Contents

Chapter 1

WHAT SAT-SOLVERS CAN AND CANNOT DO

Eugene Goldberg
Cadence Berkeley Labs, USA
egold@cadence.com

Abstract This chapter consists of two parts. In the first part we show that resolution based SAT-solvers cannot be scalable on real-life formulas unless some extra information about formula structure is known. In the second part we introduce a new way of satisfiability testing that may be used for designing more efficient and "intelligent" SAT-algorithms that will be able to take into account formula structure.

In the last few years SAT-solvers have considerably improved their performance. As a result, the size of the CNF formulas that can be solved by state-of-the-art SAT-solvers [21, 23, 16, 8] in a reasonable time has dramatically increased. This success has lead to euphoria that reminds many people working in formal verification of early optimism caused by the appearance of BDDs [4]. However, enthusiasts forget that even though SAT-solvers can sometimes solve surprisingly large formulas, they are very far from being scalable (which is the same problem that made people less optimistic about BDDs).

In this chapter, we will try to give a more realistic estimation of the capabilities of SAT-solvers. The chapter is based on the results described in [10–12] and consists of two parts. The main point of the first part is that a SAT-solver cannot be scalable unless it is provided with some information about the structure of the CNF formula to be tested for satisfiability. In this part, we consider a class of formulas describing equivalence checking of combinational circuits that have a common specification (CS). A CS S of Boolean circuits N_1 and N_2 is just a circuit of multi-valued gates called blocks. Either Boolean circuit is obtained from S by replacing each block of S with its binary implementation. We show that there is a short resolution proof that N_1 and N_2 are equivalent however finding this proof by a deterministic algorithm is most likely infeasible unless a CS of N_1 and N_2 is known. On the one hand, it is bad news. This result means that SAT-algorithms cannot be scalable on equivalence checking CNF formulas (that are important from a practical point of view) even though they have short resolution proofs of unsatisfiability and so are very "easy". On the other hand, this is good news because one can have an efficient algorithm of equivalence checking if a CS of N_1 and N_2 is known. In other words, addressing the question implied by the title of this chapter one can say that SAT-solvers cannot be scalable if no information about high-level structure of formulas is provided.

The result above implies that it is crucial for a SAT-solver to be able to take into account structural properties of formulas. The problem is that the existing SAT-solvers are based on the variable splitting paradigm introduced in the DPLL procedure [7]. During variable splitting a CNF formula is "mutilated" and its subtle structure is usually destroyed. In the second part of this chapter, we introduce a new procedure of satisfiability testing based on the notion of a stable set of points (SSP). It turns out that to prove that a CNF formula F is unsatisfiable it is sufficient to show that F evaluates to 0 (i.e. false) on a set of points called a stable set. In a sense, proving the unsatisfiability of a CNF formula by constructing its SSP can be viewed as "verification" by "simulation".

In general, SSPs are much smaller than the set of all possible assignments but the size of SSPs grows exponentially in the number of variables. So building a monolithic SSP point-by-point can not be used as the basis for designing efficient universal SAT-solvers. We describe two ways of using SSPs. First way is to compute an SSP modulo symmetries of the formula to be tested for satisfiability. In that case, even point-by-point computation of SSPs modulo symmetry can be efficient for highly symmetric formulas. Another way of using SSPs is to replace computing a monolithic SSP with constructing a sequence of much smaller SSPs of "limited" stability. Each such an SSP is stable if "movements" in some directions are forbidden.

1. Hard Equivalence Checking CNF formulas

1.1 Introduction

Since the general resolution system is the basis of almost all practical SAT-solvers, it has been the focus of attention for a long time. In the ground-breaking paper by Haken [13] it was shown that there is a class of CNF formulas for which only exponential size proofs are possible. (In the first part of this chapter we consider only unsatisfiable CNF formulas.) However, the impressive results of state-of-the-art SAT-solvers like Grasp, Sato, Chaff, BerkMin suggest that for the majority of CNF formulas one encounters in practice there should be short resolution proofs of their unsatisfiability. So a natural question to ask is whether the fact that a class of CNF formulas has short resolution proofs means that there is an algorithm that can find these short proofs or proofs that are "close" to them in length. (In complexity theory this question is posed as "whether the general resolution system is automatizable". Studying the automatizability of proof systems was started in [2]. In [18] some results on automatizability of general resolution were obtained.)

The objective of the first part of this chapter is to show that there is a class of CNF formulas that have very short resolution proofs in general resolution that are most likely very hard for a deterministic SAT-algorithm. These formulas specify equivalence checking of Boolean circuits and so they are very important from a practical point of view. This result means that the power of resolution based SAT-solvers is quite limited even for practical formulas that have provably short resolution proofs. The good news is that one can have an efficient SAT-algorithm for solving this class of formulas if some information about the structure of short proofs is provided.

The class of formulas mentioned above describe equivalence checking of circuits having a common "specification". Let N_1 and N_2 be two functionally Boolean circuits with a common specification (CS) S. The CS S is just a circuit of multi-valued gates further referred to as blocks such that N_1 (or N_2) can be obtained from S by replacing each block G of S with its implementation $I_1(G)$ (or $I_2(G)$). The circuit $I_1(G)$ (or $I_2(G)$) implements the multi-output Boolean function obtained from the truth table of G after encoding the values of multi-valued variables with binary codes.

The problem of equivalence checking of N_1 and N_2 can be easily reduced to that of testing the unsatisfiability of a CNF formula (see section 1.1.3). Let S consist of n blocks. Let F be a CNF specifying equivalence checking of N_1 and N_2. We show that the unsatisfiability of F can be proven in general resolution in $d * n * 3^{6p}$ resolution steps. Here d is a constant and p is the size of the largest block G of the CS S (in terms of the number of gates one needs to implement G in N_1 and N_2). In particular, if p is bounded by a constant then we get a class of CNF formulas (in the paper it is denoted by $M(p)$) that has linear size resolution proofs. The parameter p is called the granularity of the specification S.

In spite of the fact that formulas from $M(p)$ have short resolution proofs of unsatisfiability there are good reasons to believe that there does not exist an efficient SAT-algorithm for finding such proofs. Let F be a formula $M(p)$ specifying equivalence checking of circuits N_1 and N_2 with a CS S. Let assume that the CS S is not known. On the one hand, the problem of finding S (or a good approximation of S) is most likely NP-hard. On the other hand, the short resolution proofs mentioned above are closely related to CSs of N_1 and N_2. So given such a short proof of equivalence of N_1 and N_2 one could recover a "good" CS from this proof. Hence the existence of an efficient procedure for finding a short proof of equivalence would mean that there is an efficient algorithm for solving an NP-hard problem.

As we mentioned above the good news is that a formula F of $M(p)$ can be efficiently solved by a deterministic algorithm if some extra information is provided. This extra information is a CS S of N_1 and N_2 whose equivalence checking the formula F specifies. (Namely, one just needs to know the assignment of gates of N_1 and N_2 to blocks of S. No other information about S is needed. In particular, one needs neither any knowledge of the functionality of blocks of S nor the knowledge of binary encodings used when producing N_1 and N_2 from S.) We formulate a specification aware algorithm of checking the unsatisfiability formulas from $M(p)$ that has the same complexity as resolution proofs. That is it solves the formulas of $M(p)$ in linear time.

The first part of this chapter is structured as follows. In Section 1.1.2 we introduce the notion of a CS of Boolean circuits that plays a key role in the following exposition. Section 1.1.3 describes a common way of reducing equivalence checking to SAT. In Section 1.1.4 we introduce a class $M(p)$ of CNF formulas encoding equivalence checking of Boolean circuits with a CS of granularity p. We also describe the general resolution proof system. Section 1.1.5 describes computation of existentially implied functions that is used in Section 1.1.6. In the latter, we proof the main result of the first part of this chapter about the complexity of formulas from $M(p)$ in general resolution. In Section 1.1.7 and 1.1.8 we discuss the complexity of formulas $M(p)$ for deterministic resolution based algorithms. In Section 1.1.7 we give reasons why formulas from $M(p)$ should be hard for deterministic SAT-algorithms that do not have any knowledge of a CS of the circuits checked for equivalence. In Section 1.1.8 we describe an efficient resolution based SAT-algorithm for equivalence checking of circuits with a known CS. In Section 1.1.9 we show experimentally that formulas from $M(p)$ are hard for existing SAT-solvers while a specification aware algorithm easily solves them. In Section 1.1.10 some conclusions are made.

1.2 Common Specification of Boolean Circuits

In this section, we introduce the notion of a common specification of Boolean circuits. Let S be a combinational circuit of multi-valued blocks (further referred to as a ***specification***) specified by a directed acyclic graph H. (An example of specification is shown in Fig. 1.1a.) The sources and sinks of H correspond to primary inputs and outputs of S. Each non-source node of H corresponds to a multi-valued block computing a multi-valued function of multi-valued arguments. Each node n of H is associated with a ***multi-valued variable*** V. If n is a source of H , then the corresponding variable specifies values taken by the corresponding primary input of S. If n is a non-source node of S then the corresponding variable describes the values taken by the output of the block specified by n. If n is a source (respectively a sink), then the corresponding variable is called a ***primary input variable*** *(*respectively ***primary output variable)***. We will use the notation C=G(A,B) to indicate that a) the output of a block G is associated with a variable C; b) the function computed by the block G is G(A,B); c) only two nodes of H are connected to the node n in H and these nodes are associated with variables A and B.

Denote by $D(A)$ the ***domain*** of a variable A associated with a node of H. The value of $|D(A)|$ is called the ***multiplicity*** of A. If the multiplicity of every variable A of S is equal to 2 then S is a ***Boolean circuit***.

Now we describe how a Boolean circuit N can be produced from a specification S by encoding the multi-valued variables. Let $D(A) = \{a_1, \ldots, a_t\}$ be the domain of a variable A of S. Denote by $q(A)$ ***a Boolean encoding*** of the values of $D(A)$ that is a mapping $q : D(A) \rightarrow \{0,1\}^m$. Denote by $length(q(A))$ the number of bits in q that is the value of m. The value of $q(a_i)$, $a_i \in D(A)$ is called the ***code*** of a_i. Given an encoding q of length m of a variable A associated with a block of S, denote by $v(A)$ the set of m ***coding Boolean variables***.

EXAMPLE 1 *Let B be a multi-valued variable and $D(B) = \{b_1, b_2, b_3, b_4\}$. Then the multiplicity of the variable B is 4. Let a mapping q be specified by the following expressions $q(b_1) = 01, q(b_2) = 11, q(b_3) = 10, q(b_4) = 00$. Then q specifies an encoding of the values of B of $length(q(B))$ equal to 2. The set of coding variables $v(B) = \{q_1, q_2\}$ consists of two Boolean variables. The Boolean vector* 01 *where $q_0 = 0$,$q_1 = 1$ is the code of b_1 under the encoding q.*

In the following exposition we make the assumptions below.

ASSUMPTION 1 *Each gate of a Boolean circuit and each block of a specification has two inputs and one output.*

ASSUMPTION 2 *The multiplicity of each primary input (or output) variable of a specification is a power of 2.*

ASSUMPTION 3 *If V is a primary input (or output) variable of a specification, then length($q(A)$) = $log_2(|D(A)|)$*

ASSUMPTION 4 *If a_1 and a_2 are values of a variable A of a specification and $a_1 \neq a_2$, then $q(a_1) \neq q(a_2)$.*

ASSUMPTION 5 *If A and B are two different variables of a specification, then $v(A) \cap v(B) = \emptyset$.*

REMARK 1 *From Assumptions 2, 3 and 4 it follows that if A is a primary input (or output) variable, a mapping $q : D(A) \rightarrow \{0,1\}^m$ is bijective. In particular, any assignment to the variables of $v(A)$ is a code of some value $a \in D(A)$.*

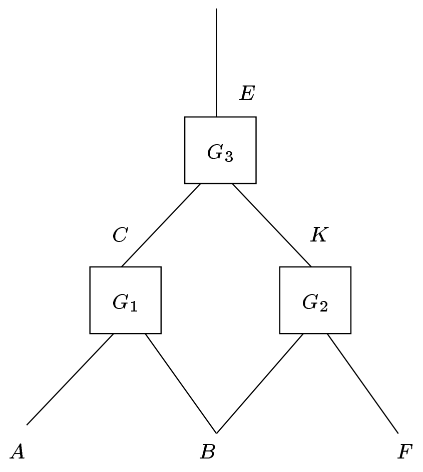
DEFINITION 1 *Given a Boolean circuit I, denote by Inp(I) (respectively Out(I)) the set of variables associated with primary inputs (respectively primary outputs) of I.*

DEFINITION 2 *Let X_1 and X_2 be sets of Boolean variables and $X_2 \subseteq X_1$. Let y be an assignment to the variables of X_1. Denote by proj(y, X_2) the* ***projection*** *of y on X_2 i.e. the part of y that consists of the assignments to the variables of X_2.*

EXAMPLE 2 *Let $X_1 = \{x_1, x_2, x_3, x_4, x_5\}$ and $X_2 = \{x_1, x_3, x_5\}$ that is $X_2 \subseteq X_1$. Let y be the assignment $(x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0, x_5 = 0)$ to the variables of X_1. Then proj(y, X_2) is equal to $(x_1 = 0, x_3 = 1, x_5 = 0)$.*

