\
&\wedge \bigwedge_{1 \le i \le |V|} \left(\mathit{at_least_one}\,\{\,[u,i] \mid u \in V\,\} \Rightarrow \left(\bigwedge_{v \in V} \mathit{at_least_one}\,\{\,[w,i] \mid w \in (\{v\} \cup N(v))\,\}\right)\right)
\end{aligned}$$

The first part partitions the vertex-set into $k \ge K$ disjoint sets. Each "nonempty" partition is then checked by the third part to be a DS. The CNF-formula now follows:

$$\begin{aligned}
X' &= X \\
F' &= \bigwedge_{v \in V} \mathit{exactly_one}\,\{\,[v,i] \mid 1 \le i \le |V|\,\} \ \wedge \bigwedge_{1 \le i \le K} \mathit{at_least_one}\,\{\,[v,i] \mid v \in V\,\} \\
&\wedge \bigwedge_{1 \le i \le |V|} \bigwedge_{u \in V} \bigwedge_{v \in V} (([u,i]) \Longrightarrow \mathit{at_least_one}\,\{\,[w,i] \mid w \in (\{v\} \cup N(v))\,\})
\end{aligned}$$

4.2.5 [GT4] GRAPH K-COLORABILITY $\propto$ SAT

INSTANCE: Graph $G = (V, E)$, positive integer $K \le |V|$.
QUESTION: $\exists f : V \to \{1, \ldots, K\}, \forall \{u,v\} \in E : f(u) \ne f(v)$?

$$\begin{aligned}
X &= \{\,[v,i] \mid v \in V, 1 \le i \le K\,\} \\
F &= \bigwedge_{v \in V} \mathit{exactly_one}\,\{\,[v,i] \mid 1 \le i \le K\,\} \\
&\wedge \bigwedge_{\{u,v\} \in E} \bigwedge_{1 \le i \le K} \mathit{at_most_one}\{[u,i],[v,i]\}
\end{aligned}$$

This problem is NP-complete for any fixed $K \ge 3$, and for $K = 3$ the cardinality of the variable-set ($|X| = 3|V|$) and the clause-set ($|F| = 4|V| + 3|E|$) is linear in the input-size of 3-colorability. Even the sum of the length of clauses ($\sum_j |C_j| = (3*2+3)*|V| + 3*2*|E| = O(|V|+|E|)$) is linear in the input-size for $K = 3$. Here the first part selects one "color" for each vertex, and the second part verifies that no conflict-edge exists.

4.2.6 [GT5] ACHROMATIC NUMBER $\propto$ SAT

INSTANCE: Graph $G = (V, E)$, positive integer $K \leq |V|$.
QUESTION: $\exists V = V_1 \dot\cup \ldots \dot\cup V_k, k \geq K, \forall 1 \leq i \leq k : V_i \in IS(G)$,
$\wedge \forall 1 \leq i < j \leq k : (V_i \cup V_j) \notin IS(G)$?

$$\begin{aligned}
X &= \{\, [v,i] \mid v \in V, 1 \leq i \leq |V| \,\} \\
F &= \bigwedge_{v \in V} exactly_one\, \{\, [v,i] \mid 1 \leq i \leq |V| \,\} \;\wedge \bigwedge_{1 \leq i \leq K} at_least_one\, \{\, [v,i] \mid v \in V \,\} \\
&\wedge \bigwedge_{1 \leq i \leq |V|} \bigwedge_{\{u,v\} \in E} at_most_one\{[u,i],[v,i]\} \\
&\wedge \bigwedge_{\substack{i \neq j \\ 1 \leq i,j \leq |V|}} \left(\begin{pmatrix} at_least_one\, \{\, [x,i] \mid x \in V \,\} \\ \wedge at_least_one\, \{\, [y,j] \mid y \in V \,\} \end{pmatrix} \Rightarrow \bigvee_{\{u,v\} \in E} ([u,i] \wedge [v,j]) \right)
\end{aligned}$$

The first two parts partition the vertex-set into $k \geq K$ disjoint sets, each of which is verified to be an IS in the third part. In the last part it is checked wether any two sets in the partition fail to be independent. To avoid problems with "empty" sets in partitions, the actual existence of these sets is forced via the implication. This leads to the CNF-implementation:

$$\begin{aligned}
X' &= X \cup \{\, [i,j,e] \mid 1 \leq i,j \leq |V|, e \in E \,\} \\
F' &= \bigwedge_{v \in V} exactly_one\, \{\, [v,i] \mid 1 \leq i \leq |V| \,\} \;\wedge \bigwedge_{1 \leq i \leq K} at_least_one\, \{\, [v,i] \mid v \in V \,\} \\
&\wedge \bigwedge_{1 \leq i \leq |V|} \bigwedge_{\{u,v\} \in E} at_most_one\{[u,i],[v,i]\} \\
&\wedge \bigwedge_{1 \leq i,j \leq |V|} at_least_one\, \{\, [i,j,e] \mid e \in E \,\} \\
&\wedge \bigwedge_{\substack{i \neq j \\ 1 \leq i,j \leq |V|}} \bigwedge_{x \in V} \bigwedge_{y \in V} \bigwedge_{e = \{u,v\} \in E} \Big(\quad ([x,i] \wedge [y,j]) \Longrightarrow (([i,j,e]) \Rightarrow ([u,i])) \quad \Big) \\
&\wedge \bigwedge_{\substack{i \neq j \\ 1 \leq i,j \leq |V|}} \bigwedge_{x \in V} \bigwedge_{y \in V} \bigwedge_{e = \{u,v\} \in E} \Big(\quad ([x,i] \wedge [y,j]) \Longrightarrow (([i,j,e]) \Rightarrow ([v,j])) \quad \Big)
\end{aligned}$$

At first glance, this seems to be a little complicated, but it is only a straightforward application of the techniques from the third chapter. The appearance of an implication to an implication is valid because it is really a disjunction.

4.2.7 [GT6] MONOCHROMATIC TRIANGLE $\propto$ SAT

INSTANCE: Graph $G = (V, E)$.
QUESTION: $\exists E = E_1 \dot\cup E_2$ such that (V, E_1) and (V, E_2) are triangle-free ?

$$\begin{aligned}
X &= \{\, [e,i] \mid e \in E, 1 \leq i \leq 2 \,\} \\
F &= \bigwedge_{e \in E} exactly_one\, \{\, [e,i] \mid 1 \leq i \leq 2 \,\} \\
&\wedge \bigwedge_{\triangle\{e_1,e_2,e_3\} \subseteq E} ([e_1,1] \vee [e_2,1] \vee [e_3,1]) \;\wedge \bigwedge_{\triangle\{e_1,e_2,e_3\} \subseteq E} ([e_1,2] \vee [e_2,2] \vee [e_3,2])
\end{aligned}$$

Here $\triangle\{e_1, e_2, e_3\}$ denotes a triple of edges, which together build a triangle. This implementation has an interesting property: Replacing *exactly_one* by *at_most_one* preserves the NP-completeness, and in this case all 2-clauses are completely negative, whereas all 3-clauses are completely positive. It is important to notice that SATISFIABILITY remains NP-complete even if applied to such restricted formulas !

