Session 1

Verification of Combinational Circuits

Satisfiability Solver

Albert-Ludwigs-Universität Freiburg

Literatur

- K. Erk, L. Priese: Theoretische Informatik – Eine umfassende Einfhrung, Springer-Verlag, 2002.
- U. Schning: Logik für Informatiker, Spektrum Akademischer Verlag, 1995.
- T. Schubert: SAT-Algorithmen und Systemaspekte: vom Mikroprozessor zum parallelen System, Dissertation, 2008.
- M. Lewis: SAT, QBF, and Multicore processors, Dissertation, 2010.
- Diverse Originalarbeiten

Definition (Propositional Logic and Syntax)

Given a set of variables $x_1, \ldots, x_n$ we start to define our propositional logic as follows:

1. Every variable x_i is an atomic formula.
2. For all formulas F_1 and F_2 we have
   - the Conjunction $(F_1 \wedge F_2)$ and
   - the Disjunction $(F_1 \vee F_2)$ propositional logic operators.
3. For every Formula F, we have its negation $(\neg F)$.
4. The total set of Formulas we consider are only those that can be generated using the above three rules.

Definition (Semantics of Propositional Logic)

An assignment $\mathscr{A}_x : \{x_1, \ldots, x_n\} \to \{0,1\}$ is a mapping from all the propositional variables $x_1, \ldots, x_n$ to their assignment (0 or 1). Extending $\mathscr{A}_x$ to $\mathscr{A} : \{F \mid F \text{ Formula}\} \to \{0,1\}$, we map every propositional formula F to the set $\{0,1\}$ according to the following rules:

1. For every F which contains variables x_i, it holds that:
   - $\mathscr{A}(x_i) = \mathscr{A}_x(x_i)$.
2. For all sub-formulas F_1 and F_2 from F, it holds that:
   - $\mathscr{A}(F_1 \wedge F_2) = 1 \Leftrightarrow \mathscr{A}(F_1) = 1$ and $\mathscr{A}(F_2) = 1$.
   - $\mathscr{A}(F_1 \vee F_2) = 1 \Leftrightarrow \mathscr{A}(F_1) = 1$ or $\mathscr{A}(F_2) = 1$.
3. For every sub-formula F' of F:
   - $\mathscr{A}(\neg F') = 1 \Leftrightarrow \mathscr{A}(F') = 0$.

Definition (Satisfiability)

- A Formula F in propositional logic is satisfiable when a mapping $\mathscr{A}$ for $\mathscr{A}(F) = 1$ exists.
- Commonly, such a mapping is referred to as a model of F, which is represented by $\mathscr{A} \models F$.
- If no assignment $\mathscr{A}$ for $\mathscr{A}(F) = 1$ exists, then F is unsatisfiable, and for all assignments $\mathscr{A}$, $\mathscr{A} \not\models F$ holds.

Propositional Logic

Definition (Literal)

A literal L is the positive ($L = x_i$) or negative ($L = \neg x_i$) occurrence of a variable in a formula.

Definition (Clause)

A formula $C = (L_1 \vee \ldots \vee L_k)$, containing literals $L_1, \ldots, L_k$ will from now on be referred to as a clause.

Definition (Conjunctive Normal Form, CNF)

A formula F in propositional logic is in conjunctive normal form when it consists of a conjunction of clauses:

$$F = \bigwedge_{j=1}^{m} C_j \qquad \text{with } C_1, \ldots, C_m \text{ clauses}$$

- Example: $(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_4)$
- An assignment $\mathscr{A}$ satisfies a CNF formula F, only when it also satisfies all the clauses in F.