DEFINITION 3 *Let C=G(A,B) be a block of specification S. Let $q(A)$, $q(B)$, $q(C)$ be encodings of variables A,B, and C respectively. A Boolean circuit I is said to* ***implement the block G*** *if the following three conditions hold:*

- *The set Inp(I) is a subset of $v(A) \cup v(B)$.*
- *The set Out(I) is equal to $v(C)$.*
- *If the set of values assigned to $v(A)$ and $v(B)$ form codes $q(a)$ and $q(b)$ respectively where $a \in D(A)$, $b \in D(B)$, then $I(z')$=$q(c)$ where*



(a)

$C = G_1(A, B)$

A	B	C
a_0	b_0	c_0
a_0	b_1	c_1
a_0	b_2	c_1
a_0	b_3	c_0
a_1	b_0	c_1
a_1	b_1	c_2
a_1	b_2	c_2
a_1	b_3	c_0

(b)

$q_1(C) = I_1(q_1(A), q_1(B))$

$q_1(A)$	$q_1(B)$		$q_1(C)$	
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	0	0

(c)

$q_2(C) = I_2(q_2(A), q_2(B))$

$q_2(A)$	$q_2(B)$		$q_2(C)$		
0	0	0	1	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	0	0	1
1	1	0	0	0	1
1	1	1	1	0	0

(d)

Figure 1.1. A specification and the functionality of two implementations of block G_1

z' is the projection of the assignment z=$(q(a),q(b))$ on Inp(I), $I(z')$ is the value taken by I at z', and c=$G(a,b)$

EXAMPLE 3 *In Fig. 1.1a a specification of three blocks is shown. The functionality of two different implementations of the block C=$G_1(A,B)$ (Fig 1.1b) is shown in Fig. 1.1c and 1.1d. Here $D(A)$=$\{a_0,a_1\}$, $D(B)$=$\{b_0,b_1,b_2,b_3\}$ and $D(C)$=$\{c_0,c_1,c_2\}$. Since A and B are primary input variables they are encoded with a minimum length code and $q_1(A)$=$q_2(A)$ and $q_1(B)$=$q_2(B)$ where $q_1(a_0)$=0, $q_1(a_1)$=1, $q_1(b_0)$=00, $q_1(b_1)$=01, $q_1(b_2)$=10, $q_1(b_3)$=11. Finally, the encodings $q_1(C)$ and $q_2(C)$ are $q_1(c_0)$=00, $q_1(c_1)$=10, $q_1(c_2) = 01$ and $q_2(c_0)$=100, $q_2(c_1)$=010, $q_2(c_2)$=001.*

REMARK 2 *The reason why Inp(I) in Definition 3 may not include all the variables of $v(A)$ and/or $v(B)$ is that the function $G(A,B)$ may not distinguish some values of A or B. ($G(A,B)$ does not distinguish, say, values $a_1, a_2 \in D(A)$, if for any $b \in D(B)$, $G(a_1,b) = G(a_2,b)$.) So to implement $G(A,B)$ the circuit I may need only a subset of variables of $v(A) \cup v(B)$. This situation is illustrated in Fig. 1.2. Due to the fact that some values of the variable C are indistinguishable by G_2, only two outputs of the implementation block $I(G_1)$ (out of the three) are connected to the inputs of $I(G_2)$. This said, henceforth, for the sake of simplicity, we will write $I(q(a),q(b))$ meaning $I(q'(a),q'(b))$, $q'(a)$= proj($q(a)$,Inp(I)) and $q'(b)$=proj($q(b)$,Inp(I)).*

DEFINITION 4 *Let S be a multi-valued circuit. A Boolean circuit N is said to* ***implement the specification S****, if it is built according to the following two rules.*

- *Each block G of S is replaced with an implementation I of G.*
- *Let the output of block G_1 (specified by variable C) be connected to an input of block G_2 (specified by the same variable C) in S. Then the outputs of the circuit I_1 implementing G_1 are properly connected to inputs of circuit I_2 implementing G_2. Namely, the primary output of I_1 specified by a Boolean variable $q_i \in v(C)$ is connected to the input of I_2 specified by the same variable of $v(C)$ if $q_i \in$ Inp(I_2).*

Fig. 1.2 gives an example of a specification (Fig. 1.2*a*) and its implementation (Fig. 1.2*b*).

REMARK 3 *Let N be an implementation of a specification S. Let p be the largest number of gates used in an implementation of a multi-valued*

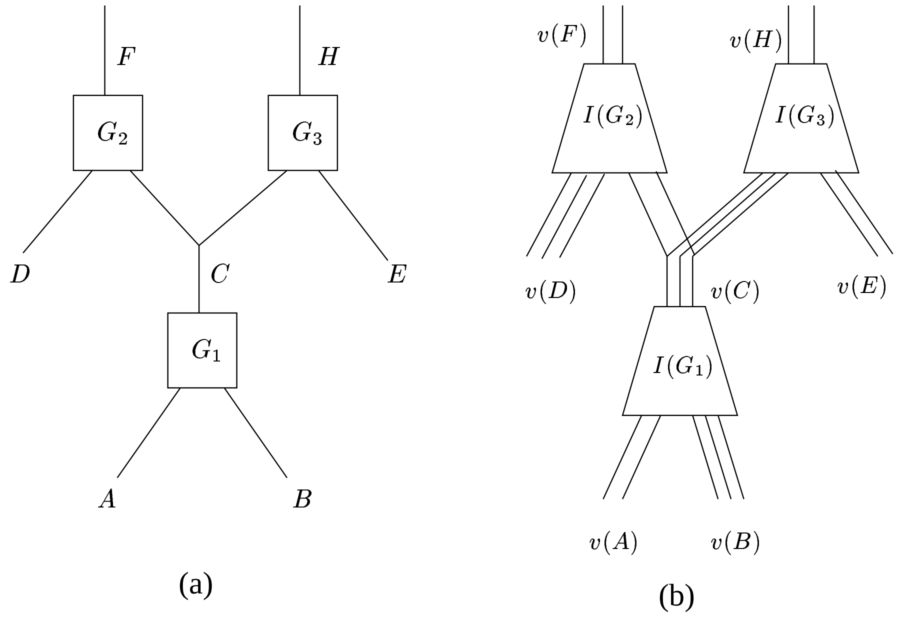


Figure 1.2. A specification and its implementation

block of S in N. We will say that S is a specification of ***granularity*** p *for N.*

Definition 5 *The* ***topological level*** *of a block G in a specification S is the length of the longest path from a primary input of S to G. (The length of a path is measured in the number of blocks on it. The topological level of a primary input is assumed to be 0.) Denote by* ***level(G)*** *the topological level of G in S.*

Remark 4 *Let N be an implementation of a specification S. From Remark 1 it follows that for any value assignment h to the input variables of N there is a unique set of values $(x_1,\dots,x_k)$, where $x_i \in D(X_i)$ such that $h{=}(q(x_1),\dots,q(x_k))$. That is there is one-to-one correspondence between assignments to primary inputs of S and N. The same applies to primary outputs of S and N.*

Definition 6 *Let N be an implementation of S. Given a Boolean vector y of assignments to the primary inputs of N, the corresponding vector $Y{=}(x_1,..,x_k)$ such that $y{=}(q(x_1),\dots,q(x_k))$ is called the* ***pre-image*** *of y.*

Proposition 1 *Let N be a circuit implementing specification S. Let $I(G)$ be the implementation of a block $C{=}G(A,B)$ of S in N. Let y be*

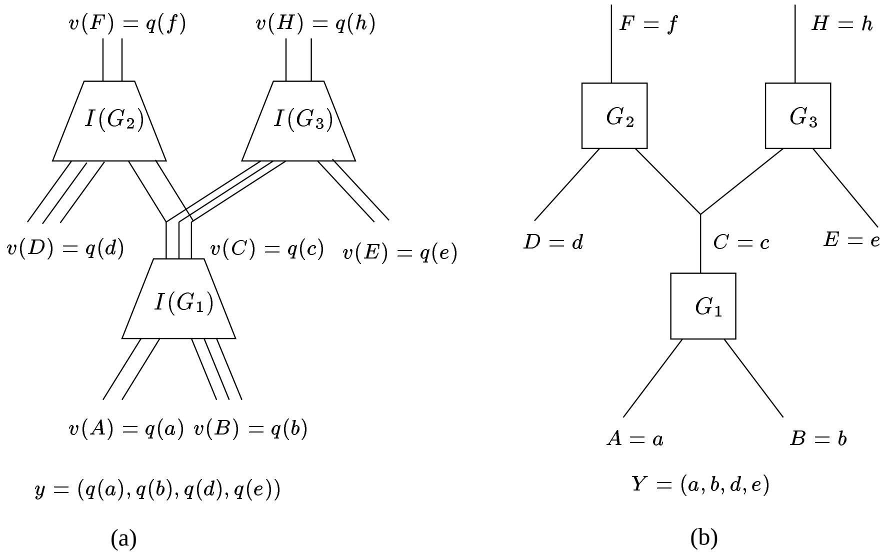


Figure 1.3. An illustration to Proposition 1

a value assignment to the primary input variables of N and Y be the pre-image of y. Then the values taken by the primary outputs of $I(G)$ (under the assignment y to the inputs of N) form the code $q(c)$ of a value c, $c \in D(C)$. The latter is the value taken by the output of G when the inputs of S take the values specified by Y.

Proofs of Proposition 1 and the following Proposition 2 are simple and so we omit them. Instead, we explain Proposition 1 in Fig. 1.3. Suppose that y is an assignment to the primary input variables of the Boolean circuit (Fig. 1.3*a*) that is an implementation of the specification shown in Fig. 1.3*b*. According to Remark 4, y can be represented as $(q(a), q(b), q(d), q(e))$ where a, b, d, e are values of the variables A, B, D, E of the specification respectively. The pre-image of y is the vector $Y = (a, b, d, e)$. Then the outputs of gates G_1,G_2 and G_3 take values $c = G_1(a, b)$,$f = G_2(d, c)$ and $h = G_3(c, e)$ respectively. Since $I(G_1)$,$I(G_2)$ and $I(G_3)$ are implementations of G_1,G_2,G_3 respectively, their outputs take values $q(c)$,$q(f)$ and $q(h)$ respectively.

PROPOSITION 2 *Let N_1, N_2 be circuits implementing a specification S. Let each primary input (or output) variable X of S have the same encoding in N_1 and N_2. Then Boolean circuits N_1 and N_2 are functionally equivalent.*

DEFINITION 7 *Let N_1, N_2 be two functionally equivalent Boolean circuits. Let N_1, N_2 implement a specification S so that for every primary* *input (output) variable X encodings $q_1(X)$ and $q_2(X)$ (used when producing N_1 and N_2 respectively) are identical. Then S is called* ***a common specification*** *(CS) of N_1 and N_2.*

ASSUMPTION 6 *Let S be a CS of N_1,N_2 and C be a variable of S. We will assume that $v_1(C)$ $=v_2(C)$ if C is a primary input variable and $v_1(C) \cap v_2(C) = \emptyset$ otherwise.*

DEFINITION 8 *Let S be a CS of N_1,N_2. Let p_1 (respectively p_2) be the granularity of S with respect to N_1 (respectively N_2). Then we will say that S is a CS of N_1,N_2 of* ***granularity*** *p = max(p_1,p_2).*

DEFINITION 9 *Given two functionally equivalent Boolean circuits N_1, N_2, S is called the* ***finest common specification*** *if it has the smallest granularity p among all the CSs of N_1 and N_2.*

1.3 Equivalence Checking as SAT

In this section, we recall a common way of reducing equivalence checking to the satisfiability problem.

DEFINITION 10 *A disjunction of literals of Boolean variables not containing two literals of the same variable is called a* ***clause****. A conjunction of clauses is called a* ***conjunctive normal form*** *(CNF).*

DEFINITION 11 *Given a CNF F,* ***the satisfiability problem*** *(SAT) is to find a value assignment to the variables of F for which F evaluates to 1 (also called a* ***satisfying assignment****) or to prove that such an assignment does not exist. A clause K of F is said to be* ***satisfied*** *by a value assignment y if $K(y) = 1$. If $K(y) = 0$, the clause K is said to be* ***falsified*** *by y.*

The standard conversion of an equivalence checking problem into an instance of SAT is performed in two steps. Let N_1 and N_2 be Boolean circuits to be checked for equivalence. At the first step of this conversion, a circuit M called ***a miter*** [3] is formed from N_1 and N_2. The miter M is obtained by 1) identifying the corresponding primary inputs of N_1 and N_2; 2) XORing each pair of corresponding primary outputs of N_1 and N_2; 3) ORing the outputs of the added XOR gates. So the miter of N_1 and N_2 evaluates to 1 if and only if for some input assignment a primary output of N_1 and the corresponding output of N_2 evaluate to different values. Therefore, the problem of checking the equivalence of N_1 and N_2 is equivalent to testing the satisfiability of the miter of N_1 and N_2.

At the second step of conversion, the satisfiability of the miter is reduced to that of a CNF formula F. This formula is a conjunction of CNF formulas F_1,..,F_n specifying the functionality of the gates of M and a one-literal clause that is satisfied only if the output of M is set to 1. The CNF F_i specifies the i-th gate g_i of M. Any assignment to the variables of F_i that is inconsistent with the functionality of g_i falsifies a clause of F_i (and vice versa, a consistent assignment satisfies all the clauses of F_i.) For instance, the AND gate $y{=}x_1x_2$ is specified by the following three clauses $\sim x_1 \vee \sim x_2 \vee y$, $x_1 \vee \sim y$, $x_2 \vee \sim y$.