4.2.8 [GT7] FEEDBACK VERTEX SET $\propto$ SAT

INSTANCE: Directed graph $G = (V, A)$, positive integer $K \leq |V|$.
QUESTION: $\exists V' \subseteq V, |V'| \leq K : G - V'$ has no directed cycles (is a DAG)?

$$
\begin{aligned}
X &= \{\, [v,i] \mid v \in V, 1 \leq i \leq |V| \,\} \\
F &= \bigwedge_{1 \leq i \leq |V|} \mathit{exactly\_one}\{\, [v,i] \mid v \in V \,\} \wedge \bigwedge_{v \in V} \mathit{exactly\_one}\{\, [v,i] \mid 1 \leq i \leq |V| \,\} \\
&\wedge \bigwedge_{(u,v) \in A} \bigwedge_{K < i \leq |V|} \left(([u,i]) \Longrightarrow \mathit{at\_least\_one} \left\{ [v,j] \;\middle|\; \begin{array}{l} 1 \leq j \leq K \\ i < j \leq |V| \end{array} \right\} \right)
\end{aligned}
$$

The implementation relies on the fact that any DAG (directed acyclic graph) possesses a numbering of its vertices with all of its arcs being directed from lower to higher numbered vertices. The first two parts specify a one-to-one numbering of the vertices, where the vertices with numbers up to K are considered to build V'. In the third part the previously mentioned DAG-property is checked for the reduced graph.

4.2.9 [GT8] FEEDBACK ARC SET $\propto$ SAT

INSTANCE: Directed graph $G = (V, A)$, positive integer $K \leq |A|$.
QUESTION: $\exists A' \subseteq A, |A'| \leq K : G - A'$ has no directed cycles ?

$$
\begin{aligned}
X &= \{\, [v,i] \mid v \in V, 1 \leq i \leq |V| \,\} \cup \{\, [a,j] \mid a \in A, 1 \leq j \leq K \,\} \\
F &= \bigwedge_{1 \leq i \leq |V|} \mathit{exactly\_one}\{\, [v,i] \mid v \in V \,\} \wedge \bigwedge_{v \in V} \mathit{exactly\_one}\{\, [v,i] \mid 1 \leq i \leq |V| \,\} \\
&\wedge \bigwedge_{1 \leq j \leq K} \mathit{at\_most\_one}\{\, [a,j] \mid a \in A \,\} \\
&\wedge \bigwedge_{a=(u,v) \in A} \bigwedge_{1 < i \leq |V|} \left(\begin{array}{l} ([u,i] \wedge \mathit{none}\{\, [v,j] \mid i < j \leq |V| \,\}) \\ \Longrightarrow \mathit{at\_least\_one}\{\, [a,l] \mid 1 \leq l \leq K \,\} \end{array} \right)
\end{aligned}
$$

This implementation uses the same idea as in the formula of [GT7]: if any arc is directed from higher to lower numbered vertices, then it must be one of those at most K arcs, which are to be removed from G!

4.2.10 [GT9] PARTIAL FEEDBACK EDGE SET $\propto$ SAT

INSTANCE: Graph $G = (V, E)$, positive integers $K \leq |E|$ and $L \leq |V|$.
QUESTION: $\exists E' \subseteq E, |E'| \leq K$: E' contains at least one edge of every cycle of length L or less in G ?

$$
\begin{aligned}
X &= \{\, [e,i] \mid e \in E, 1 \leq i \leq K \,\} \\
F &= \bigwedge_{1 \leq i \leq K} \mathit{at\_most\_one}\{\, [e,i] \mid e \in E \,\} \\
&\wedge \bigwedge_{\bigcirc C \in E}^{|C| \leq L} \mathit{at\_least\_one}\{\, [e,i] \mid e \in C, 1 \leq i \leq K \,\}
\end{aligned}
$$

Here $\bigcirc C$ denotes a cycle of G. The above implementation has one major drawback: The number of cycles of length ≤L grows very fast. However, the above problem is NP-complete even for any fixed value $L \geq 3$. In the case of $L = 3$, the number of clauses is $O(|V|^3)$ in the above implementation.

4.2.11 [GT10] MINIMUM MAXIMAL MATCHING ∝ SAT

INSTANCE: Graph $G = (V, E)$, positive integer $K \leq |E|$.
QUESTION: $\exists E' \subseteq E, |E'| \leq K, \forall \{u, v\} \in E : \{u, v\} \cap V(E') \neq \emptyset$?

$$\begin{array}{rcl} X & = & \{\,[e, i] \mid e \in E, 1 \leq i \leq K\,\} \\ F & = & \bigwedge\limits_{1 \leq i \leq K} \mathit{at\_most\_one}\,\{\,[e, i] \mid e \in E\,\} \;\wedge \bigwedge\limits_{e,f \in E}^{e \cap f \neq \emptyset} \mathit{at\_most\_one}\,\{\,[e, i], [f, i] \mid 1 \leq i \leq K\,\} \\ & \wedge & \bigwedge\limits_{e \in E} \left(\begin{array}{l} \mathit{none}\,\{\,[e, i] \mid 1 \leq i \leq K\,\} \Rightarrow \\ \mathit{at\_least\_one}\,\{\,[f, i] \mid 1 \leq i \leq K, f \in E, e \cap f \neq \emptyset\,\} \end{array} \right) \end{array}$$

The first two parts of the formula specify that a matching with size at most K is chosen. The third part ensures that the chosen matching is indeed maximal. $V(E')$ in the problem formulation denotes the set of vertices "used" by all edges of E'.

4.2.12 [GT15] PARTITION INTO CLIQUES ∝ SAT

INSTANCE: Graph $G = (V, E)$, positive integer $K \leq |V|$.
QUESTION: $\exists V = V_1 \dot\cup \ldots \dot\cup V_k, k \leq K, \forall 1 \leq i \leq k : V_i$ is a complete graph ?

$$\begin{array}{rcl} X & = & \{\,[v, i] \mid v \in V, 1 \leq i \leq K\,\} \\ F & = & \bigwedge\limits_{v \in V} \mathit{exactly\_one}\,\{\,[v, i] \mid 1 \leq i \leq K\,\} \\ & \wedge & \bigwedge\limits_{1 \leq i \leq K} \bigwedge\limits_{\{u,v\} \notin E}^{u \neq v} \mathit{at\_most\_one}\{[u, i], [v, i]\} \end{array}$$

Here the first part partitions the vertex set into $k \leq K$ disjoint sets, and the second part verifies the clique-property for each partition.