The Propositional Satisfiability Problem

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Definition (SAT-Problem)

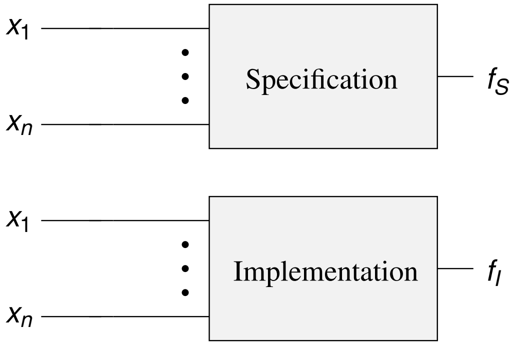
Given a formula F as defined earlier, the question we are now considering is: Does there exists an assignment $\mathscr{A}$ for the variables in F such that $\mathscr{A}(F) = 1$? If so, F is satisfiable.

- This question/problem is commonly referred to as:
  - Satisfiability Problem
  - SAT-Problem
- Similarly, the terms for propositional and Boolean formulas will be used equally.
- Also, a method or algorithm used to solve individual SAT problems is called a SAT solver.

SAT and Verification of Combinational Circuits

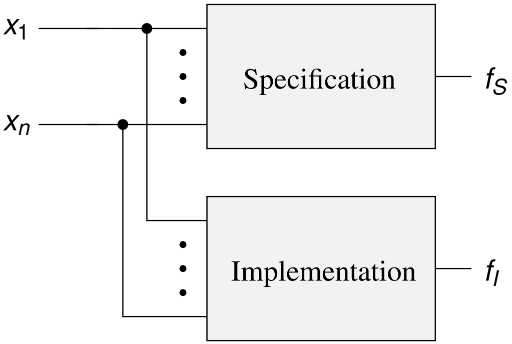
- Given:
  - A specification and an implementation of a combinational circuit.
- Question:
  - Are the specification and implementation **functionally** equivalent?
- Using SAT based methods to prove equivalence
  - Using the specification and implementation, generate a so called Miter circuit.
  - Convert the Miter circuit into a Boolean formula.
  - Solve the formula with SAT Algorithm (SAT Solver).
- The specification and implementation of a combinational circuit are functionally equivalent when the Boolean formula representing the Miter circuit is unsatisfiable.

Construction of the Miter Circuit



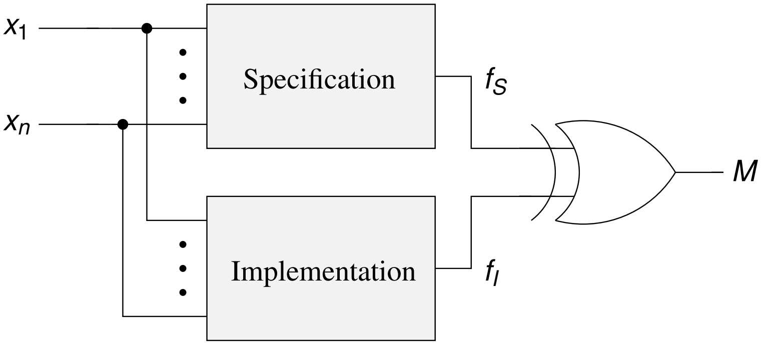
$\Rightarrow$ Connect the corrisponding inputs.

Construction of the Miter Circuit



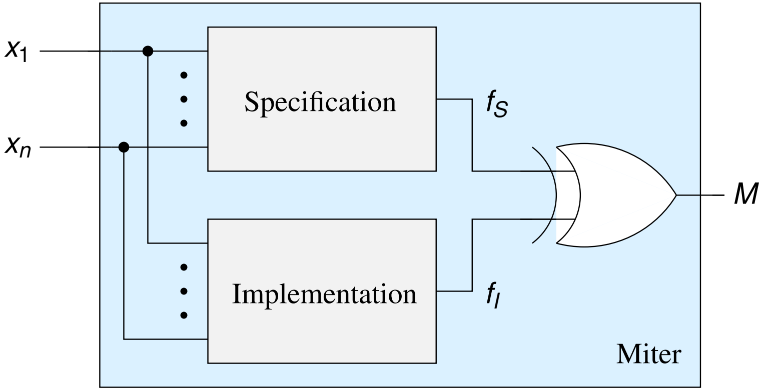
$\Rightarrow$ Compare the outputs using an XOR gate.