1.4 Class $M(p)$ and general resolution

In this short section we formally define the class of equivalence checking formulas we consider in the first part of this chapter. Besides, we describe the general resolution system.

DEFINITION 12 *Given a constant p, a CNF formula F is a member of the* ***class M(p)*** *if and only if it satisfies the following two conditions.*

- *F is the CNF formula (obtained by the procedure described in Section 1.1.3) specifying the miter of a pair of functionally equivalent circuits N_1,N_2.*

- *N_1,N_2 has a CS of granularity p' where $p' \leq p$.*

DEFINITION 13 *Let K and K' be clauses having opposite literals of a variable (say variable x) and there is only one such variable. The* ***resolvent*** *of K , K' in variable x is the clause that contains all the literals of K and K' but the positive (i.e. literal x) and negative (i.e. literal $\sim x$) literals of x. The operation of producing the resolvent of K and K' is called* ***resolution****.*

DEFINITION 14 ***General resolution*** *is a proof system of propositional logic that has only one inference rule. This rule is to resolve two existing clauses to produce a new one. Given a CNF formula F, a proof L(F) of unsatisfiability of F in the general resolution system consists of a sequence of resolutions resulting in the derivation of an* ***empty clause*** *(i.e. a clause without literals).*

General resolution is complete. This means that given an unsatisfiable formula F there is always a sequence of resolutions $L(F)$ in which an empty clause is derived.

1.5 Computation of existentially implied functions

In this section, we introduce the notion of existentially implied functions that is used in Section 1.1.6 in the definitions of filtering and correlation functions.

DEFINITION 15 *Let $F(X_1, X_2)$ be a Boolean function where X_1 and X_2 are sets of Boolean variables. The function $H(X_2)$ is called* ***existentially implied*** *by F if*

- *$F(X_1,X_2) \rightarrow H(X_2)$*
- *if $H(z)$=1 where z is an assignment to the variables of X_2, then there is an assignment y to the variables of X_1 such that $F(y,z)$=1.*

REMARK 5 *Given a function $F(X_1,X_2)$, the function $H(X_2)$ existentially implied by F is unique. It can be obtained from F by existentially quantifying away the variables of X_1.*

PROPOSITION 3 *Let $F(X_1, X_2)$ and $H(X_2)$ be CNF formulas where $H(X_2)$ consists of all the clauses depending only on variables from X_2 that can be derived from $F(X_1,X_2)$ by resolution. Then $H(X_2)$ is existentially implied by $F(X_1,X_2)$.*

Proof. The CNF $F(X_1,X_2)$ implies $H(X_2)$ because each clause of H is implied by F since it is derived by resolution. Assume that H is not existentially implied by F. Then there is an assignment z to the variables of X_2 such that $H(z)$=1 and for any assignment y to the variables of X_1, $F(y,z)$=0. However, this means that F implies a clause K depending only on variables of X_2 such that $K(z)$=0. Since K should be in H, then $H(z)$ should be equal to 0, which leads to a contradiction.

DEFINITION 16 *Let F be a set of clauses. Denote by* ***supp(F)*** *the set of variables whose literals occur in clauses of F.*

To estimate the complexity of obtaining the function existentially implied by F in general resolution, we need the following proposition.

PROPOSITION 4 *Let F be a set of clauses that implies a clause K. Then there is a sequence of at most $3^{|supp(F)|}$ resolution steps that results in the derivation of the clause K or a clause that implies K.*

Proof. Denote by F' the formula that is obtained from F by making the assignments that set the literals of K to 0 (and removing the satisfied clauses and the literals set to 0). It is not hard to see that F' is unsatisfiable since it implies an empty clause. So there is a resolution proof $L(F')$ that results in deducing an empty clause. Then by replacing each clause of F' involved in $L(F')$ with its "parent" clause from F we get a sequence of resolutions resulting in deducing either the clause K or a clause that implies K. The number of resolvents in $L(F')$ cannot be more than $3^{|supp(F')|}$ (which is the total number of clauses of $|supp(F')|$ variables) and so it cannot be more than $3^{|supp(F)|}$.

REMARK 6 *From Propositions 3 and 4 it follows that given a CNF $F(X_1, X_2)$ one can obtain the function $H(X_2)$ existentially implied by F in no more than $3^{|supp(F)|}$ resolution steps.*

1.6 Equivalence Checking in General Resolution

In this section, we prove some results about the complexity of formulas of the class $M(p)$ (describing equivalence checking of circuits with a CS of granularity p) in general resolution. The main idea of the proof is that if S is a CS of N_1 and N_2, then their equivalence checking reduces to computing filtering and correlation functions. For each variable C of the specification S one needs to compute filtering functions $Ff(v_1(C))$,$Ff(v_2(C))$ and the correlation function $Cf(v_1(C), v_2(C))$. Here $v_1(C)$ (respectively $v_2(C)$) are coding variables of the encoding $q_1(C)$ (respectively $q_2(C)$) used when obtaining the implementation N_1 (respectively N_2).

The three main properties of these functions are that

- They can be built based only on the information about the topology of S and about "assignment" of gates of N_1 and N_2 to blocks of S.
- Filtering functions and correlation functions corresponding to primary input variables of the specification are constants.
- Filtering and correlation functions for a variable C specifying the output of a block $G(A, B)$ can be computed "locally" from filtering and correlation functions of variables A and B and CNFs specifying implementations $I_1(G)$ and $I_2(G)$. So these functions can be computed in topological order starting with inputs and proceeding to outputs.

A general scheme for the computation of filtering and correlation functions is shown in Fig. 1.4. To compute the filtering functions $Ff(v_1(C))$ and $Ff(v_2(C))$ and the correlation function $Cf(v_1(C), v_2(C))$ one needs to know filtering functions $Ff(v_1(A))$,$Ff(v_2(A))$,$Ff(v_1(B))$, $Ff(v_2(B))$ and correlation functions $Cf(v_1(A), v_2(A))$,$Cf(v_1(B), v_2(B))$.

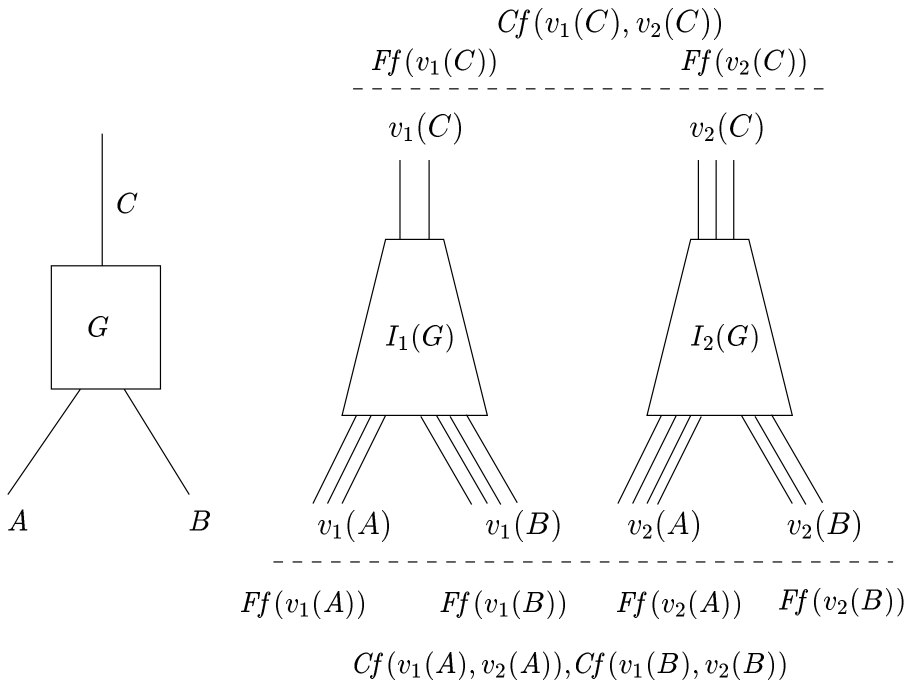


Figure 1.4. Computation of filtering and correlation functions

In this chapter, we consider computation of filtering and correlation functions (represented as CNF formulas) in the general resolution proof system. However, one can use other ways of computing these functions, for example, employing BDDs[4].

DEFINITION 17 *Let N be an implementation of a specification S. Let C be a variable of S. A function Ff is called* ***a filtering function*** *if:*

- *supp(Ff) $\subseteq$ v(C).*
- *If an assignment z to the variables of $v(C)$ is a code $q(c)$, $c \in D(C)$, then Ff(z)=1. Otherwise, Ff(z)=0.*

REMARK 7 *If C is a primary input variable of S , then Ff(v(C))$\equiv$1. Indeed, as it follows from Remark 1, any assignment to $v(C)$ is the code of a value $c \in D(C)$.*

PROPOSITION 5 *Let N be an implementation of a specification S. Let C=G(A,B) be a block of S. Let F be the CNF formula specifying N built as described in Section 1.1.3 and $F(I(G))$ be the part of F specifying the*

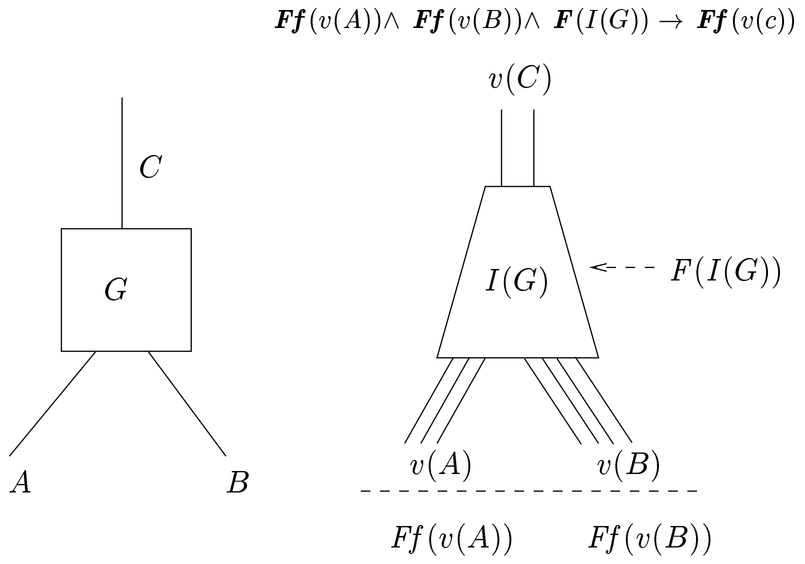


Figure 1.5. Computation of a filtering function

implementation I(G) of G in N. Then P existentially implies Ff(v(C)) where P=Ff(v(A)) $\wedge$ Ff(v(B)) $\wedge$ F(I(G)).

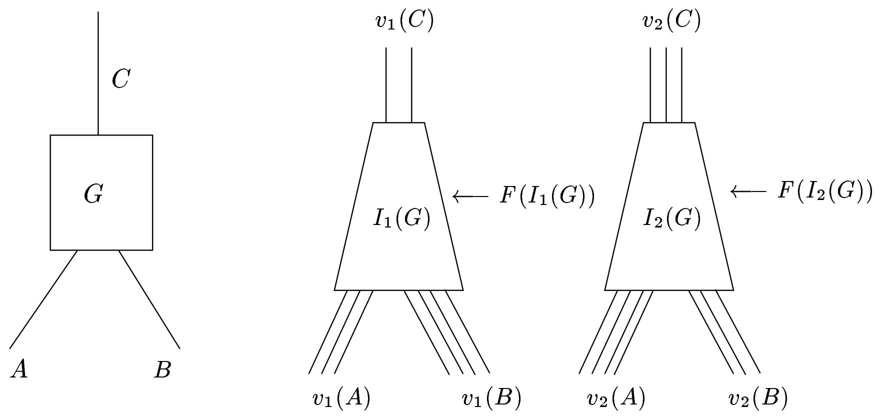
Proof. To make it easier for the reader to "visualize" the proof, we illustrate the proposition in Fig. 1.5. To prove that $P \rightarrow Ff(v(C))$ one needs to show that any assignment that sets P to 1 also sets $Ff(v(C))$ to 1. It is not hard to see that the support of all the functions of the expression $P \rightarrow Ff(v(C))$ is a subset of $supp(F(I(G)))$. Let h=(x,y,z) be an assignment that sets P to 1 where x,y,z are assignments to the variables from $v(A)$,$v(B)$ and $v(C)$ respectively. Then h has to set to 1 the functions $Ff(v(A))$, $Ff(v(B)$, $F(I(G))$. Since h sets $Ff(v(A))$ to 1, then x=$q(a)$, $a \in D(A)$. Since h sets $Ff(v(B))$ to 1, then y=$q(b)$, $b \in D(B)$. So $h = (q(a), q(b), z)$. To set to 1 $F(I(G))$, assignment z has to be equal to $q(c)$, where c=$G(a,b)$. Then h sets $Ff(v(C))$ to 1.