4.2.13 [GT20] INDEPENDENT SET ∝ SAT

INSTANCE: Graph $G = (V, E)$, positive integer $K > 0$.
QUESTION: $\exists V' \subseteq V, |V'| \geq K, \forall \{u, v\} \in E : \{u, v\} \not\subseteq V'$?

$$\begin{array}{rcl} X & = & \{\,[v, i] \mid v \in V, 1 \leq i \leq K\,\} \\ F & = & \bigwedge\limits_{1 \leq i \leq K} \mathit{at\_least\_one}\,\{\,[v, i] \mid v \in V\,\} \\ & \wedge & \bigwedge\limits_{\{u,v\} \in E} \mathit{at\_most\_one}\,\{\,[u, i], [v, i] \mid 1 \leq i \leq K\,\} \end{array}$$

The first part selects at least K vertices, and the second part verifies that no pair of those vertices is adjacent.

4.2.14 [GT27] PLANAR SUBGRAPH $\propto$ SAT

INSTANCE: Graph $G = (V, E)$, positive integer $K \leq |E|$.
QUESTION: $\exists E' \subseteq E, |E'| \geq K : G' = (V, E')$ is planar ?

$$
\begin{aligned}
X &= \{ [v,i] \mid v \in V, 1 \leq i \leq |V| \} \cup \{ [x,y] \mid 1 \leq x \leq 2|V|-4, 1 \leq y \leq |V| \} \\
&\cup \{ [e,x] \mid e \in E, 1 \leq x \leq 2|V|-4 \} \cup \{ [i,e] \mid 1 \leq i \leq K, e \in E \} \\
F &= \bigwedge_{1 \leq i \leq K} \mathit{at\_least\_one} \{ [i,e] \mid e \in E \} \wedge \bigwedge_{e \in E} \mathit{at\_most\_one} \{ [i,e] \mid 1 \leq i \leq K \} \\
&\wedge \bigwedge_{v \in V} \mathit{exactly\_one} \{ [v,i] \mid 1 \leq i \leq |V| \} \wedge \bigwedge_{1 \leq i \leq |V|} \mathit{exactly\_one} \{ [v,i] \mid v \in V \} \\
&\wedge \bigwedge_{1 \leq y \leq |V|} \bigwedge_{1 \leq x < 2|V|-4} \bigwedge_{x < z \leq 2|V|-4} \Big(([x,y] \wedge \overline{[x+1,y]}) \Rightarrow (\overline{[z,y]}) \Big) \\
&\wedge \bigwedge_{e=\{u,v\} \in E} \bigwedge_{1 \leq i \leq K} (([i,e]) \Longrightarrow \mathit{at\_least\_one} \{ [e,x] \mid 1 \leq x \leq 2|V|-4 \}) \\
&\wedge \bigwedge_{e=\{u,v\} \in E} \bigwedge_{1 \leq i \leq |V|} \bigwedge_{1 \leq x \leq 2|V|-4} (([u,i] \wedge [e,x]) \Rightarrow ([x,i])) \\
&\wedge \bigwedge_{e=\{u,v\} \in E} \bigwedge_{1 \leq j \leq |V|} \bigwedge_{1 \leq x \leq 2|V|-4} (([v,j] \wedge [e,x]) \Rightarrow ([x,j])) \\
&\wedge \bigwedge_{e=\{u,v\} \in E} \bigwedge_{\substack{i \neq j \\ 1 \leq i,j \leq |V|}} \bigwedge_{1 \leq x \leq 2|V|-4} \bigwedge_{\min\{i,j\} < k < \max\{i,j\}} \Big(([u,i] \wedge [v,j] \wedge [e,x]) \Rightarrow (\overline{[x,k]}) \Big)
\end{aligned}
$$

This implementation is in CNF-form. It is the implementation for PLANARITY with some added formulas. The first two parts ensure that at least K different edges are in E'. The sixth expression is modified to force a "good position" for the selected edges of E' only.

4.2.15 [GT39] HAMILTONIAN PATH $\propto$ SAT

INSTANCE: Graph G=(V,E).
QUESTION: $\exists f : \{1, \ldots, |V|\} \xrightarrow{1:1} V, \forall 1 \leq i < |V| : \{f(i), f(i+1)\} \in E$?

$$
\begin{aligned}
X &= \{ [v,i] \mid v \in V, 1 \leq i \leq |V| \} \\
F &= \bigwedge_{1 \leq i \leq |V|} \mathit{exactly\_one} \{ [v,i] \mid v \in V \} \wedge \bigwedge_{v \in V} \mathit{exactly\_one} \{ [v,i] \mid 1 \leq i \leq |V| \} \\
&\wedge \bigwedge_{1 \leq i < |V|} \bigwedge_{v \in V} (([v,i]) \Longrightarrow \mathit{at\_least\_one} \{ [w,i+1] \mid w \in N(v) \})
\end{aligned}
$$

In the first two parts the one-to-one numbering of the vertices is specified. In the third part it is verified that edges between the $i-th$ and $(i+1)-th$ vertex exist for all $1 \leq i < |V|$. This implementation has O($|V|^2$) clauses, if the technique from chapter 8 for *exactly_one* is used.

4.2.16 [ND2] MAXIMUM LEAF SPANNING TREE $\propto$ SAT

INSTANCE: Graph $G = (V, E)$, positive integer $K \leq |V|$.
QUESTION: Is there a spanning tree for G in which K or more vertices have degree 1 ?

$$
X = \{ [v,i] \mid v \in V, 1 \leq i \leq |V| \}
$$

$$
\begin{aligned}
F &= \bigwedge_{1\le i\le |V|} \mathit{exactly\_one}\,\{\,[v,i]\mid v\in V\,\}\\
&\wedge \bigwedge_{v\in V} \mathit{exactly\_one}\,\{\,[v,i]\mid 1\le i\le |V|\,\}\\
&\wedge \bigwedge_{1\le i\le K}\ \bigwedge_{v\in V} (([v,i]) \Longrightarrow \mathit{at\_least\_one}\,\{\,[w,j]\mid w\in N(v), K<j\le |V|\,\})\\
&\wedge \bigwedge_{K<i<|V|}\ \bigwedge_{v\in V} (([v,i]) \Longrightarrow \mathit{at\_least\_one}\,\{\,[w,j]\mid w\in N(v), i<j\le |V|\,\})
\end{aligned}
$$

The above formula ensures the existence of a connected subgraph such that the vertices with number 1 to K have degree 1. This corresponds to the existence of a spanning tree with K or more leaves (level numbering).

4.2.17 [SP4] SET-SPLITTING $\propto$ SAT

INSTANCE: Collection C of subsets of a finite set S.
QUESTION: $\exists S = S_1 \dot\cup S_2, \forall c\in C : c \not\subseteq S_1 \wedge c \not\subseteq S_2$?

$$
\begin{aligned}
X &= \{\,[s]\mid s\in S\,\}\\
F &= \bigwedge_{c\in C} \mathit{at\_least\_one}\,\{\,[s]\mid s\in c\,\}\\
&\wedge \bigwedge_{c\in C} \mathit{at\_least\_one}\left\{\,\overline{[s]}\mid s\in c\,\right\}
\end{aligned}
$$

Here the partitioning into S_1 and S_2 is defined by the truth values of the variables. An element s belongs to S_1 if $[s]$ is set to $true$, otherwise it belongs to S_2. Thus, the required property is checked by the above formula. In the case of $|c| = 2$ for each $c \in C$ this corresponds to the test whether a graph is bipartite, which can be done in polynomial time because the above formula is in 2-CNF (see chapter 7.1) in this case.