Construction of the Miter Circuit



$\Rightarrow$ Miter Circuit

Construction of the Miter Circuit



$\Rightarrow M = 1 \Leftrightarrow$ specification and implementation are not equal.

Verification of Combination Circuits using SAT

Notes:

- The outlined procedure can be extended to circuits with more than one output.
- Most SAT algorithms accept only formulas in CNF form as an input. As such, our Miter circuits need to be converted to, and represented as, a CNF formula.
- Regarding equivalence checking algorithms, BDD based approaches are memory restricted. In contrast, search based SAT methods are time limited.

Verification of Combination Circuits using SAT

Next:

- Consider how to convert and represented the Miter circuit as a Boolean formula in CNF form.
- Complexity of solving SAT problems.
- Describe how SAT solvers and algorithms work.

Converting a Propositional Formula into CNF

Definition (Equivalence)

Two formulas F and G in propositional logic are equivalent ($F \equiv G$), iff for all possible assignments $\mathscr{A}$, $\mathscr{A}(F) = \mathscr{A}(G)$ holds.

Theorem

For every formula F in propositional logic, an equivalent formula F' in CNF form can be produced.

Proof.

Using induction and our Formula constuction rules, we can show that this is indeed the case. □

Converting a Propositional Formula into CNF

- Given: a propositional logic F
- Conversion
  1. In F, replace every occurance of the sub-formalas having the form:

     $\neg\neg F_1$ with F_1;

     $\neg(F_1 \wedge F_2)$ with $(\neg F_1 \vee \neg F_2)$;

     $\neg(F_1 \vee F_2)$ with $(\neg F_1 \wedge \neg F_2)$;

     until these types of sub-formulas in F no longer exist.
  2. In F replace every occurance of the sub-formalas having the form:

     $F_1 \vee (F_2 \wedge F_3)$ with $(F_1 \vee F_2) \wedge (F_1 \vee F_3)$;

     $(F_1 \wedge F_2) \vee F_3$ with $(F_1 \vee F_3) \wedge (F_2 \vee F_3)$;

     until these types of sub-formulas in F no longer exist.
- Results: A formula F' in CNF form that is equivalent to F.

Converting a Propositional Formula into CNF

Definition (Size of a Formula)

The size of a formula F of our declared logic (shown as $|F|$), is defined as the number of operators $\Diamond$ in F, where $\Diamond \in \{\wedge, \vee, \neg\}$.

Theorem

For every propositional logic formula of our form, with a size of $(2 \cdot m - 1)$, there exist an equivalent formula in CNF form with a maximum size of $(m \cdot 2^m - 1)$.

Converting a Propositional Formula into CNF

Proof.

Consider the following conversion method:

$$F_m = \bigvee_{j=1}^{m} (L_{j,1} \wedge L_{j,2})$$

where $L_{j,1} \neq L_{j,2}$, and in this case only containing positive literals $L_{1,1}$, $L_{1,2}$, …, $L_{m,1}$, $L_{m,2}$. The size of such a formula is obviously $(2 \cdot m - 1)$. A minimal formula that is equivalents, and in conjunctive normal is shown in F'_m.

$$F'_m = \bigwedge_{k_1,\ldots,k_m \in \{1,2\}} (L_{1,k_1} \vee \ldots \vee L_{m,k_m})$$

F'_m has a total of 2^m clauses. For the conjunctions of all the clauses, $(2^m - 1)$ AND operators are needed. Since every clause has m literals, $(m-1)$ OR operators are needed for every clause. Therefore, the size of formula F'_m is:

$$|F'_m| = 2^m - 1 + 2^m \cdot (m-1) = m \cdot 2^m - 1.$$

□

Converting a Propositional Formula into CNF

Example 1

- Given:
  - $F_1 = x_1\, x_2 \vee x_3\, x_4$
  - $m = 2$
  - $|F_1| = (2 \cdot m - 1) = 3$
- Conversion of F_1 into an equivalent CNF F', with $F_1 \equiv F'$:
  - $F' = (x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (x_1 \vee x_4) \wedge (x_2 \vee x_4)$
  - $|F'| = (m \cdot 2^m - 1) = 7$