Assume that $Ff(v(C))$ is not existentially implied by P. Then there exists an assignment z=$q(c)$, $c \in D(C)$ such that $Ff(z)$=1 and for any assignments x and y to the variables of $v(A)$ and $v(B)$ respectively, $P(x,y,z)$=0. However, $P(q(a)$, $q(b)$, $z) = 1$ where a and b are values of A and B such that $G(a,b)$=c, which leads to a contradiction.

DEFINITION 18 *Let S be a CS of circuits N_1 and N_2 and C be a variable of S. A function Cf is called* ***a correlation function*** *for encodings q_1 and q_2 of the values of C (used when producing N_1 and N_2) if :*

- *supp(Cf) $\subseteq v_1(C) \cup v_2(C)$.*

$$\textbf{\textit{Filtering}} \wedge \textbf{\textit{Correlation}} \wedge \textbf{\textit{Implementation}} \rightarrow \textbf{\textit{Cf}}(v_1(C), v_2(C))$$



$$\textit{Filtering} = Ff(v_1(A)) \wedge Ff(v_1(B)) \wedge Ff(v_2(A)) \wedge Ff(v_2(B))$$

$$\textit{Correlation} = Cf(v_1(A), v_2(A)) \wedge Cf(v_1(B), v_2(B))$$

$$\textit{Implementation} = F(I_1(G)) \wedge F(I_2(G))$$

Figure 1.6. Computation of a correlation function

- *Cf(z_1, z_2)=1 for any assignment z_1 to v_1(C) and z_2 to v_2(C) such that z_1=q_1(c) and z_2=q_2(c) where c $\in$ D(C). Otherwise Cf(z_1, z_2)=0.*

REMARK 8 *If C is a primary input variable of S, then Cf(v_1(C),v_2(C)) $\equiv$ 1. Indeed, as it follows from Remark 1, any assignment to v_1(C) or v_2(C) is the code of a value c $\in$ D(C). Besides, from the definition of CS it follows that q_1(C)=q_2(C). Finally, from Assumption 6 it follows that $v_1(C) = v_2(C)$. So any assignment (x,y) to the variables of $v_1(C)$,$v_2(C)$ can be represented as (q_1(c),q_2(c)), c $\in$ D(C).*

PROPOSITION 6 *Let S be a CS of circuits N_1,N_2. Let $C = G(A,B)$ be a block of S. Let F be the CNF formula specifying the miter of N_1,N_2 built as described in Section 1.1.3. Let $F(I_1(G))$ and $F(I_2(G))$ be the part of F specifying the implementation $I_1(G)$ and $I_2(G)$ of G in N_1 and N_2 respectively. Then P existentially implies Cf($v_1(C), v_2(C)$) where*

- *P = Filtering $\wedge$ Correlation $\wedge$ Implementation*
- *Filtering = Ff(v_1(A)) $\wedge$ Ff(v_1(B)) $\wedge$ Ff(v_2(A)) $\wedge$ Ff(v_2(B))*
- *Correlation = Cf(v_1(A),v_2(A)) $\wedge$ Cf(v_1(B),v_2(B))*

- *Implementation* = $F(I_1(G)) \wedge F(I_2(G))$.

Proof. To make it easier for the reader to "visualize" the proof, we illustrate the proposition in Fig. 1.6. To prove that P implies $Cf(v_1(C),v_2(C))$ one needs to show that any assignment that sets P to 1 also sets $Cf(v_1(C),v_2(C))$ to 1. It is not hard to see that the support of all the functions of the expression $P \rightarrow Cf(v_1(C),v_2(C))$ is a subset of $supp(F(I_1(G)) \cup supp(F(I_2(G))$. Let h=(x_1, x_2, y_1, y_2, z_1, z_2) be an assignment that sets P to 1 where x_1, x_2, y_1, y_2, z_1, z_2 are assignments to $v_1(A)$, $v_2(A)$, $v_1(B)$, $v_2(B)$, $v_1(C)$, $v_2(C)$ respectively. Then h has to set to 1 all the functions the conjunction of which forms P. Since h has to set the function *Filtering* to 1, then x_1=$q_1(a_1)$, x_2=$q_2(a_2)$ where a_1,$a_2 \in D(A)$ and y_1=$q_1(b_1)$, y_2=$q_2(b_2)$, where b_1,$b_2 \in D(B)$. So h=($q_1(a_1)$,$q_2(a_2)$, $q_1(b_1)$,$q_2(b_2)$, z_1, z_2). Since h sets the function *Correlation* to 1, then a_1 has to be equal to a_2 and b_1 has to be equal to b_2. So h can be represented as ($q_1(a)$,$q_2(a)$, $q_1(b)$,$q_2(b)$, z_1, z_2) where $a \in D(A)$ and $b \in D(B)$. Since h sets the function *Implementation* to 1, then z_1 has to be equal to $q_1(c)$, c=$G(a,b)$ and z_2 has to be equal to $q_2(c)$. So h is equal to ($q_1(a)$,$q_2(a)$,$q_1(b)$,$q_2(b)$,$q_1(c)$,$q_2(c)$) and hence it sets the correlation function $Cf(v_1(C),v_2(C))$ to 1.

Assume that $Cf(v_1(C),v_2(C))$ is not existentially implied by P. Then there exists an assignment z_1=$q_1(c)$, z_2=$q_2(c)$ to the variables of $v_1(C)$ and $v_2(C)$ respectively such that $Cf(z_1, z_2)$=1 and for any assignment x_1, x_2, y_1, y_2 to the variables of $v_1(A)$, $v_2(A)$, $v_1(B)$, $v_2(B)$ respectively, $P(x_1, x_2, y_1, y_2, z_1, z_2)$=0. However, $P(q_1(a), q_2(a), q_1(b), q_2(b), z_1, z_2)$=1 where a, b are the values of A and B respectively for which c=$G(a,b)$. This leads to a contradiction.

PROPOSITION 7 *Let F be a formula of $M(p)$ specifying the miter of circuits N_1,N_2 obtained from a CS S of granularity p. The unsatisfiability of F can be proven by a resolution proof of no more than $d*n*3^{6p}$ resolution steps where n is the number of blocks in S and d is a constant.*

Proof. From Proposition 5 and Proposition 6 it follows that one can deduce correlation and filtering functions for all the variables of S starting with blocks of topological level 1 and proceeding in topological order. Indeed, let C=$G(A,B)$ be a block of topological level 1. Then A and B are primary input variables and the filtering and correlation functions for them are known (they are tautologies). Then $Ff(v_1(C))$ and $Ff(v_2(C))$ are existentially implied by $F(I_1(G))$ and $F(I_2(G))$ respectively. According to Proposition 5 $Ff(v_1(C))$ (respectively $Ff(v_2(C))$) can be derived by resolving clauses of $F(I_1(G))$ (respectively $F(I_2(G))$). Similarly, the correlation function $Cf(v_1(C),v_2(C))$ is existentially implied by $F(I_1(G)) \wedge F(I_2(G))$. So it can be derived from the latter by resolution. After filtering and correlation functions are computed for all the variables of level 1, the same procedure can be applied to variables of topological level 2 and so on. If S consists of n blocks, then in n steps one can deduce correlation functions for the primary output variables of S. At each step two filtering and one correlation function are computed for a variable $C{=}G(A{,}B)$ of S. The complexity of this step is no more than 3^{6p}. Indeed, the support of all functions mentioned in Proposition 5 and Proposition 6 needed for computing $Ff(v_1(C))$, $Ff(v_2(C))$ and $Cf(v_1(C){,}v_2(C))$ is a subset of $E{=}supp(F(I_1(G))) \cup supp(F(I_2(G)))$. The total number of gates in $I_1(G)$ and $I_2(G)$ is bounded by $2p$, each gate having 2 inputs and 1 output. So the total number of variables in E cannot be more than $6p$. Then according to Remark 6 in no more than 3^{6p} steps one can deduce CNFs $Ff(v_1(C))$, $Ff(v_2(C))$ and $Cf(v_1(C){,}v_2(C))$. Then the total number of resolution steps one needs to deduce correlation functions for primary output variables of S is bounded by $n{*}3^{6p}$.

Now we show that from the correlation functions for primary output variables of S one can deduce an empty clause in the number of resolution steps linear in $n * p$. Let C be a primary output variable specifying the output of a block G of N. Let $I_1(G)$ and $I_2(G)$ be the implementations of G in N_1 and N_2 respectively. Let $|D(C)| = 2^k$ (By Assumption 2 the multiplicity of C is a power of 2.) Then $length(q_1(C)){=}$ $length(q_2(C)){=}k$. (By Assumption 3 values of C are encoded by a minimal length encoding.)

Now we show that there is always a correlation function $Cf(v_1(C){,}v_2(C))$ specified by the CNF consisting of k pairs of two literal clauses specifying the equivalence of corresponding outputs of $I_1(G)$ and $I_2(G)$. Let f_1 and f_2 be two Boolean variables of $v_1(C)$ and $v_2(C)$ respectively that specify corresponding outputs of N_1 and N_2. Since S is a CS of N_1 and N_2, then $q_1(C) = q_2(C)$. So any assignment $q_1(c), q_2(c)$ to $v_1(C)$ and $v_2(C)$ that satisfies $Cf(v_1(C), v_2(C))$ also satisfies clauses $K'{=}f_1 \vee \sim f_2$ and $K''{=}{\sim}f_1 \vee f_2$. So K' and K'' are implied by $Cf(v_1(C){,}v_2(C))$ and can be deduced by the procedure described in the proof of Proposition 6. (The resolution steps one needs to deduce equivalence clauses are already counted in the expression $n * 3^{6p}$.)

Using each pair of equivalence clauses K' and K'' and the clauses specifying the gate $g{=}\text{XOR}(f_1{,}f_2)$ of the miter, one can deduce a single literal clause ${\sim}g$. This clause requires setting the output of this XOR gate to 0. Each such a clause can be deduced in the number of resolutions bounded by a constant and the total number of such clauses cannot be more than $n{*}p$. Finally, from these unit clauses and the clauses specifying the final OR gate of the miter, the empty clause can be deduced in the number of resolutions bounded by $n{*}p$. So the empty clause is deduced in no more than $n{*}3^{6p} + d'{*}n{*}p$ steps where d' is a constant. Finally, one can pick a constant d such $n{*}3^{6p} + d'{*}n{*}p \leq d{*}n{*}3^{6p}$

Remark 9 *In Proposition 7 we give a very conservative estimate of the complexity of deducing filtering and correlation functions. In practice this complexity can be much lower. In a sense, the best way to interpret the theory developed in this section is that the problem of equivalence checking of circuits N_1,N_2 with a CS S of n blocks can be partitioned into n subproblems of computing filtering and correlation functions for each variable of S.*

Remark 10 *In general, two functionally equivalent circuits N_1,N_2 may have more than one CS. In that case, when estimating the complexity of equivalence checking of N_1,N_2, it is natural to use the finest CS (see Definition 9).*

1.7 Equivalence Checking of Circuits with Unknown CS

In Section 1.1.6 we considered equivalence checking in general resolution that is a non-deterministic proof system. This means that the proof is guided by an "oracle" that points to the next pair of clauses to be resolved. Deterministic algorithms do not have the luxury of using an oracle. A natural question is whether a deterministic algorithm can benefit from the fact that the formulas from $M(p)$ have short proofs of unsatisfiability in general resolution. (In this section, we assume that one has to prove the unsatisfiability of a formula F, $F \in M(p)$ specifying equivalence checking of N_1,N_2 and no CS of N_1,N_2 is known.) A theory studying the complexity of finding proofs started only a few years ago [2, 18] and so it cannot fully answer this question yet. However, there is a good reason to believe that formulas of $M(p)$ are hard for deterministic algorithms. (Henceforth, by a deterministic algorithm we mean a resolution based deterministic SAT-algorithm.) Indeed, let us make the following two very plausible assumptions. First assumption is that there is a subclass M^* of formulas from $M(p)$ such that resolution proofs described in the proof of Proposition 7 (we will refer to them as ***specification driven proofs***) are "much shorter" than any other kind of resolution proofs. Second assumption is that finding a non-trivial CS of two Boolean circuits N_1 and N_2 is hard. If the two assumptions above are true then formulas from M^* should be hard. Indeed, specification driven resolution proofs very closely follow a CS of N_1 and N_2. So knowing a short resolution proof of the unsatisfiability of F,$F \in M^*$ one could easily recover the CS that "guided" that proof. That would mean that there is an efficient algorithm for extracting a common specification of N_1 and N_2, which contradicts our second assumption. One more argument in support of the conjecture that formulas from $M(p)$ are hard for deterministic algorithms is that formulas from $M(p)$ are hard for the best existing SAT-solvers (see Section 1.1.9).

To give the reader an idea of how big the difference between the size of non-deterministic and deterministic proofs might be, let us consider the class of formulas $M(p)$ where p is bounded by a constant. From Proposition 7 it follows that specification driven proofs consist of at most $d * n * 3^{6p}$ resolution steps that is they have linear size. On the other hand, the complexity of these formulas for a deterministic algorithm should be $Length(F)^{g(p)}$ where F is a formula of $M(p)$, $Length(F)$ is the length of F and $g(p)$ is a monotone increasing function that is linear (or close to linear) in p. One argument in favor of such complexity is that a deterministic algorithm views the whole formula F as one "block" and the complexity of specification driven proofs is exponential in the size of the maximal block. Another reason is that as it was shown in [9] one can always pick binary encodings of multi-valued variables of a CS so that every specification driven proof will have to contain "long" clauses whose length is a monotone increasing function of p. Then even formulas from a class $M(p)$ with a quite small value of p, like p=10, can be extremely hard for a deterministic algorithm. So it is quite possible that no matter how good and efficient your resolution based SAT-solver is it will not be able to solve even formulas of linear complexity!