4.2.18 [SP8] HITTING SET $\propto$ SAT

INSTANCE: Collection C of subsets of a finite set S, positive integer $K \le |S|$.
QUESTION: $\exists S' \subseteq S, |S'| \le K, \forall c \in C : c \cap S' \ne \emptyset$?

$$
\begin{aligned}
X &= \{\,[s,i]\mid s\in S, 1\le i\le K\,\}\\
F &= \bigwedge_{1\le i\le K} \mathit{at\_most\_one}\,\{\,[s,i]\mid s\in S\,\}\\
&\wedge \bigwedge_{c\in C} \mathit{at\_least\_one}\,\{\,[s,i]\mid s\in c, 1\le i\le K\,\}
\end{aligned}
$$

This is just a generalization of VERTEX COVER ([GT1]), where $|c| = 2$ for all $c \in C$. The first part selects at most K elements form S to be part of S'. The second part verifies that this selection is indeed a Hitting set.

4.2.19 [SR21] GROUPING BY SWAPPING $\propto$ SAT

INSTANCE: Finite alphabet Σ, string $x \in \Sigma^*$, and a positive integer K.
QUESTION: Is there a sequence of K or fewer adjacent symbol interchanges that converts x into a string y in which all occurrences of each symbol $a \in \Sigma$ are in a single block, i.e. y has no subsequences of the form aba for $a, b \in \Sigma$ and $a \ne b$?
$(\Sigma = \{1, \dots, n\}, |x| = k)$

$$
\begin{aligned}
X &= \{\,[t,i,j] \mid 0 \le t \le K, 1 \le i \le k, 1 \le j \le n\,\} \\
&\cup \; \{\,[t,i] \mid 1 \le t \le K, 1 \le i < k\,\} \\
F &= \bigwedge_{1 \le i \le k} ([0,i,x_i]) \wedge \bigwedge_{1 \le i \le k} \bigwedge_{\substack{x_i \ne j \\ 1 \le j \le n}} \left(\overline{[0,i,j]}\right) \\
&\wedge \; \bigwedge_{1 \le t \le K} \bigwedge_{1 \le i \le k} exactly\_one \,\{\,[t,i,j] \mid 1 \le j \le n\,\} \\
&\wedge \; \bigwedge_{1 \le t \le K} exactly\_one \,\{\,[t,i] \mid 1 \le i < k\,\} \\
&\wedge \; \bigwedge_{1 \le t \le K} \bigwedge_{2 \le i \le k} \bigwedge_{1 \le h < i} \left(([t,i]) \Longrightarrow \bigwedge_{1 \le j \le n} ([t,h,j] \Longleftrightarrow [t-1,h,j]) \right) \\
&\wedge \; \bigwedge_{1 \le t \le K} \bigwedge_{1 \le i < k-1} \bigwedge_{i+1 < h \le k} \left(([t,i]) \Longrightarrow \bigwedge_{1 \le j \le n} ([t,h,j] \Longleftrightarrow [t-1,h,j]) \right) \\
&\wedge \; \bigwedge_{1 \le t \le K} \bigwedge_{1 \le i < k} \left(([t,i]) \Longrightarrow \bigwedge_{1 \le j \le n} ([t,i,j] \Longleftrightarrow [t-1,i+1,j]) \right) \\
&\wedge \; \bigwedge_{1 \le t \le K} \bigwedge_{1 \le i < k} \left(([t,i]) \Longrightarrow \bigwedge_{1 \le j \le n} ([t,i+1,j] \Longleftrightarrow [t-1,i,j]) \right) \\
&\wedge \; \bigwedge_{1 \le i < h \le k} \bigwedge_{1 \le j \le n} \left(([K,i,j] \wedge [K,h,j]) \Longrightarrow \bigwedge_{i < g < h} ([K,g,j]) \right)
\end{aligned}
$$

In this formula it is assumed that x contains at least one element from Σ more than once. This enables swapping "without any effect", such that the test for a successful series of swaps can be done for the $K-th$ stage only (in the last part). The first two parts place the input string x into stage 0. The third part verifies that x has exactly one character at each position and each stage. The fourth part specifies a sequence of K swaps of adjacent elements. The remaining parts then specify the behavior of each swap between the stages. The implementation in CNF-form can easily be derived by the techniques described in chapter 3 but we will omit the details here.

Chapter 5

Numbers

5.1 Counting

In this section, a method will be developed to encode "counting" of elements in a more efficient way than in the fourth chapter. For example, VERTEX COVER ([GT1]) needs $|V| * K$ variables only to ensure, that at most K vertices are chosen. The technique described in this section will decrease this number to only $O(|V| * \log K)$ variables and also diminish the number of clauses needed for this task from $O(|E| + |V| * K)$ to $O(|E| + |V| * \log K)$.

Now each element of the original set will possess one array of $\lceil \log(K+1) \rceil$ and one array of $\lceil \log(K+1) \rceil + 1$ boolean variables. The first array represents the actual "sum", the other is playing the role of "carry-flags". Here *true* represents the value 1, and *false* represents the value 0. An element is chosen if, and only if, the "least carry-bit" is *true*. Denote the arrays of element i by a_i and c_i. Now a_i and the rest of c_i except the least element are determined as:

$$a_{ij} = a_{(i-1)j} \oplus c_{ij} \ , \ c_{i(j+1)} = \begin{cases} \overline{a_{(i-1)j}} \wedge c_{ij} \, , \, if \ decrementing \\ a_{(i-1)j} \wedge c_{ij} \, , \, if \ incrementing \end{cases}$$

In addition a "starting array" holding the value of K ($K-1$) in the case of minimization (maximization) is available. The $\lceil \log(K+1) \rceil - th$ bit of each c_i-array detects any overflow. If the problem is formulated as "$\leq K$", then a clause $none \, \{ \, c_{i(\lceil \log(K+1) \rceil)} \mid 1 \leq i \leq n \, \}$ will prevent us from choosing more than K elements (elements with least carry-bit set true). In the other case ("$\geq K$"), in addition to the starting-value "$K-1$", the clause $at\_least\_one \, \{ \, c_{i(\lceil \log(K+1) \rceil)} \mid 1 \leq i \leq n \, \}$ will ensure that an overflow occurs. This implies that "$> K-1$", that is "$\geq K$" elements have been chosen. This new techniques will be demonstrated with two examples taken from the fourth chapter:

5.1.1 [GT1] VERTEX COVER $\propto$ SAT

INSTANCE: Graph $G = (V, E)$, V=$\{1, \ldots, n\}$, positive integer $K \leq |V|$.
QUESTION: $\exists V' \subseteq V, |V'| \leq K, \forall \{u, v\} \in E : \{u, v\} \cap V' \neq \emptyset$?