Converting a Propositional Formula into CNF

Example 2

- Given
  - $F_2 = x_1\, x_2 \vee x_3\, x_4 \vee \ldots \vee x_{17}\, x_{18} \vee x_{19}\, x_{20}$
  - $m = 10$
  - $|F_2| = 19 = (2 \cdot m - 1)$
- For F_2, the CNF representation F'' has a size of:
  - $|F''| = (m \cdot 2^m - 1) = (10 \cdot 2^{10} - 1) = (10 \cdot 1024 - 1) = 10239$

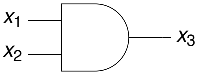
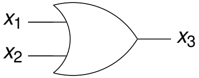
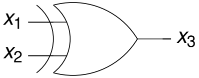
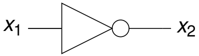
Tseitin-Transformation

To avoid the possible exponential size of the CNF representation of a circuit (represented by a function F), the following alternative approach can be applied:

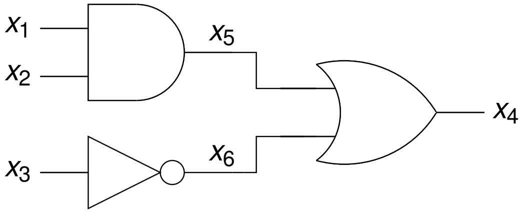
- Construct a formula F' that is satisfiably equivalent to F. Meaning if F is satisfiable, then F' is satisfiable.
- For each gate, intermediate "helper" variables are introduced into the CNF F', which do not appear in F.
- For each gate a "characteristic function" which is in CNF form will be substituted for every occurrence of that particular gate. The "characteristic function" will evaluate to 1, iff the assignments of the respective gate signal would also cause the output gate to go to 1.
- To generate the final CNF for the entire circuit, all the gate functions that are used will be ANDed together.

$\Rightarrow$ Tseitin-Transformation

Tseitin-Transformation

Gate	Function	CNF Formula
	$x_3 \equiv x_1 \wedge x_2$	$(\neg x_3 \vee x_1) \wedge (\neg x_3 \vee x_2) \wedge (x_3 \vee \neg x_1 \vee \neg x_2)$
	$x_3 \equiv x_1 \vee x_2$	$(x_3 \vee \neg x_1) \wedge (x_3 \vee \neg x_2) \wedge (\neg x_3 \vee x_1 \vee x_2)$
	$x_3 \equiv x_1 \oplus x_2$	$(\neg x_3 \vee x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee \neg x_1 \vee x_2) \wedge (x_3 \vee x_1 \vee \neg x_2)$
	$x_2 \equiv \neg x_1$	$(x_2 \vee x_1) \wedge (\neg x_2 \vee \neg x_1)$

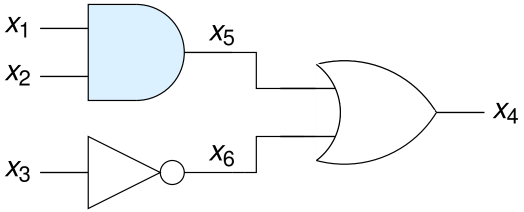
Tseitin-Transformation



$$F_{SK} = (x_1 \wedge x_2) \vee \neg x_3$$

$$\begin{aligned} F_{SK}^{CNF} = {} & (\neg x_5 \vee x_1) \wedge (\neg x_5 \vee x_2) \wedge (x_5 \vee \neg x_1 \vee \neg x_2) \wedge \\ & (x_6 \vee x_3) \wedge (\neg x_6 \vee \neg x_3) \wedge \\ & (x_4 \vee \neg x_5) \wedge (x_4 \vee \neg x_6) \wedge (\neg x_4 \vee x_5 \vee x_6) \end{aligned}$$