1.8 A Procedure of Equivalence Checking for Circuits with a Known CS

In the previous section, we gave some reasons why formulas from $M(p)$ should be hard for a deterministic resolution based SAT-algorithm. Let S be a CS of Boolean circuits N_1,N_2 and p be the granularity of S. Let F be the formula of $M(p)$ specifying the equivalence checking of N_1,N_2. The good news is that if S is known then there is an efficient algorithm for proving the unsatisfiability of F. This algorithm also proceeds in topological order of variables of S computing filtering and correlation functions. The only difference with specification guided proofs of general resolution is that the "power" of the proof "oracle" is limited. Namely, in general resolution this oracle guides every resolution step of the proof (pointing to the next pair of clauses to resolve). In the deterministic algorithm described below the specification S serves as an oracle of "limited" power. Namely, this oracle helps only to identify subcircuits

$I_1(G)$ and $I_2(G)$ N_1 and N_2 that are implementations of the same block $C = G(A,B)$. Finding the correlation function $Cf(v_1(C), v_2(C))$ and filtering functions $Ff(v_1(C))$ and $Ff(v_2(C))$ is done by this algorithm without any "help".

Our procedure of equivalence checking consists of two stages:

1. For each variable C of S compute filtering functions $Ff(v_1(C))$, $Ff(v_2(C))$ and the correlation function $Cf(v_1(C),\ v_2(C))$ proceeding in topological order of variables. If C is a primary input variable, then $Ff(v_1(C))$, $Ff(v_2(C))$ and $Cf(v_1(C),\ v_2(C))$ are tautologies. Let C=$G(A,B)$. Then $Ff(v_1(C))$ is built by computing the function existentially implied (see Definition 15) by $Ff(v_1(A)) \vee Ff(v_1(B)) \vee F(I_1(G))$. ($F(I_1(G))$ is a subset of F specifying the implementation of G in N_1. The function $Ff(v_2(C)$ is built similarly to $Ff(v_1(C))$.) The function $Cf(v_1(C)$,$v_2(C))$ is built by computing the function existentially implied by $Ff(v_1(A)) \vee Ff(v_1(B)) \vee Ff(v_2(A)) \vee Ff(v_2(B)) \vee Cf(v_1(A),\ v_2(A)) \vee Cf(v_1(B),\ v_2(B)) \vee F(I_1(G)) \vee F(I_2(G))$.

2. Once correlation functions are computed for all primary output variables of S, finish the proof of unsatisfiability of F by invoking a SAT-solver like [8],[16]. (This SAT-solver is applied to the CNF consisting of the clauses describing the correlation functions for the primary output variables of S, the clauses specifying the gates XORing primary outputs of N_1 and N_2 and the final OR gate of the miter.)

The complexity of this procedure is about the same as in general resolution which is equal to $d * n * 3^{6p}$ where d is a constant and n is the number of blocks. The only difference is that in general resolution no resolvent is generated twice while the procedure above may generate identical clauses when computing correlation or filtering functions. So it will have to take care of removing duplicate clauses.

The described procedure is flexible with respect to the method of computing existentially implied functions. Below we describe a few options. Let F be a CNF and $supp(F) = X_1 \cup X_2$. Suppose one needs to compute a CNF $H(X_2)$ that is existentially implied by F. If the value of $|X_2|$ is small, one can compute $H(X_2)$ by running 2^k SAT-checks where k=$|X_2|$. For every assignment z to the variables of X_2 one needs to check if there is an assignment y to the variables of X_1 such that (y,z) satisfies F. If such an assignment exists then the next assignment is checked. Otherwise, a clause consisting of literals of variables from X_2 that is falsified by the assignment z is added to the clauses of $H(X_2)$.

If the size of X_2 is large, one can compute filtering and correlation functions by existential quantification of the variables of X_1. In terms of SAT, existential quantification of a CNF F in a variable w of X_1 means adding to F all the resolvents that can be produced by resolving clauses of F in w. Of course, existential quantification in all the variables of X_1 is very expensive in SAT and so it works only for blocks of a small size. However, less expensive methods for computing $H(X_2)$ in terms of SAT can be and should be developed.

1.9 Experimental Results

The objective of experiments was to show that equivalence checking of circuits with a fine CS S is easy if S is known and is hard otherwise. To produce circuits having a fine CS we used the following procedure. To get multi-valued specifications with realistic topologies we "borrowed" them from MCNC-91 benchmark circuits as follows. First, all the benchmarks were technology mapped using SIS [20] consisting only of two-input AND gates. Then from each obtained circuit N a multi-valued specification S was produced by replacing each two-input binary gate with a two-input single output block of four-valued variables. (In other words, S changes the functionality of N while preserving its topology.) Then from S two functionally equivalent Boolean circuits N_1, N_2 implementing S were produced using two different sets of two-bit encodings of four-valued values. The encodings were picked in such a way that the two different implementations of the same four-valued block in N_1 and N_2 had no functionally equivalent outputs. This way we guaranteed that internal functionally equivalent points in N_1 and N_2 may occur only by accident.

Note that after encoding, the number of inputs and outputs in N_1 and N_2 is twice the number of inputs and outputs in the original Boolean circuit N. For instance, the two circuits produced from C6288 used as a "specification" have the topology of a 16-bit multiplier and the number of inputs and outputs of a 32-bit multiplier.

In experiments we used the best tools that were available to us. Namely, we used the SAT-solver BerkMin downloaded from [1], the program Nanotrav built on top of the Colorado University Decision Diagram (CUDD) package [6] and a SAT-based equivalence checker CSAT [14] (courtesy of Prof. Li of UCSB). We also tried the SAT-solver Zchaff [16], but BerkMin was up to three orders of magnitude faster on our formulas. In the experiments we used the special mode of BerkMin designed for equivalence checking that is described at [1]. BerkMin was run on the formula specifying the miter M of N_1 and N_2 as described in Section 1.1.3. Nanotrav was used to build a BDD for the miter M and CSAT checked the satisfiability of the miter's output. We first ran the three tools on "regular" MCNC benchmarks to verify optimized versus non-optimized circuits. (We do not report these results). The tools showed quite decent performance. For example, BerkMin was able to quickly verify all the instances including the multiplier C6288. The same kind of performance was shown by CSAT. Nanotrav was able to build BDDs for all the miters except C6288 very quickly (in a few seconds). In all the experiments we ran Nanotrav using settings suggested by Fabio Somenzi (private communication). In particular, the variable sifting option was on. In Table 1.1.9 we give runtimes of the three programs shown in our experiments. All the programs were run on a SUNW Ultra-80 system with clock frequency 450MHz. In all the experiments the time limit was set to 60,000 sec. (16.6 hours). The results of the best out of the three programs is shown in black. In the last column we report run times of a trivial CS driven procedure. This procedure computes filtering and correlation function of blocks in terms of SAT by existentially quantifying variables (as it was described in Section 1.1.8) and eventually deduces an empty clause.

It is not hard to see that run times of the CS driven procedure are linear in the size of circuits to be checked for equivalence. This is due to the fact that the size of specification blocks is fixed (and very small). On the other hand, the instances we generated turned out to be hard for the three chosen tools. Even if one compares the best run times with run times of the CS driven procedure, it is not hard to see that the former quickly increased as the size of the instances grew.

It is unlikely that an industrial strength equivalence checker would do much better on the circuits we generated because they have no functionally equivalent points. Besides, one can always produce much harder equivalence checking problems by using ***even a slightly*** more coarse specification (Recall that in the experiments we used a very fine CS S consisting of four-valued blocks. That is the circuits produced from S were "almost" identical.) As we mentioned in the introduction, the problem of finding a short proof of equivalence of N_1,N_2 if a CS is not known, comes down to recovering this CS from the description of N_1,N_2 which is computationally very hard (if not infeasible).

1.10 Conclusions

In the first part of this chapter, we introduced a class $M(p)$ of CNF formulas specifying equivalence checking of Boolean circuits with a common specification (CS). We showed that formulas of $M(p)$ are "easy" for general resolution and gave reasons why those formulas should be hard for a deterministic algorithm that does not know a CS of the circuits to be checked for equivalence. We also gave some experimental evidence that formulas from $M(p)$ are hard for existing SAT-solvers. Besides, we formulated an efficient SAT-algorithm for equivalence checking of cir-

Table 1.1. Equivalence checking of circuits with a fine CS

Name of "specification"	Number of variables	Number of clauses	CSAT (sec.)	Nanotrav (BDDs) (sec.)	BerkMin (sec.)	CS driven (sec.)
C880	1,612	9,373	162.8	60,000	**3.7**	1.1
ttt2	2,770	17,337	281.0	**1.0**	11.7	1.3
x4	4,166	24,733	284.3	**4.7**	17.3	1.8
i9	4,954	29,861	75.3	**1.5**	32.7	2.1
term1	3,504	22,229	1,604.6	40.9	**35.9**	1.6
c7552	11,282	69,529	282.0	60,000	**52.8**	3.6
c3540	5,248	33,199	34,905.8	60,000	**64.1**	2.3
rot	5,980	35,229	163.6	19,315.6	**72.2**	2.1
9symml	960	6,105	31.07	**1.9**	113.2	0.5
frg2	10,316	62,943	13,610.4	**22.6**	131.4	2.9
frg1	3,230	20,575	**265.8**	60,000	330.3	1.7
i10	12,998	77,941	60,000	60,000	**445.0**	4.8
des	28,902	179,895	12,520.3	**9.7**	451.7	12.1
dalu	9,426	59,991	17,496.9	60,000	**518.6**	3.1
x1	8,760	55,571	13,580.3	13,009.6	**950.2**	2.8
alu4	4,736	30,465	8,020.4	**135.1**	992.6	2.0
i8	14,524	91,139	60,000	**98.0**	1,051.5	5.1
c6288	9,540	61,421	60,000	60,000	**1,955.1**	5.2
k2	11,680	74,581	60,000	59,392.9	**5,121.5**	4.3
too_large	58,054	376,801	60,000	60,000	60,000	15.2
t481	19,042	123,547	60,000	60,000	60,000	6.3

cuits with a known CS. The results of the first part of this chapter lead to the following two conclusions.

- A resolution based SAT-solver (most probably) cannot be scalable even on "easy" and practical formulas unless some extra information about the structure of short proofs is provided. (In case of equivalence checking this extra information is provided by a CS.)

- The SAT-solvers of the future should be very "intelligent" that is very receptive to structural properties of the formula to be tested for satisfiability.

2. Stable Sets of Points

2.1 Introduction

In the first part of this chapter, we showed that it is extremely important for a SAT-solver to be "receptive" to structural properties of CNF formulas. However, the existing algorithms are not very good at taking into account such properties. One of the reasons is that currently there is no "natural" way of traversing the search space. For example, in the DPLL procedure [7] which is the basis of almost all algorithms used in practice the search is organized as a binary tree. In reality, the search tree is used only to impose a linear order on the points of the Boolean space to avoid visiting the same point twice. However, this order may be in conflict with "natural" relationships between points of the Boolean space that are imposed by the CNF formula to be checked for satisfiability (for example, if this formula has some symmetries).

In the second part, we introduce the notion of a stable set of points (SSP) [11]. We believe that SSPs can serve as a basis for constructing algorithms that traverse the search space in a "natural" way. This may lead to creating SAT-solvers that are much more "intelligent" and efficient than the existing state-of-the-art SAT-solvers. We show that a CNF formula F is unsatisfiable if and only if there is a set of points of the Boolean space that is stable with respect to F. If F is satisfiable then any subset of points of the Boolean space is unstable, and an assignment satisfying F will be found in the process of constructing an SSP. We describe a simple algorithm for constructing an SSP. Interestingly, this algorithm is, in a sense, an extension of Papadimitriou's algorithm [17] (or a similar algorithm that is used in the well-known program called Walksat [19]).

A very important fact is that, generally speaking, a set of points that is stable with respect to a CNF formula F depends only on the clauses (i.e. disjunctions of literals) F consists of. So the process of constructing an SSP can be viewed as a "natural" way of traversing the search space when checking F for satisfiability. In particular, if F has symmetries, they can be easily taken into account when constructing an SSP. To illustrate this point, we consider the class of CNF formulas that are symmetric with respect to a group of permutations. We show that in this case for proving the unsatisfiability of a CNF formula it is sufficient to construct a set of points that is stable modulo symmetry.

If, for a class of formulas, SSPs are exponentially large, computing a monolithic SSP point-by-point is too time and memory consuming. We experimentally show that this is the case for hard random CNFs formulas. One of the possible solutions to this problem is to exclude some directions (i.e. variables) from consideration when computing an SSP. Such a set of points is stable only with respect to "movements" in the allowed directions. By excluding directions one can always get an SSP of small size. We sketch a procedure of satisfiability testing in which computing a monolithic SSP is replaced with constructing a sequence of small SSPs with excluded directions.