$$\begin{aligned}
X \;&=\; \{ \, [i, a, j] \mid 0 \leq i \leq |V|, 0 \leq j < \lceil \log(K+1) \rceil \, \} \\
&\cup\; \{ \, [i, c, j] \mid 1 \leq i \leq |V|, 0 \leq j \leq \lceil \log(K+1) \rceil \, \} \\
F \;&=\; \bigwedge_{0 \leq i < \lceil \log(K+1) \rceil}^{K_i = 0} \left(\overline{[0, a, i]} \right) \wedge \bigwedge_{0 \leq i < \lceil \log(K+1) \rceil}^{K_i = 1} ([0, a, i]) \\
&\wedge\; \bigwedge_{1 \leq i \leq |V|} \; \bigwedge_{0 \leq j < \lceil \log(K+1) \rceil} \left(\left(\overline{[i-1, a, j]} \wedge [i, c, j] \right) \Longleftrightarrow ([i, c, j+1]) \right) \\
&\wedge\; \bigwedge_{1 \leq i \leq |V|} \; \bigwedge_{0 \leq j < \lceil \log(K+1) \rceil} \left(\left(([i-1, a, j] \wedge \overline{[i, c, j]}) \vee (\overline{[i-1, a, j]} \wedge [i, c, j]) \right) \Longleftrightarrow ([i, a, j]) \right) \\
&\wedge\; none \, \{ \, [i, c, \lceil \log(K+1) \rceil] \mid 1 \leq i \leq |V| \, \} \; \wedge \bigwedge_{\{u,v\} \in E} at_least_one \{ \, [u, c, 0], [v, c, 0] \, \}
\end{aligned}$$

The first two parts place the binary representation of K into the starting array $[0,a]$. The third and fourth part force the correctness of any calculation, and the fifth part assures, that no overflow occurs, and therefore that at most K vertices are chosen. The last part verifies that the chosen vertex-set is a vertex cover. The CNF-implementation follows:

$$
\begin{aligned}
X' &= X \\
F' &= \bigwedge_{0 \le i < \lceil \log(K+1) \rceil}^{K_i=0} \left(\overline{[0,a,i]}\right) \wedge \bigwedge_{0 \le i < \lceil \log(K+1) \rceil}^{K_i=1} ([0,a,i]) \\
&\wedge \bigwedge_{1 \le i \le |V|} \bigwedge_{0 \le j < \lceil \log(K+1) \rceil} \left(\left(\overline{[i-1,a,j]} \wedge [i,c,j]\right) \Longrightarrow ([i,c,j+1])\right) \\
&\wedge \bigwedge_{1 \le i \le |V|} \bigwedge_{0 \le j < \lceil \log(K+1) \rceil} \left(\left(\overline{[i-1,a,j]}\right) \Longleftarrow ([i,c,j+1])\right) \\
&\wedge \bigwedge_{1 \le i \le |V|} \bigwedge_{0 \le j < \lceil \log(K+1) \rceil} (([i,c,j]) \Longleftarrow ([i,c,j+1])) \\
&\wedge \bigwedge_{1 \le i \le |V|} \bigwedge_{0 \le j < \lceil \log(K+1) \rceil} \left(\left([i-1,a,j] \wedge \overline{[i,c,j]}\right) \Longrightarrow ([i,a,j])\right) \\
&\wedge \bigwedge_{1 \le i \le |V|} \bigwedge_{0 \le j < \lceil \log(K+1) \rceil} \left(\left(\overline{[i-1,a,j]} \wedge [i,c,j]\right) \Longrightarrow ([i,a,j])\right) \\
&\wedge \bigwedge_{1 \le i \le |V|} \bigwedge_{0 \le j < \lceil \log(K+1) \rceil} \left(([i,a,j] \wedge [i-1,a,j]) \Longrightarrow (\overline{[i,c,j]})\right) \\
&\wedge \bigwedge_{1 \le i \le |V|} \bigwedge_{0 \le j < \lceil \log(K+1) \rceil} \left(\left([i,a,j] \wedge \overline{[i-1,a,j]}\right) \Longrightarrow ([i,c,j])\right) \\
&\wedge \; \mathit{none}\{\, [i,c,\lceil \log(K+1) \rceil] \mid 1 \le i \le |V| \,\} \wedge \bigwedge_{\{u,v\} \in E} \mathit{at\_least\_one}\{\, [u,c,0],[v,c,0] \,\}
\end{aligned}
$$

5.1.2 [GT20] INDEPENDENT SET $\propto$ SAT

INSTANCE: Graph $G=(V,E)$, V=$\{1,\ldots,n\}$, positive integer $K > 0$.
QUESTION: $\exists V' \subseteq V, |V'| \ge K, \forall \{u,v\} \in E : \{u,v\} \not\subseteq V'$?

$$
\begin{aligned}
X &= \{\, [i,a,j] \mid 0 \le i \le |V|, 0 \le j < \lceil \log(K+1) \rceil \,\} \\
&\cup \{\, [i,c,j] \mid 1 \le i \le |V|, 0 \le j \le \lceil \log(K+1) \rceil \,\} \\
F &= \bigwedge_{0 \le i < \lceil \log(K+1) \rceil}^{(K-1)_i=0} \left(\overline{[0,a,i]}\right) \wedge \bigwedge_{0 \le i < \lceil \log(K+1) \rceil}^{(K-1)_i=1} ([0,a,i]) \\
&\wedge \bigwedge_{1 \le i \le |V|} \bigwedge_{0 \le j < \lceil \log(K+1) \rceil} \left(\left(\overline{[i-1,a,j]} \wedge [i,c,j]\right) \Longleftrightarrow ([i,c,j+1])\right) \\
&\wedge \bigwedge_{1 \le i \le |V|} \bigwedge_{0 \le j < \lceil \log(K+1) \rceil} \left(\left(([i-1,a,j] \wedge \overline{[i,c,j]}) \vee (\overline{[i-1,a,j]} \wedge [i,c,j])\right) \Longleftrightarrow ([i,a,j])\right) \\
&\wedge \; \mathit{at\_least\_one}\{\, [i,c,\lceil \log(K+1) \rceil] \mid 1 \le i \le |V| \,\} \wedge \bigwedge_{\{u,v\} \in E} \mathit{at\_most\_one}\{\, [u,c,0],[v,c,0] \,\}
\end{aligned}
$$

This is nearly the same formula as above, except that K is replaced by $K-1$, *at_least_one* is replaced by *at_most_one* and the occurence of an overflow is enforced.