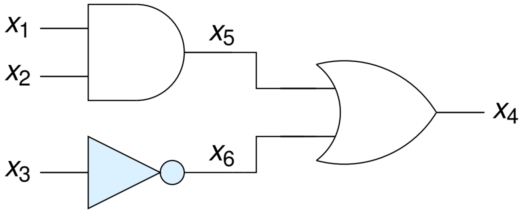
Tseitin-Transformation



$$F_{SK} = (x_1 \wedge x_2) \vee \neg x_3$$

$$\begin{aligned} F_{SK}^{CNF} = {} & (\neg x_5 \vee x_1) \wedge (\neg x_5 \vee x_2) \wedge (x_5 \vee \neg x_1 \vee \neg x_2) \wedge \\ & (x_6 \vee x_3) \wedge (\neg x_6 \vee \neg x_3) \wedge \\ & (x_4 \vee \neg x_5) \wedge (x_4 \vee \neg x_6) \wedge (\neg x_4 \vee x_5 \vee x_6) \end{aligned}$$

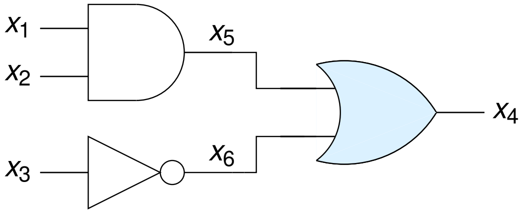
Tseitin-Transformation



$$F_{SK} = (x_1 \wedge x_2) \vee \neg x_3$$

$$\begin{aligned} F_{SK}^{CNF} = {} & (\neg x_5 \vee x_1) \wedge (\neg x_5 \vee x_2) \wedge (x_5 \vee \neg x_1 \vee \neg x_2) \wedge \\ & (x_6 \vee x_3) \wedge (\neg x_6 \vee \neg x_3) \wedge \\ & (x_4 \vee \neg x_5) \wedge (x_4 \vee \neg x_6) \wedge (\neg x_4 \vee x_5 \vee x_6) \end{aligned}$$

Tseitin-Transformation



$$F_{SK} = (x_1 \wedge x_2) \vee \neg x_3$$

$$\begin{aligned} F_{SK}^{CNF} = & (\neg x_5 \vee x_1) \wedge (\neg x_5 \vee x_2) \wedge (x_5 \vee \neg x_1 \vee \neg x_2) \wedge \\ & (x_6 \vee x_3) \wedge (\neg x_6 \vee \neg x_3) \wedge \\ & (x_4 \vee \neg x_5) \wedge (x_4 \vee \neg x_6) \wedge (\neg x_4 \vee x_5 \vee x_6) \end{aligned}$$

Tseitin-Transformation

As long as the CNF representation of each gate only consist of a fixed number of clauses, the number of clauses required for the entire CNF formula will grow linearly with respect to the number of gates in the circuit (also valid for the size of the formula).

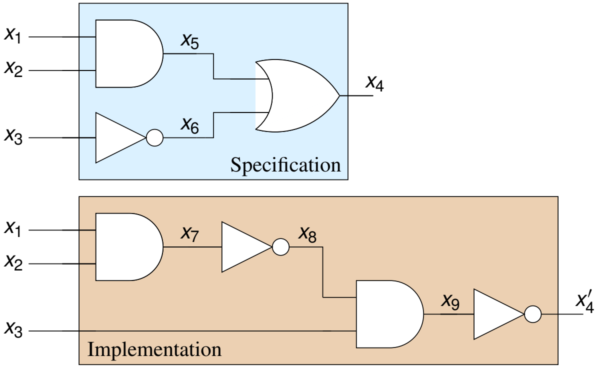
Tseitin-Transformation

Equivalence checking using satisfiably equivalent CNF representations.