The second part of this chapter is structured as follows. In Section 1.2.2 we introduce the notion of an SSP. Section 1.2.3 relates an SSP with a set of points "reachable" from a point. A simple algorithm for building an SSP point-by-point is described in Section 1.2.4. We also show experimentally in Section 1.2.4 that even small CNF formulas may have large sets of SSPs and so computing SSPs point-by-point is in general infeasible. In Sections 1.2.5, 1.2.6 we discuss two possible ways of using SSPs. In Section 1.2.5 we show that to prove a symmetric CNF formula to be unsatisfiable it is sufficient to build an SSP modulo symmetries of that formula. Such an SSP can be sometimes efficiently built even point-by-point. Section 1.2.6 shows that the computation of a monolithic SSP can be replaced with the construction of so called SSPs with excluded directions whose size is easy to control. Finally, some conclusions are made in Section 1.2.7.

2.2 Stable Set of Points

In this section, we introduce the notion of an SSP. Let F be a CNF formula of n variables $x_1,\ldots,x_n$. Denote by B the set $\{0,1\}$ of values taken by a Boolean variable. Denote by B^n the set of points of the Boolean space specified by variables x_1,...,x_n. ***A point*** of B^n is an assignment of values to all n variables.

Definition 19 *Let p be a point of the Boolean space falsifying a clause C. The* ***1-neighborhood of the point*** *p with respect to the clause C (written* ***Nbhd(p, C)****) is the set of points that are at Hamming distance 1 from p and that satisfy C.*

Remark 11 *It is not hard to see that the number of points in Nbhd(p, C) is equal to that of literals in C.*

Example 4 *Let* $C = x_1 \vee \overline{x_3} \vee x_6$ *be a clause specified in the Boolean space of 6 variables* $x_1,\ldots,x_6$. *Let* $p = (x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0, x_5 = 1, x_6 = 0)$ *be a point falsifying C. Then Nbhd(p, C) consists of the following three points:* $p_1 = (\boldsymbol{x_1{=}1}, x_2 = 1, x_3 = 1, x_4 = 0, x_5 = 1, x_6 = 0)$, $p_2 = (x_1 = 0, x_2 = 1, \boldsymbol{x_3{=}0}, x_4 = 0, x_5 = 1, x_6 = 0)$, $p_3 = (x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0, x_5 = 1, \boldsymbol{x_6{=}1})$.

Points p_1,p_2,p_3 are obtained from p by flipping the value of variables x_1,x_3,x_6 respectively i.e. the variables whose literals are in C.

Denote by $Z(F)$ the set of points at which F takes value 0. If F is unsatisfiable, $Z(F) = B^n$.

DEFINITION 20 *Let F be a CNF formula and P be a subset of $Z(F)$. Mapping g of P to F is called a* ***transport function*** *if, for any $p \in P$, the clause $g(p) \in F$ is falsified by p. In other words, a transport function g:$P \rightarrow F$ is meant to assign each point $p \in P$ a clause that is falsified by p.*

REMARK 12 *We call mapping $P \rightarrow F$ a transport function because, as it is shown in Section 1.2.3, such a mapping allows one to introduce some kind of "movement" of points in the Boolean space.*

DEFINITION 21 *Let P be a nonempty subset of $Z(F)$, F be a CNF formula, and g: $P \rightarrow F$ be a transport function. The set P is called* ***stable*** *with respect to F and g if $\forall p \in P$, $Nbhd(p,g(p)) \subseteq P$. As it was mentioned before, "stable set of points" abbreviates to* ***SSP****.*

REMARK 13 *Henceforth, if we say that a set of points P is stable with respect to a CNF formula F without mentioning a transport function, we mean that there is a function g:$P \rightarrow F$ such that P is stable with respect to F and g.*

EXAMPLE 5 *Consider an unsatisfiable CNF formula F consisting of the following 7 clauses: $C_1 = x_1 \vee x_2$, $C_2 = \overline{x_2} \vee x_3$, $C_3 = \overline{x_3} \vee x_4$, $C_4 = \overline{x_4} \vee x_1$, $C_5 = \overline{x_1} \vee x_5$, $C_6 = \overline{x_5} \vee x_6$, $C_7 = \overline{x_6} \vee \overline{x_1}$. Clauses of F are composed of literals of 6 variables: x_1,...,x_6. The following 14 points form an SSP P: p_1=000000, p_2=010000, p_3=011000, p_4=011100, p_5=111100, p_6=111110, p_7=111111, p_8=011111, p_9=011011, p_{10}=010011, p_{11}=000011, p_{12}=100011, p_{13}=100010, p_{14}=100000. (Values of variables are specified in the order variables are numbered. For example, p_4 consists of assignments x_1=0, x_2=1, x_3=1, x_4=1, x_5=0, x_6=0.) The set P is stable with respect to the transport function g specified as: $g(p_1) = C_1$, $g(p_2) = C_2$, $g(p_3) = C_3$, $g(p_4) = C_4$, $g(p_5) = C_5$, $g(p_6) = C_6$, $g(p_7) = C_7$, $g(p_8) = C_4$, $g(p_9) = C_3$, $g(p_{10}) = C_2$, $g(p_{11}) = C_1$, $g(p_{12}) = C_7$, $g(p_{13}) = C_6$, $g(p_{14}) = C_5$.*

The set P and the transport function g are given in Fig. 1.7. Next to each point p_i, the clause C_k=$g(p_i)$ is shown. Besides, for each point p_i the two points comprising $Nbhd(p_i,g(p_i))$ are indicated by arrows.

$$\begin{tikzcd}
C_1 = x_1 \vee x_2,\ p_1{=}000000 \arrow[r, leftrightarrow] \arrow[d, leftrightarrow] & p_{14}{=}100000,\ C_5 = \sim x_1 \vee x_5 \arrow[d, leftrightarrow] \\
C_2 = \sim x_2 \vee x_3,\ p_2{=}010000 \arrow[d, leftrightarrow] & p_{13}{=}100010,\ C_6 = \sim x_5 \vee x_6 \arrow[d, leftrightarrow] \\
C_3 = \sim x_3 \vee x_4,\ p_3{=}011000 \arrow[d, leftrightarrow] & p_{12}{=}100011,\ C_7 = \sim x_6 \vee \sim x_1 \arrow[d, leftrightarrow] \\
C_4 = \sim x_4 \vee x_1,\ p_4{=}011100 \arrow[d, leftrightarrow] & p_{11} = 000011,\ C_1 = x_1 \vee x_2 \arrow[d, leftrightarrow] \\
C_5 = \sim x_1 \wedge x_5,\ p_5{=}111100 \arrow[d, leftrightarrow] & p_{10}{=}010011,\ C_2 = \sim x_2 \vee x_3 \arrow[d, leftrightarrow] \\
C_6 = \sim x_5 \wedge x_6,\ p_6{=}111110 \arrow[d, leftrightarrow] & p_9{=}011011,\ C_3 = \sim x_3 \vee x_4 \arrow[d, leftrightarrow] \\
C_7 = \sim x_6 \vee \sim x_1,\ p_7{=}111111 \arrow[r, leftrightarrow] & p_8{=}011111,\ C_4 = \sim x_4 \vee x_1
\end{tikzcd}$$

Figure 1.7. Illustration to Example 5

It is not hard to see that g indeed is a transport function i.e. for any point p_i of P it is true that $C(p_i){=}0$ where $C = g(p_i)$. Besides, for every point p_i of P, the condition $Nbhd(p, g(p)) \subseteq P$ of Definition 5 holds. Consider, for example, point $p_{10}{=}010011$. The value of $g(p_{10})$ is C_2, $C_2 = \overline{x_2} \vee x_3$ and $Nbhd(p_{10}, C_2) = \{p_{11} = 000011, p_9 = 011011\}$, the latter being a subset of P.

PROPOSITION 8 *If there is a set of points that is stable with respect to a CNF formula F, then F is unsatisfiable.*

Proof Assume the contrary. Let P be a set of points that is stable with respect to F and a transport function g, and p^* be a satisfying assignment i.e. $F(p^*) = 1$. It is not hard to see that $p^* \notin P$ because each point $p \in P$ is assigned a clause $C = g(p)$ such that $C(p){=}0$ and so $F(p){=}0$. Let p be a point of P that is the closest to p^* in Hamming distance. Denote by C the clause that is assigned to p by the transport function g i.e. $C = g(p)$. Denote by Y the set of variables values of which are different in p and p^*.

Let us show that C can not have literals of variables of Y. Assume the contrary, i.e. that C contains a literal of $x \in Y$. Then, since P is stable with respect to F and g, it has to contain the point p' which is obtained from p by flipping the value of x. But then $p' \in P$ is closer to p^* than p. So we have a contradiction. Since $C(p)$=0 and C does not contain literals of variables whose values are different in p and p^* we have to conclude that $C(p^*) = 0$. This means that p^* is not a satisfying assignment and so we have a contradiction.

PROPOSITION 9 *Let F be an unsatisfiable CNF formula of n variables. Then set $Z(F)$ is stable with respect to F and any transport function $Z(F) \rightarrow F$.*

Proof Since F is unsatisfiable, then $Z(F) = B^n$. For each point $p \in B^n$, condition $Nbhd(p, g(p)) \subseteq B^n$ holds.

REMARK 14 *From propositions 8 and 9 it follows that a CNF F is unsatisfiable if and only if there is a set of points stable with respect to F.*

2.3 SSP as a reachable set of points

In this section, we introduce the notion of reachability that will be used in Section 1.2.4 to formulate an algorithm for constructing an SSP. Our main objective here is to show that the set of points reachable from a point of the Boolean space is an SSP unless this set contains a satisfying assignment.

DEFINITION 22 *Let F be a CNF formula and g: $Z(F) \rightarrow F$ be a transport function. A sequence of k points $p_1, \ldots, p_k$, $k \geq 2$ is called a **path** from p_1 to p_k in a set P with a transport function g if points $p_1,\ldots,p_{k-1}$ are in P and $p_i \in Nbhd(p_{i-1},g(p_{i-1}))$, $2 \leq i \leq k$. (Note that the last point of the path, i.e. p_k, does not have to be in P.) We will assume that no point appears twice (or more) in a path.*

EXAMPLE 6 *Consider the CNF formula and transport function of Example 5. Let P be the set of points specified in Example 5. The sequence of points p_1,p_{14},p_{13},p_{12} forms a path from p_1 to p_{12}. Indeed, it is not hard to check that $Nbhd(p_1, g(p_1)) = \{p_2, p_{14}\}$, $Nbhd(p_{14}, g(p_{14})) = \{p_{13}, p_1\}$, $Nbhd(p_{13}, g(p_{13})) = \{p_{14}, p_{12}\}$, $Nbhd(p_{12}, g(p_{12})) = \{p_{13}, p_{11}\}$. So each point p' of the path (except the starting point i.e. p_1) is contained in the set $Nbhd(p'', g(p''))$ where p'' is the preceding point.*

DEFINITION 23 *Let F be a CNF formula. A point p'' is called **reachable** from a point p' by means of a transport function $g : Z(F) \rightarrow F$ if there is a path from p' to p'' with the transport function g. Denote by $Reachable(p, g)$ the set consisting of a point p and all the points that are reachable from p by means of the transport function g.*

PROPOSITION 10 *Let F be a satisfiable CNF formula, p be a point of $Z(F)$, and s be a satisfying assignment (i.e. $s \notin Z(F)$) that is the closest to p in Hamming distance. Let g:$Z(F) \rightarrow F$ be a transport function. Then in $Z(F)$ there is a path from p to s with the transport function g i.e. the satisfying assignment s is reachable from p.*

Proof Denote by Y the set of variables whose values are different in p and s. Since $F(p)$=0, then $p \in Z(F)$ and the function g assigns a clause C to p where $C(p)$=0. All literals of C are set to 0 by p. On the other hand, since s is a satisfying assignment, then at least one literal of C is set to 1 by s. Then C contains a literal of a variable y from Y. Denote by p' the point obtained from p by flipping the value of y in p. The point p' is reachable from p by means of the transport function g. If $|Y| = 1$, then p' is the satisfying assignment s. If $|Y| > 1$, then p' cannot be a satisfying assignment since, by our assumption, the satisfying assignment s is the closest to p. Then after applying the same reasoning to the point p', we conclude that the clause assigned to p' by g must contain a literal of a variable y' from $Y \setminus \{y\}$. Flipping the value of y' in p' we produce a point p'' that is either the satisfying assignment s or is at distance $|Y| - 2$ from s. Going on in this manner we reach the satisfying assignment s in $|Y|$ steps.

PROPOSITION 11 *Let P be a set of points that is stable with respect to a CNF formula F and a transport function $g : P \rightarrow F$. Then $\forall p \in P$, $Reachable(p, g) \subseteq P$.*

Proof Assume the contrary, i.e. that there is a point $p^* \in Reachable(p, g)$ that is not in P. Let H be a path from p to p^*. Denote by p'' the first point in the sequence of points specified by H that is not in P. (Points are numbered from p to p^*). Denote by p' the point preceding p'' in H. The point p' is in P and the latter is stable with respect to F and g. So $Nbhd(p', g(p')) \subseteq P$. The point p'' is in $Nbhd(p', g(p'))$ and so it has to be in P. We have a contradiction.