5.2 Calculating

Here a technique will be developed that enables us to implement even "mathematical problems" dealing with numbers only. The most important thing is that the number of variables remains polynomial in the input-size of the problem (linear, in fact!). It is only a more general usage of "numbers" as in the previous section, and will be demonstrated here by the problem SUBSET-SUM ([SP13]):

5.2.1 [SP13] SUBSET SUM $\propto$ SAT

INSTANCE: Finite set $A = \{a_1, \ldots, a_n\}$, size $s(a) \in Z^+$ for each $a \in A$, positive integer B.
QUESTION: $\exists A' \subseteq A : \sum_{a \in A'} s(a) = B$?

$$
\begin{array}{rcl}
X & = & \{\,[i,s,j] \mid 0 \le i \le n, 0 \le j < \lceil \log(B+1) \rceil\,\} \;\cup\; \{\,[i,a,j] \mid 1 \le i \le n, 0 \le j < \lceil \log(B+1) \rceil\,\} \\
& \cup & \{\,[i,c,j] \mid 1 \le i \le n, 0 \le j \le \lceil \log(B+1) \rceil\,\} \;\cup\; \{\,[i] \mid 1 \le i \le n\,\} \\
F & = & \bigwedge\limits_{1 \le i \le n} \bigwedge\limits_{\substack{(a_i)_j = 0 \\ 0 \le j < \lceil \log(B+1) \rceil}} (\overline{[i,a,j]}) \wedge \bigwedge\limits_{1 \le i \le n} \bigwedge\limits_{\substack{(a_i)_j = 1 \\ 0 \le j < \lceil \log(B+1) \rceil}} ([i,a,j]) \\
& \wedge & \bigwedge\limits_{\substack{B_j = 0 \\ 0 \le j < \lceil \log(B+1) \rceil}} (\overline{[0,s,j]}) \wedge \bigwedge\limits_{\substack{B_j = 1 \\ 0 \le j < \lceil \log(B+1) \rceil}} ([0,s,j]) \\
& \wedge & \bigwedge\limits_{1 \le i \le n} \bigwedge\limits_{0 \le j < \lceil \log(B+1) \rceil} \left(\begin{array}{c} ([i] \wedge [i,a,j]) \\ \Longleftrightarrow \\ \left(\left(\left([i-1,s,j] \vee \overline{[i,c,j]}\right) \wedge \left(\overline{[i-1,s,j]} \vee [i,c,j]\right)\right) \Longleftrightarrow ([i,s,j])\right) \end{array} \right) \\
& \wedge & \bigwedge\limits_{1 \le i \le n} \bigwedge\limits_{0 \le j < \lceil \log(B+1) \rceil} \left(\left(\overline{[i-1,s,j]}\right) \Longrightarrow (([i,c,j] \vee ([i,a,j] \wedge [i])) \Longleftrightarrow ([i,c,j+1]))\right) \\
& \wedge & \bigwedge\limits_{1 \le i \le n} \bigwedge\limits_{0 \le j < \lceil \log(B+1) \rceil} (([i-1,s,j]) \Longrightarrow (([i,c,j] \wedge ([i,a,j] \wedge [i])) \Longleftrightarrow ([i,c,j+1]))) \\
& \wedge & \mathit{none}\,\{\,[i,c,\lceil \log(B+1) \rceil] \mid 1 \le i \le n\,\} \;\wedge\; \mathit{none}\,\{\,[i,c,0] \mid 1 \le i \le n\,\} \\
& \wedge & \mathit{none}\,\{\,[n,s,j] \mid 0 \le j < \lceil \log(B+1) \rceil\,\}
\end{array}
$$

There are arrays of $\lceil \log(B+1) \rceil$ boolean variables for every $1 \le i \le n$ with names a (the value of a_i), c (the carry-flags) and s (the actual sum). The array s_0 holds the value B. Additionally, there is a single variable $[i]$ for each $1 \le i \le n$, which is true if and only if a_i has to be an element of A'. Now the first four parts place the binary representations of B and the elements of A into the correct positions. The next three parts specify a correct calculation ($s_i = s_{i-1} - [i] * a_i$, where $[i]$ stands for 0 (1) if it is false (true)), followed by two parts forcing all least "carry-bits" to be zero and disabling any overflow. The last part verifies that the elements from the chosen subset A' (the a_i with $[i]$ set to true) sum up to B ($s_0 = B \Longrightarrow s_n = 0$).

The CNF-representation of the above problem can easily be obtained by repeatedly applying the methods form the third chapter. It should be clear how to program other mathematical problems like, e.g., SUBSET PRODUCT, where one has to specify the type of calculation (e.g. multiplication) on the binary representations first. The rest is then problem-dependent but similar to the techniques proposed in the fourth chapter.

Chapter 6

Heuristics

In this chapter different heuristics for solving SAT-problems are described. They all are part of some Satisfiability solving algorithms using backtracking. Different methods for selecting the next literal in such backtracking-algorithms are presented.

6.1 Useful existing heuristics

It seems to be impossible to describe properties that a "best heuristic" for solving Satisfiability should have. However below some simple properties, which are efficient in "real" implementations of Satisfiability-solvers, are described.

6.1.1 Direct setting of ≤ 1-literal clauses

A very simple technique is to look for the existence of clauses with ≤ 1 literals at every step of Satisfiability-solving. If there is any clause with 0 literals (the empty clause, $\square$), the partial assignment determined so far cannot be extended to any solution, and so backtracking should be invoked (this is the main idea of backtracking). If there is any clause consisting of exactly one literal, this literal should be set in the only correct (i.e., its clause satisfying) way. This, together with the simplification of the rest-formula, leads to a correct reduction of the size of the actual formula at hand, and thus also to an efficient evaluation.

Now let us look at the "useful formulas" *exactly_one* and *at_most_one*. It is easy to see that after setting one literal of the arguments so that it is satisfied, there will be (negative) 1-literal clauses for all remaining literals of the argument-list. These clauses are now selected by the heuristic and any such formula with k literals is set completely in time $O(k)$, after the first of its literals is set satisfied, which is very efficient.

6.1.2 Direct setting of only positive (only negative) literals

If one keeps track of the number of positive and negative occurrences of all actual literals, it is a good idea to take any literal which occurs only positively (only negatively) in the remaining clauses, and set it true (false). In the case of looking for any first solution of a given formula, this is always a correct step, which reduces the formula-size and thus the running-time.

6.1.3 Solving Horn- and 2-CNF-instances directly

If the actual formula is in Horn-form or 2-CNF-form (see 7.1 and 7.2), the corresponding linear-time algorithms ([1],[4]) for solving this special instances should be used, because they are optimal. This will speed up computation by cutting off subtrees of the original search-tree in linear instead of possibly exponential time.

6.2 Interesting new heuristics

6.2.1 Connected selection

Going back to the first implementation of the Hamilton Path problem ([GT39]), it seems to be a good heuristic to set literals whose positions are "close together" in the matrix defined by the variables names. This is because incompatible assignments leed to earlier conflicts (empty clauses) and earlier backtracking, which should result in better runtime behavior. The property "close together" can be defined as follows: At every step of the selection, a literal has to be chosen from an arbitrary unsatisfied clause which has been reduced in size by previous variable-settings, if any such variable exists. If, in addition, in the case of the nonexistence of such variables those variables are preferred which occur in already satisfied clauses, this combined method assures that components of the clause-graph ($F = C_1 \wedge \ldots \wedge C_r, X = \{x_1, \ldots, x_n\} \rightarrow G = (V, E), V = \{C_1, \ldots, C_r\} \cup \{x_1, \ldots, x_n\}, E = \{\,\{x_i, C_j\} \,|\, (x_i \in C_j) \vee (\overline{x_i} \in C_j)\,\}$) are tested separately, which is very efficient. The clause-graph does not first need to be divided into connected components when using the above mentioned method! (This technique corresponds to the well-known technique of program transformation for Prolog, where one evaluates test predicates as early as possible).