- Given:
  - $F = x_1\,x_2 \vee x_3\,x_4 \vee \ldots \vee x_{17}\,x_{18} \vee x_{19}\,x_{20}$
  - $|F| = 19 = (2 \cdot m - 1)$ with $m = 10$
- Conversion of F into an equivalent CNF F' with $F \equiv F'$:
  - $|F'| = (m \cdot 2^m - 1) = (10 \cdot 2^{10} - 1) = (10 \cdot 1024 - 1) = 10239$
- Tseitin-Transformation from F into satisfiably equivalent CNF F'':
  - $|F''| = \underbrace{100}_{10\,UND-Gatter} + \underbrace{81}_{9\,ODER-Gatter} + \underbrace{18}_{18\wedge} = 199$

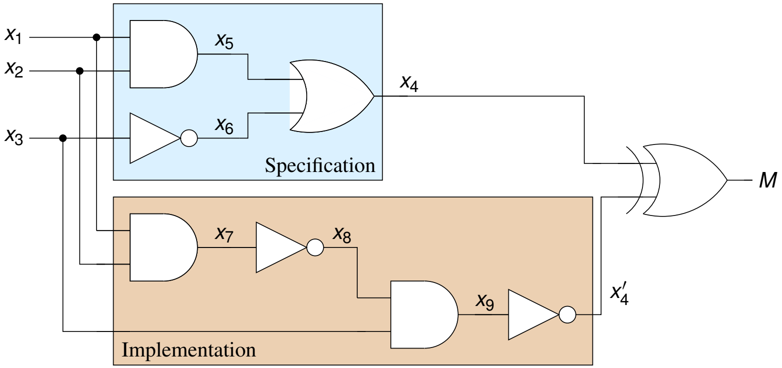
Verification of Combinational Circuit using SAT

Given the following specification and implementation of a combinational circuit:



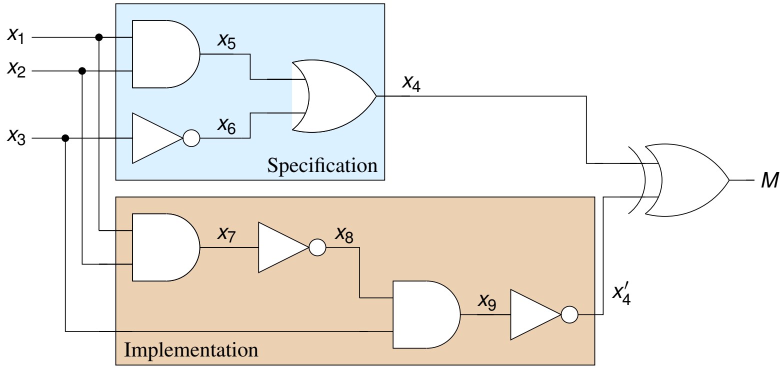
Question: Are the specification and implementation functionally equivalent?

Verification of Combinational Circuit using SAT



$$\begin{aligned}F_M = {} & (\neg x_5 \vee x_1) \wedge (\neg x_5 \vee x_2) \wedge (x_5 \vee \neg x_1 \vee \neg x_2) \wedge (x_6 \vee x_3) \wedge (\neg x_6 \vee \neg x_3) \wedge \\ & (x_4 \vee \neg x_5) \wedge (x_4 \vee \neg x_6) \wedge (\neg x_4 \vee x_5 \vee x_6) \wedge (\neg x_7 \vee x_1) \wedge (\neg x_7 \vee x_2) \wedge \\ & (x_7 \vee \neg x_1 \vee \neg x_2) \wedge (x_7 \vee x_8) \wedge (\neg x_7 \vee \neg x_8) \wedge (\neg x_9 \vee x_3) \wedge (\neg x_9 \vee x_8) \wedge \\ & (x_9 \vee \neg x_3 \vee \neg x_8) \wedge (x_9 \vee x_4') \wedge (\neg x_9 \vee \neg x_4') \wedge (\neg M \vee \neg x_4 \vee \neg x_4') \wedge \\ & (\neg M \vee x_4 \vee x_4') \wedge (M \vee \neg x_4 \vee x_4') \wedge (M \vee x_4 \vee \neg x_4') \wedge (M)\end{aligned}$$