PROPOSITION 12 *Let F be a CNF formula, $g : Z(F) \rightarrow F$ be a transport function, and p be a point from $Z(F)$. If $P = Reachable(p, g)$ does not contain a satisfying assignment for F, then P is stable with respect to F and g, and so F is unsatisfiable.*

Proof Assume the contrary i.e. that P is not stable. Then there exists a point p' of $Reachable(p,g)$ (and so reachable from p) such that a point p'' of $Nbhd(p',g(p'))$ is not in $Reachable(p,g)$. Since p'' is reachable from p' it is also reachable from p. We have a contradiction.

REMARK 15 *From Proposition 12 it follows that a CNF F is satisfiable if and only if, given a point $p \in Z(F)$ and a transport function $g : Z(F) \rightarrow F$, the set Reachable(p, g) contains a satisfying assignment.*

In [11] properties of SSPs are discussed in more detail.

2.4 Testing Satisfiability of CNF Formulas by SSP Construction

In this section, we describe a simple algorithm for constructing an SSP that is based on Proposition 12. Let F be a CNF formula to be checked for satisfiability. The idea is to pick a point p of the Boolean space and construct the set *Reachable*(p, g). Since no transport function $g : Z(F) \rightarrow F$ is known beforehand, it is built on the fly. In the description of the algorithm given below, the set *Reachable*(p, g) is broken down into two parts: *Boundary* and *Body. Boundary* consists of those points of the current set *Reachable*(p, g) whose 1-neighborhood has not been explored yet. At each step of the algorithm a point p' of *Boundary* is extracted and a clause C falsified by p' is assigned as the value of $g(p')$. Then the set *Nbhd*(p', C) is generated and its points (minus those that are already in *Body* $\cup$ *Boundary*) are added to *Boundary*. This goes on until a stable set is constructed (F is unsatisfiable) or a satisfying assignment is found (F is satisfiable).

1 Generate a starting point p. *Boundary* $= \{p\}$. *Body*$=\emptyset$, $g = \emptyset$.

2 If *Boundary* is empty, then *Body* is an SSP and F is unsatisfiable. The algorithm terminates.

3 Pick a point $p' \in$ *Boundary*. *Boundary*=*Boundary* $\setminus \{p'\}$.

4 Find a set M of clauses that are falsified by point p'. If $M = \emptyset$, then the CNF formula F is satisfiable and p' is a satisfying assignment. The algorithm terminates.

5 Pick a clause C from M. Take C as the value of $g(p')$. Generate *Nbhd*(p', C). *Boundary* $=$ *Boundary* $\cup$ (*Nbhd*$(p', C) \setminus$*Body*). *Body* $=$ *Body* $\cup \{p'\}$.

6 Go to step 2.

Interestingly, the algorithm described above can be viewed as an extension of Papadimitriou's algorithm [17] (or a similar algorithm used in the program Walksat [19]) to the case of unsatisfiable CNF formulas. Papadimitriou's algorithm (and Walksat) can be applied only to satisfiable CNF formulas since it does not store visited points of the Boolean space. An interesting fact is that the number of points that one has to explore to prove the unsatisfiability of a CNF formula can be very small. For instance, in example 5, an SSP of a CNF formula of 6 variables consists only of 14 points while the Boolean space of 6 variables consists of 64 points. It can be shown that for a subclass of the class of 2-CNF formulas (a clause of a 2-CNF formula contains at most 2 literals) the size of minimum SSPs grows linearly in the number of variables of the formula.

A natural question to ask is: "What is the size of SSPs for "hard" CNF formulas?". One example of such formulas are random CNFs for which general resolution was proven to have exponential complexity [5]. Table 1.2 gives the results of computing SSPs for CNF formulas from the "hard" domain (the number of clauses is 4.25 times the number of variables [15]). For computing SSPs we used the algorithm described above enhanced by the following heuristic. When picking a clause to be assigned to the current point p' of *Boundary* (Step 5), we give preference to the clause C (falsified by p') for which the maximum number of points of $\mathit{Nbhd}(p', C)$ are already in *Body* or *Boundary.* In other words, when choosing the clause C to be assigned to p', we try to minimize the number of new points we have to add to *Boundary.*

We generated 10 random CNFs of each size (number of variables). The starting point was chosen randomly. Table 1.2 gives the average values of the SSP size and the share (percent) of the Boolean space taken by an SSP. It is not hard to see that the SSP size grows very quickly. So even for very small formulas it is very large. An interesting fact though is that the share of the Boolean space taken by the SSP constructed by the described algorithm steadily decreases as the number of variables grows.

The poor performance of the proposed algorithm on random CNF formulas suggests that computing a "monolithic" SSP point-by-point is too time and memory consuming. There are at least three ways of solving this problem. First way concerns computing SSPs for symmetric CNF formulas. In Section 1.2.5 we show that to prove that a symmetric CNF formula is unsatisfiable it suffices to build a set of points that is stable modulo symmetry. Such a set of points can be very small. Another way of dealing with the exponential blow-up of SSPs is described in Section 1.2.6. The idea is to exclude some directions (i.e. variables) from consideration when computing an SSP. This way the size of an SSP can be drastically reduced. By constructing an SSP with excluded directions one obtains a new implicate of the formula. By adding this implicate to the formula we make it "simpler" (in terms of the size of its SSPs). By computing SSPs with excluded directions and adding the

Table 1.2. SSPs of "hard" random CNF formulas

number of variables	`SSP size`	#SSP/#All_Space (%)
10	430	41.97
11	827	40.39
12	1,491	36.41
13	2,714	33.13
14	4,931	30.10
15	8,639	26.36
16	16,200	24.72
17	30,381	23.18
18	56,836	21.68
19	103,428	19.73
20	195,220	18.62
21	392,510	18.72
22	736,329	17.55
23	1,370,890	16.34

corresponding implicates we replace the computation of a monolithic SSP with the construction of a sequence of small size SSPs. A third (and probably most promising) way of making SSP computation more efficient is to build SSP in big "chunks" clustering "similar" points. We do not study this idea here leaving it for future research.

2.5 Testing Satisfiability of Symmetric CNF Formulas by SSP Construction

In this section, we introduce the notion of a set of points that is stable modulo symmetry. This notion allows one to modify the algorithm of SSP construction given in Section 1.2.4 to take into account a formula's symmetry. The modification itself is described at the end of the section. We consider only the case of permutations. However, a similar approach can be applied to a more general class of symmetries e.g. to the case when a CNF formula is symmetric under permutations combined with the negation of some variables.

Definition 24 *Let $X = \{x_1, \dots, x_n\}$ be a set of Boolean variables. A* ***permutation*** *π defined on set X is a bijective mapping of X onto itself.*

Let $F = \{C_1, \dots, C_k\}$ be a CNF formula. Let $p = (x_1, \dots, x_n)$ be a point of B^n. Denote by $\pi(p)$ the point $(\pi(x_1), \dots, \pi(x_n))$. Denote by $\pi(C_i)$ the clause that is obtained from $C_i \in F$ by replacing variables $x_1, \dots, x_n$ with variables $\pi(x_1), \dots, \pi(x_n)$ respectively. Denote by $\pi(F)$

the CNF formula obtained from F by replacing each clause C_i with $\pi(C_i)$.

DEFINITION 25 *A CNF formula F is called* ***symmetric*** *with respect to permutation π if the CNF formula $\pi(F)$ consists of the same clauses as F. In other words, F is symmetric with respect to π if each clause $\pi(C_i)$ of $\pi(F)$ is identical to a clause $C_k \in F$.*

PROPOSITION 13 *Let p be a point of B^n and C be a clause falsified by p i.e. $C(p)=0$. Let π be a permutation of variables $\{x_1,\ldots,x_n\}$ and $C' = \pi(C)$ and $p' = \pi(p)$. Then $C'(p') = 0$.*

Proof Let $\delta(x_i)$ be the literal of a variable x_i that is present in C. This literal is set to 0 by the value of x_i in p. The variable x_i is mapped to $\pi(x_i)$ in the clause C' and the point p'. Then the value of $\pi(x_i)$ in the point p' is the same as that of x_i in p. So the value of literal $\delta(\pi(x_i))$ in the point p' is the same as the value of $\delta(x_i)$ in p i.e. 0. Hence, the clause C' is falsified by p'.

REMARK 16 *From Proposition 13 it follows that if F is symmetric with respect to a permutation π then $F(p) = F(\pi(p))$. In other words, F takes the same value at points p and $\pi(p)$.*

The set of the permutations, with respect to which a CNF formula is symmetric, forms a group. Henceforth, we will denote this group by G. The fact that a permutation π is an element of G will be denoted by $\pi \in G$. Denote by 1 the identity element of G.

DEFINITION 26 *Let B^n be the Boolean space specified by variables $X=\{x_1,\ldots,x_n\}$ and G be a group of permutations specified on X. Denote by* ***symm(p,p′,G)*** *the following binary relation between points of B^n. A pair of points (p,p') is in $symm(p,p',G)$ if and only if there is $\pi \in G$ such that $p' = \pi(p)$.*

DEFINITION 27 *Points p and p' are called* ***symmetric*** *if they are in the same equivalence class of symm(p,p',G).*

DEFINITION 28 *Let F be a CNF formula that is symmetric with respect to a group of permutations G and P be a subset of $Z(F)$. The set P is called* ***stable modulo symmetry*** *with respect to F and a transport function g: $P \rightarrow F$ if for each $p \in P$, every point $p' \in Nbhd(p,g(p))$ is either in P or there is a point p'' of P that is symmetric to p'.*

PROPOSITION 14 *Let B^n be the Boolean space specified by variables $X = \{x_1,\ldots,x_n\}$. Let p be a point of B^n, C be a clause falsified by*

p, and a point $q \in Nbhd(p,C)$ be obtained from p by flipping the value of a variable x_i. Let π be a permutation of variables from X, p' be equal to $\pi(p)$, C' be equal to $\pi(C)$, and $q' \in Nbhd(p',C')$ be obtained from p' by flipping the value of variable $\pi(x_i)$. Then $q' = \pi(q)$. In other words, for each point q of $Nbhd(p,C)$ there is a point q' of $Nbhd(p',C')$ that is symmetric to q.

Proof The value of a variable x_k, $k \neq i$ in q is the same as in p. Besides, the value of the variable $\pi(x_k)$ in q' is the same as in p' (q' is obtained from p' by changing the value of the variable $\pi(x_i)$ and since $k \neq i$ then $\pi(x_k) \neq \pi(x_i)$). Since $p' = \pi(p)$, then the value of x_k in q is the same as the value of variable $\pi(x_k)$ in q'. On the other hand, the value of variable x_i in q is obtained by negation of the value of x_i in p. The value of the variable $\pi(x_i)$ in q' is obtained by the negation of the value of $\pi(x_i)$ in p'. Hence the values of the variable x_i in q and the variable $\pi(x_i)$ in q' are the same. So $q' = \pi(q)$.

PROPOSITION 15 *Let F be a CNF formula, P be a subset of $Z(F)$, and $g : P \rightarrow F$ be a transport function. If P is stable modulo symmetry with respect to F and g, then the CNF formula F is unsatisfiable.*

Proof Denote by $K(p)$ the set of all points that are symmetric to the point p i.e. that are in the same equivalence class of the relation *symm* as p. Denote by $K(P)$ the union of the sets $K(p)$, $p \in P$. Extend the domain of transport function g from P to $K(P)$ in the following way. Suppose p' is a point that is in $K(P)$ but not in P. Then there is a point $p \in P$ that is symmetric to p' and so $p' = \pi(p)$, $\pi \in G$. We assign $C' = \pi(C)$, $C = g(p)$ as the value of g at p'. If there is more than one point of P that is symmetric to p', we pick any of them.

Now we show that $K(P)$ is stable with respect to F and g: $K(P) \rightarrow F$. Let p' be a point of $K(P)$. Then there is a point p of P that is symmetric to p' and so $p' = \pi(p)$. Then from Proposition 14 it follows that for any point q of $Nbhd(p, g(p))$ there is a point $q' \in Nbhd(p', g(p'))$ such that $q' = \pi(q)$. On the other hand, since P is stable modulo symmetry, then for any point q of $Nbhd(p, g(p))$ there is a point $q'' \in P$ symmetric to q and so $q = \pi^*(q'')$, $\pi^* \in G$ (π^* may be equal to $1 \in G$ if q is in P). Then $q' = \pi(\pi^*(q''))$. Hence q' is symmetric to $q'' \in P$ and so $q' \in K(P)$. This means that $Nbhd(p', g(p')) \subseteq K(P)$ and so $K(P)$ is stable. Then according to Proposition 8, the CNF formula F is unsatisfiable.

REMARK 17 *The idea of the proof was suggested to the author by Howard Wong-Toi [22].*

Proposition 16 *Let $P \subseteq B^n$ be a set of points that is stable with respect to a CNF formula F and transport function $g : P \rightarrow F$. Let P' be a subset of P such that for each point p of P that is not in P' there is a point $p' \in P'$ symmetric to p. Then P' is stable with respect to F and g modulo symmetry.*

Proof Let p' be a point of P'. Let q' be a point of $Nbhd(p',g(p'))$. Point p' is in P because $P' \subseteq P$. Since P is a stable set then $q' \in P$. From the definition of the set P' it follows that if q' is not in P' then there is a point $r' \in P'$ that is symmetric to q'. So each point q' of $Nbhd(p', g(p'))$ is either in P' or there is a point of P' that is symmetric to q'.