6.2.2 Priorities given by implementations

It seems to be a good idea to give priorities to the variables used in an "implementation" such as the implementations described in chapter 4. This will influence the selection of the next variable in solving Satisfiability, such that any other heuristic will have to determine the next variable to be selected among the remaining variables with highest priority (different variables having the same priority are ok). This gives the programmer something like "control" over the evaluation-strategy of the solver (this corresponds to the seminal paper: Algorithm = Logic + Control, [9]). If, for example, there is an implementation of a problem which can be transformed into one of the efficiently solvable formulas with only a polylogarithmic number of variable settings, then priority given to these variables, results in a polynomial-time-algorithm in connection with 6.1.3 !

Chapter 7

Expressiveness

The general expressibility of propositional logic is clear: Every problem in $\mathcal{NP}$ can be described as the problem, whether or not a specific propositional formula is satisfiable. Likewise, every problem in Co-$\mathcal{NP}$ is solvable by answering the question, whether a specific propositional formula is a tautology (or unsatisfiable). Therefore we will now look at subclasses of general propositional logic.

7.1 2-CNF formulas

A propositional formula is in 2-CNF if and only if it is in CNF and has at most 2 literals per clause. It seems hard to imagine any practical problem that can be represented in 2-CNF. One problem that can be formulated in 2-CNF is the following (see Chapter 8 for an important application):

7.1.1 NON-REACHABILITY $\propto$ SAT

INSTANCE: Directed graph $G = (V, A)$, $x, y \in V$.
QUESTION: Is there no directed path from x to y in G ?

$$\begin{aligned} X &= \{ [v] \mid v \in V \} \\ F &= \bigwedge_{\{u,v\} \in A} (([u]) \Longrightarrow ([v])) \\ &\wedge \ ([x]) \wedge \left(\overline{[y]}\right) \end{aligned}$$

The first part implies that any descendant v of a vertex u with $[u]$ set to $true$ will have $[v]$ set to $true$ if F is satisfiable. Thus, the last two parts enforce the non-satisfiability in the case of a directed path from x to y, whereas the following variable setting satisfies F if there is no such a path: set all $[z]$ to true for all descendants z of x and all remaining variables to $false$.

The difficulty with this problem is, that its complement is surely in $\mathcal{P}$ = Co-$\mathcal{P}$, since 2-CNF can be solved in linear time [1], but cannot be used, for example, in the premise of an implication (loosely speaking: "$\overline{unsatisfiable} = tautology \neq satisfiable$").

7.2 Horn formulas

Horn formulas are propositional formulas in CNF with at most one positive literal per clause. They can be tested for satisfiability in linear time [4], and they seem to be of more practical realm than 2-CNF formulas. Below a little example shows how even simple types of knowledge can be represented in Horn-formulas (see [3] for more advanced applications). A little knowledge-base dealing with graph-classes and graph-properties serves as example. Here R is the part of the formula representing the *rules*, F is the part representing the *facts*, and q is the *question* to be shown logically derivable by the above system. Let us first define the example:

$$
\begin{array}{rcl}
X & = & \{\,[non\_homeomorphic\_subgraph, s] \mid s \in \{K_3, K_4, K_5, K_{23}, K_{33}\}\,\} \\
& \cup & \{\,[connected, i] \mid 1 \le i \le 3\,\} \cup \{\,[outerplanar, i] \mid 1 \le i \le 3\,\} \cup \{\,[genus, i] \mid 0 \le i \le 1\,\} \\
& \cup & \{\,[separable, i] \mid 1 \le i \le 2\,\} \cup \{\,[tree, i] \mid 1 \le i \le 3]\,\} \cup \{\,[partial\_tree, i] \mid 1 \le i \le 5\,\} \\
& \cup & \{\,[tree\_width, i] \mid 1 \le i \le 5\,\} \cup \{\,[almost\_tree, i] \mid 1 \le i \le 4\,\} \cup \{\,[connected], [planar]\,\} \\
& \cup & \{\,[outerplanar], [Halin], [torodial], [acyclic], [forest], [tree], [series\_parallel], [cactus]\,\} \\
R & = & (([non\_homeomorphic\_subgraph, K_3]) \Longrightarrow ([non\_homeomorphic\_subgraph, K_4])) \\
& \wedge & (([non\_homeomorphic\_subgraph, K_4]) \Longrightarrow ([non\_homeomorphic\_subgraph, K_5])) \\
& \wedge & (([non\_homeomorphic\_subgraph, K_4]) \Longrightarrow ([non\_homeomorphic\_subgraph, K_{33}])) \\
& \wedge & (([non\_homeomorphic\_subgraph, K_{23}]) \Longrightarrow ([non\_homeomorphic\_subgraph, K_{33}])) \\
& \wedge & (([non\_homeomorphic\_subgraph, K_{23}]) \Longrightarrow ([non\_homeomorphic\_subgraph, K_5])) \\
& \wedge & \left(([planar]) \Longleftrightarrow \left(\begin{array}{c} [non\_homeomorphic\_subgraph, K_5] \\ \wedge[non\_homeomorphic\_subgraph, K_{33}] \end{array}\right)\right) \\
& \wedge & \left(([outerplanar]) \Longleftrightarrow \left(\begin{array}{c} [non\_homeomorphic\_subgraph, K_4] \\ \wedge[non\_homeomorphic\_subgraph, K_{23}] \end{array}\right)\right) \\
& \wedge & (([outerplanar, 1]) \Longleftrightarrow ([outerplanar])) \wedge (([outerplanar, 1]) \Longrightarrow ([outerplanar, 2])) \\
& \wedge & (([outerplanar, 2]) \Longrightarrow ([outerplanar, 3])) \wedge (([outerplanar, 3]) \Longrightarrow ([planar])) \\
& \wedge & (([Halin]) \Longrightarrow ([outerplanar, 2])) \wedge (([Halin]) \Longrightarrow ([connected, 3])) \\
& \wedge & (([connected, 1]) \Longleftrightarrow ([connected])) \wedge (([connected, 2]) \Longrightarrow ([connected, 1])) \\
& \wedge & (([connected, 3]) \Longrightarrow ([connected, 2])) \wedge (([planar]) \Longleftrightarrow ([genus, 0])) \\
& \wedge & (([torodial]) \Longleftrightarrow ([genus, 1])) \wedge (([genus, 0]) \Longrightarrow ([genus, 1])) \\
& \wedge & (([acyclic]) \Longrightarrow ([forest])) \wedge (([forest] \wedge [connected]) \Longrightarrow ([tree])) \\
& \wedge & (([series\_parallel]) \Longrightarrow ([separable, 2])) \wedge (([tree]) \Longrightarrow ([separable, 1])) \\
& \wedge & (([tree]) \Longleftrightarrow ([tree, 1])) \wedge (([tree, 1]) \Longrightarrow ([partial\_tree, 1])) \\
& \wedge & (([tree, 2]) \Longrightarrow ([partial\_tree, 2])) \wedge (([tree, 3]) \Longrightarrow ([partial\_tree, 3])) \\
& \wedge & (([partial\_tree, 1]) \Longrightarrow ([partial\_tree, 2])) \wedge (([partial\_tree, 2]) \Longrightarrow ([partial\_tree, 3])) \\
& \wedge & (([tree\_width, 1]) \Longleftrightarrow ([partial\_tree, 1])) \wedge (([tree\_width, 2]) \Longleftrightarrow ([partial\_tree, 2])) \\
& \wedge & (([tree\_width, 3]) \Longleftrightarrow ([partial\_tree, 3])) \wedge (([cactus]) \Longleftrightarrow ([almost\_tree, 1])) \\
& \wedge & (([almost\_tree, 1]) \Longrightarrow ([tree\_width, 2])) \wedge (([almost\_tree, 2]) \Longrightarrow ([tree\_width, 3])) \\
& \wedge & (([Halin]) \Longrightarrow ([partial\_tree, 3])) \\
& \wedge & (([outerplanar, 1]) \Longrightarrow ([partial\_tree, 2])) \wedge (([outerplanar, 2]) \Longrightarrow ([partial\_tree, 5])) \\
& \wedge & (([series\_parallel]) \Longleftrightarrow ([non\_homeomorphic\_subgraph, K_4])) \\
& \wedge & (([partial\_tree, 2]) \Longleftrightarrow ([non\_homeomorphic\_subgraph, K_4]))
\end{array}
$$