Definition 29 *Let F be a CNF formula, G be its group of permutations, p be a point of $Z(F)$, and g: $P \rightarrow F$ be a transport function. A set $Reachable(p, g, G)$ is called the set of points* ***reachable from p modulo symmetry*** *if a) the point p is in $Reachable(p, g, G)$ b) each point p' that is reachable from p by means of the transport function g is either in $Reachable(p, g, G)$ or there exists a point $p'' \in Reachable(p, g, G)$ that is symmetric to p'.*

Proposition 17 *Let F be a CNF formula, G be its group of permutations, p be a point of $Z(F)$, and $g : P \rightarrow F$ be a transport function. If the set P=$Reachable(p, g, G)$ does not contain a satisfying assignment, then it is stable modulo symmetry with respect to F and g and so F is unsatisfiable.*

Proof Assume the contrary, i.e. that P is not stable modulo symmetry. Then there is a point $p' \in P$ (reachable from p modulo symmetry) such that a point p'' of $Nbhd(p',g(p'))$ is not in P and P does not contain a point symmetric to p''. On the other hand, p'' is reachable from p' and so it is reachable from p modulo symmetry. We have a contradiction.

Remark 18 *From Proposition 17 it follows that a CNF F that is symmetric with respect to a group of permutations G is satisfiable if and only if, given a point $p \in Z(F)$, a transport function $g : Z(F) \rightarrow F$, the set $Reachable(p, g, G)$ contains a satisfying assignment.*

Let F be a CNF formula and G be its group of permutations. According to Proposition 17 when testing the satisfiability of F it is sufficient to construct a set $Reachable(p, g, G)$. This set can be built by the algorithm of Section 1.2.4 in which step 5 is modified in the following way. Before adding a point p'' from $Nbhd(p', C) \backslash (Body \cup Boundary)$ to $Boundary$ it is checked if there is a point p^* of $Boundary \cup Body$ that is symmetric to p''. If such a point exists, then p'' is not added to $Boundary$.

For highly symmetric formulas the difference between the SSPs and SSPs modulo symmetry can be huge. For example, for pigeon-hole formulas the size of SSPs is exponential in the number of holes while the size of minimum SSPS modulo symmetry is linear in the number of holes [11].

2.6 SSPs with Excluded Directions

Unfortunately, the theory developed in Section 1.2.5 does not help in solving CNF formulas that have no (or have very few) symmetries. In this section, we describe a different way of reducing the size of SSPs. The idea is to replace the computation of a single SSP with the construction of a sequence of SSPs whose stability is "limited". These SSPs are called SSPs with excluded directions. The key point is that by excluding some directions from consideration one can drastically reduce the size of SSPs. The construction of an SSP with excluded directions allows one to generate a new clause that is an implicate of the initial CNF formula. This clause can be added to the current formula, which makes the obtained formula simpler in terms of the size of SSPs. For the new formula we can again build an SSP with excluded directions deducing a new implicate of the formula. A sketch of the procedure of satisfiability testing based on constructing SSPs with excluded directions is given at the end of the section.

DEFINITION 30 *Let F be a CNF formula.* ***A set of excluded directions*** *is a set E of literals that a) does not contain opposite literals of the same variable; b) there is no clause C of F such that all literals of C are in E.*

DEFINITION 31 *Let F be a CNF formula and C be a clause of F. Let E be a set of excluded directions. Denote by* ***Nbhd(p,C,E)*** *the set of points of $Nbhd(p,C)$ that set to 1 only the literals of C that are not in E.*

REMARK 19 *Since, according to Definition 30, there is at least one literal of C that is not in E, then $Nbhd(p,C,E)$ is nonempty.*

EXAMPLE 7 *Let a point p be equal to $(x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 1, x_6 = 1)$. Let a clause C of a CNF F be equal to $x_1 \vee x_3 \vee \overline{x_6}$ and the set E of excluded directions be equal to $\{x_4, \overline{x_6}\}$. The set $Nbhd(p,C)$ consists of points p_1, p_2 and p_3 obtained from p by flipping the values of variables x_1, x_3, x_6 respectively. On the other hand, set $Nbhd(p,C,E)$ consists only of points p_1, p_2 because the point p_3 sets to 1 an "excluded" literal, namely the literal $\overline{x_6}$ of E.*

Definition 32 *Let P be a nonempty subset of $Z(F)$, F be a CNF formula, and g: $P \rightarrow F$ be a transport function. Let E be a set of excluded directions. The set P is called* ***stable with respect to F, g and E*** *if a) each point p of P sets all the literals of E to 0; b) for each point p of P, $Nbhd(p, g(p), E) \subseteq P$.*

Proposition 18 *If there is a set of points that is stable with respect to a CNF formula F and a set E of excluded directions, then any assignment satisfying F has to set to 1 at least one literal of E. In other words, the clause obtained by the disjunction of the literals of E is an implicate of F.*

Proof Let P be a set of points that is stable with respect to F, a transport function g and a set E of excluded directions. Make the assignments setting all the literals of E to 0. Remove from F all the clauses that are satisfied by these assignments and remove from the rest of the clauses all the literals that are in E (since they are set to 0). The obtained formula F' is unsatisfiable because the set P is stable with respect to F' and a transport function g'. Indeed, according to Definition 31, each point p of P sets all the literals of E to 0. Then the clause $C = g(p)$ of F cannot be satisfied by the assignment setting a literal l of E to 0. (If a clause C is satisfied by this assignment, it must contain the literal $\overline{l}$ but then C cannot be falsified by p.) So all the clauses assigned to the points of P by g are still in F'. Denote by g' the transport function that maps a point p of P to the clause C' obtained from the clause $C = g(p)$ by removing all the literals of E. It is not hard to see that $Nbhd(p, C') = Nbhd(p, C, E)$. So for each point p of P it is true that $Nbhd(p, g'(p)) \subseteq P$.

Remark 20 *A set of points stable with respect to a CNF F and a set E of excluded directions can be constructed by the algorithm of Section 1.2.4 modified in the following way. At step 1 the algorithm generates a starting point setting all the literals from E to 0. At step 5 it generates set $Nbhd(p', C, E)$ instead of $Nbhd(p', C)$.*

Example 8 *Let $p_1 = (x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0, x_5 = 0, x_6 = 0, x_7 = 0)$. Let F be a CNF formula containing clauses $C_1 = x_1 \vee x_2 \vee x_3, C_2 = \overline{x_1} \vee x_4 \vee x_5$ (and maybe some other clauses). Let the set E of excluded directions be equal to $\{x_2, x_3, x_4, x_5\}$. Denote by p_2 the point obtained from p_1 by flipping the value of x_1. Taking into account that p_1 falsifies clause C_1 and p_2 falsifies clause C_2 we can form the following transport function g: $g(p_1) = C_1, g(p_2) = C_2$. It is not hard to see that the set of points $P = \{p_1, p_2\}$ is stable with respect to clauses C_1, C_2, transport function g, and set E. Indeed, since literals x_2 and x_3 of C_1 are in E then $Nbhd(p_1, g(p_1), E) = \{p_2\} \subseteq P$. On the other hand, since* *literals x_4 and x_5 of C_2 are in E then $Nbhd(p_2, g(p_2), E) = \{p_1\} \subseteq P$. From Proposition 18 we conclude that the clause $C = x_2 \vee x_3 \vee x_4 \vee x_5$ equal to the disjunction of literals of E is an implicate of the formula F. On the other hand, it is not hard to see that C is actually the resolvent of clauses C_1 and C_2.*

REMARK 21 *From Example 8 it follows that for an unsatisfiable formula F we can always choose a set E of excluded directions so that there is a set of two points that is stable with respect to F and E. Indeed, due to completeness of general resolution, in F there is always a pair of clauses C_1 and C_2 that produce a new resolvent. Then we form the set E of excluded directions consisting of all the literals of C_1 and C_2 except the literals of the variable in which the two clauses are resolved.*

Below we sketch a procedure of satisfiability testing based on computing SSPs with excluded directions.

1 Compute an SSP P of a limited size trying to minimize the set E of excluded directions

2 Stop if a satisfying assignment is found. The formula is satisfiable.

3 Stop if $E = \emptyset$. The formula is unsatisfiable.

4 Add the deduced clause (disjunction of the literals of E) to the current CNF formula.

5 Go to step 1.

The idea of the procedure is that adding new implicates gradually reduces the complexity of the initial formula F in terms of the size of "monolithic" SSPs. The claim that the size of SSPs decreases is based on the following observations. Any set of points that is stable with respect to a CNF formula F is also stable with respect to a CNF $F \cup \{C\}$ where C is a clause. So by adding clauses we preserve the best SSPs seen so far and may produce even smaller ones. The latter follows from the fact that by adding new implicates we will eventually produce an empty clause (at step 3 of the procedure above) and any set of clauses containing an empty clause has an SSP consisting of only one point.

An important advantage of obtaining new implicates by computing SSPs with excluded directions is that directions can be excluded on the fly. The choice of directions to exclude should be aimed at the reduction of the size of the constructed SSP (that is the directions that may lead to the blow-up of the SSP should be excluded). Besides, when excluding directions one can make use of the information about the structure of the CNF formula to be tested for satisfiability.

2.7 Conclusions

In the second part of this chapter we show that satisfiability testing of a CNF formula reduces to constructing a stable set of points (SSP). An SSP of a CNF formula can be viewed as an inherent characteristic of this formula. We give a simple procedure for computing an SSP. As a practical application we show that the proposed procedure of SSP construction can be easily modified to take into account symmetry (with respect to variable permutation) of CNF formulas. Finally, we introduce the notion of an SSP with excluded direction and describe a procedure of satisfiability testing based on constructing such SSPs. We believe that developing the theory of SSPs may lead to creating SAT-algorithms that are much more efficient and "intelligent" than the ones implemented in the state-of-the-art SAT-solvers.

References

[1] BerkMin web page. http://eigold.tripod.com/BerkMin.html

[2] Bonet M.,Pitassi T., Raz R. *On interpolation and automatization for Frege systems.* SIAM Journal on Computing, 29(6):1939-1967, 2000.

[3] Brand D. *Verification of large synthesized designs.* Proceedings of ICCAD-1993, pp. 534-537.

[4] Bryant R. *Graph based algorithms for Boolean function manipulation.* IEEE Trans. on Computers, C(35):677-691.

[5] V.Chvatal, E.Szmeredi. *Many hard examples for resolution.* J. of the ACM,vol. 35, No 4, pp.759-568.

[6] CUDD web page. http://vlsi.colorado.edu/~fabio/

[7] M.Davis, G.Logemann, D.Loveland. *A Machine program for theorem proving.* Communications of the ACM, 1962,vol. 5,pp. 394-397.

[8] Goldberg E., Novikov Ya. *BerkMin: A fast and robust SAT-solver.* Design, Automation, and Test in Europe (DATE '02), pp. 142-149, March 2002.

[9] Goldberg E., Novikov Ya. *How good are current resolution based SAT-solvers.* presented at SAT-2003,Margherita Ligure - Portofino (Italy), May 5-8,2003.

[10] Goldberg E., Novikov Ya. *Equivalence Checking of Dissimilar Circuits.* Presented at IWLS-2003. Laguna Beach, California, USA,May 28-30,2003.

[11] E. Goldberg. *Testing Satisfiability of CNF Formulas by Computing a Stable Set of Points.* Proceedings of Conference on Automated Deduction, CADE 2002, pp.161-180.

[12] E. Goldberg. *Proving Unsatisfiability of CNFs locally.* Journal of Automated Reasoning. vol 28:417-434, 2002.

[13] A.Haken. *The intractability of resolution.* Theor. Comput. Sci. 39 (1985),297-308.

[14] F. Lu, L.-C. Wang, K.-T. Cheng, R. Huang. *A circuit SAT solver with signal correlation guided learning*, DATE-2003, pp. 892-898.

[15] D.Mitchell, B.Selman, H.J.Levesque. *Hard and easy distributions of SAT problems.* Proceedings AAAI-92, San Jose,CA, 459-465.

[16] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, S. Malik. *Chaff: Engineering an efficient SAT-solver.* Proceedings of DAC-2001,pp. 530-535.

[17] C.Papadimitriou. *On selecting a satisfying truth assignment.* Proceedings of FOCS-91, pp. 163-169

[18] Razborov A., Alekhnovich M. *Resolution is not automatizable unless W[p] is tractable.* Proc. of the 42^{nd} IEEE FOCS-2001, pages 210-219.

[19] B.Selman, H.Kautz, B.Cohen. *Noise strategies for improving local search.* Proceedings of AAAI-94,Vol. 1, pp. 337-343.

[20] E. Sentovich, K. Singh, C. Moon, H. Savoj, R. Brayton, A. Sangiovanni -Vincentelli, *Sequential circuit design using synthesis and optimization.* Proceedings of ICCAD, pp 328-333, October 1992.

[21] Silva J., Sakallah K. *GRASP: A Search Algorithm for Propositional Satisfiability.* IEEE Transactions of Computers, 1999, Vol. 48,pp. 506-521.

[22] H.Wong-Toi. *Private communication.*

[23] H.Zhang. SATO: *An efficient propositional prover.* Proceedings of CADE-1997, pp. 272-275.

Index