If one knows that a graph has no subgraph homeomorphic to K_{23}, and one wants to know whether such a graph is torodial, one has to check whether T given by

$$T = R \wedge F \wedge (\overline{q})\,,\; F = ([non\_homeomorphic\_subgraph, K_{23}]),\; q = [torodial]$$

is unsatisfiable. This is easy, because the above formula can be translated into Horn form with the techniques form chapter 3.3. The satisfiability (and also unsatisfiability) can be checked in linear time. The above formula is unsatisfiable because the attribute $[torodial]$ is a logical consequence of $[non\_homeomorphic\_subgraph, K_{23}]$, and the presence of $\left(\overline{[torodial]}\right)$ results in unsatisfiability. Since R is always satisfiable (set all variables to false), the formula remains satisfiable if testing a term q, that is not a logical consequence.

Chapter 8

Internal variables

In this chapter the technique of using "internal variables" is described. Here "internal variables" are newly created. They help to reduce the cardinality of the clause set (already used in 3.2).

8.1 Efficient implementation of *at_most_one* and *exactly_one*

The definition

$$at\_most\_one(l_1, \ldots, l_x) = \bigwedge_{1 \le i < j \le x} (\overline{l_i} \vee \overline{l_j})$$

from chapter 2 has one major drawback: It has $O(x^2)$ clauses! Here a method for programming *at_most_one* in 2CNF- and Horn-form with $< 3x$ clauses is described. The only *price* to be paid for this transformation is the introduction of $x - 2$ new variables.

$$\begin{aligned} X &= \{\, l_1, \ldots, l_x \,\} \cup \{\, r_2, \ldots, r_{x-1} \,\} \\ F &= (\overline{l_1} \vee \overline{l_2}) \wedge (l_1 \Rightarrow r_2) \wedge (l_2 \Rightarrow r_2) \\ &\wedge \bigwedge_{3 \le j \le x} (\overline{r_{j-1}} \vee \overline{l_j}) \wedge \bigwedge_{3 \le j < x} (r_{j-1} \Rightarrow r_j) \wedge \bigwedge_{3 \le j < x} (l_j \Rightarrow r_j) \end{aligned}$$

In this formula r_j is *true* if, and only if, exactly one of the literals $l_1, \ldots, l_j$ is set to *true*. Thus, the last clause $(\overline{r_{x-1}} \vee \overline{l_x})$ verifies the property *at_most_one* for $\{l_1, \ldots, l_x\}$.

Note that in using this efficient implementation of *at_most_one*, the formula *exactly_one* is also implementable with $\le 3x$ clauses by its definition (chapter 2).

Chapter 9

Summary

This paper provides a rich pool of easy and hard instances for Satisfiability-solving algorithms by introducing a simple way to "progam" any problem from $\mathcal{NP}$ directly in propositional logic.

The use of operations like *at_most_one* makes programming in propositional logic more easy and readable. The techniques of how to deal with numbers described in chapter 5 opens the big area of mathematical problems for Satisfiability and helps sometimes to get more compact (but less readable) representations of progams. Chapter 6 gives at least some hints how implementations of problems in Satisfiability can lead to new algorithmic/heuristic ideas for Satisfiability solvers.

These days there are quite a few International contests comparing the power of various algorithms for solving SAT. Perhaps this paper gives some hints for the generation of meaningful (not only random) formulas, that may (at least additionally) serve as basis for such comparisons.

Three "little" open problems at the end:

1. Is there any (Co-)Horn-implementation for PLANARITY (4.1.3), which has a number of clauses polynomial in $|V|$ for input graph G=(V,E)?

2. How can the subclass of problems from $\mathcal{P}$, which admit (Co-)Horn-implementations of polynomial size, be characterized?

3. Is the above subclass of $\mathcal{P}$ perhaps the whole class $\mathcal{P}$?

Bibliography

[1] Aspvall, B., Plass, M.F., Tarjan, R.E., A Linear-time Algorithm for Testing the Truth of certain quantified boolean Formulas, Information Processing Letters, Vol. 8(3), 1979, pp. 121-123.

[2] Cook, S.A., The complexity of theorem-proving procedures, Third annual ACM Symposium on Theory of Computing, 1971, pp. 151-158.

[3] de Kleer, An assumption-based TMS, Journal Artificial Intelligence, Vol. 28(2), 1986, pp. 127-162.

[4] Dowling, W.F., Gallier, J.H., Linear-time Algorithms for testing the Satisfiability of propositional Horn formulae, Journal Logic Programming, 1984, pp. 267-284.

[5] Dubois, O., On the r,s-SAT Satisfiability problem and a conjecture of Tovey, Discrete Applied Mathematics, Vol. 26, 1990, pp. 51-60.

[6] Garey, M.R., Johnson, D.S., Computers and Intractability: A Guide to the theory of NP-Completeness, San Francisco, CA: Freeman, 1979.

[7] Jayakumar, K., Thulasiraman, K., Swamy, M.N., "Planar Embedding: Linear-Time Algorithms for Vertex Placement and Edge Ordering", IEEE Transactions on Circuits and Systems, Vol. 35(3), 1988, pp. 334-344.

[8] Knuth, D.E, Nested Satisfiability, Acta Informatica, Vol. 28, 1990, pp. 1-6.

[9] Kowalski, "Algorithm = Logic + Control", Communications of the ACM, Vol. 22, 1979, pp. 424-436.

[10] Lichtenstein, D., Planar Formulae and their uses, SIAM Journal Computing, Vol. 11(2), 1982, pp. 329-343.