Verification of Combinational Circuit using SAT



$$
\begin{aligned}
F_M = {} & (\neg x_5 \vee x_1) \wedge (\neg x_5 \vee x_2) \wedge (x_5 \vee \neg x_1 \vee \neg x_2) \wedge (x_6 \vee x_3) \wedge (\neg x_6 \vee \neg x_3) \wedge \\
& (x_4 \vee \neg x_5) \wedge (x_4 \vee \neg x_6) \wedge (\neg x_4 \vee x_5 \vee x_6) \wedge (\neg x_7 \vee x_1) \wedge (\neg x_7 \vee x_2) \wedge \\
& (x_7 \vee \neg x_1 \vee \neg x_2) \wedge (x_7 \vee x_8) \wedge (\neg x_7 \vee \neg x_8) \wedge (\neg x_9 \vee x_3) \wedge (\neg x_9 \vee x_8) \wedge \\
& (x_9 \vee \neg x_3 \vee \neg x_8) \wedge (x_9 \vee x_4') \wedge (\neg x_9 \vee \neg x_4') \wedge (\neg M \vee \neg x_4 \vee \neg x_4') \wedge \\
& (\neg M \vee x_4 \vee x_4') \wedge (M \vee \neg x_4 \vee x_4') \wedge (M \vee x_4 \vee \neg x_4') \wedge (M)
\end{aligned}
$$

F_M is unsatisfiable $\Rightarrow$ specification and implementation equivalent!

Complexity and SAT Problems

- A CNF formula belongs to the class of “k-SAT” problems iff each clause in the formula has exactly k literals.
- S.A. Cook, 1971: 3-SAT Problem is NP-Complete
- Therefore, in “general”, the SAT problem is NP-Complete as all CNF formulas can be converted into a 3-SAT problem.
- In special cases, we can solve the problems in linear or polynomial time:
  - 2-SAT (formulas contain only binary clauses).
  - Horn-Formula (every clause in the formula contains a maximum of one positive literal).
  - ...

Complexity and SAT Problems

Observations in practice:

- Modern SAT algorithms are now able to solve many industrially relevant and academically interesting problems in a reasonable amount of time.
- Commonly, industrial problems with 100,000's of variables, and millions of clauses can be solved.

Applications for SAT algorithms:

- Combinational Equivalence Checking
- Automatic Test Pattern Generation
- Bounded Model Checking, Model Checking
- AI Planning
- ...

Overview of SAT Algorithms

Complete Algorithms (vs Incomplete)

- Ability - Can prove the unsatisfiability of a CNF-formula due to a systematic approach the solver uses.
- DP Algorithm
  - M. Davis, H. Putnam, 1960
  - Based on resolution
- DLL Algorithm
  - M. Davis, G. Logemann, D. Loveland, 1962
  - Based on a depth first search
- Modern SAT Algorithms
  - Based on the DLL Algorithm, however, they include powerful resolution techniques, efficient data structures, and many more acceleration techniques.
  - zChaff, MiniSat, MiraXT, precosat, lingeling, antom

Overview of SAT Algorithms

Incomplete Algorithms

- Normally based on local searches.
- Basic concept:
  - Generate an initial variable assignment.
  - Until the formula is satisfied, keep modifying the assignments using some heuristic (i.e. “flip” the value of a specific variable).
- GSat, WSat (H.A. Kautz, B. Selman, 1992 & 1996)
- Cannot in general prove a formula is unsatisfiable.

$\Rightarrow$ Will not be considered further in these talks!

Notation

- A clause $C = (L_1 \vee \ldots \vee L_n)$ can be regarded as a set of literals: $C = (L_1, \ldots, L_n)$.
- The empty clause, represented by $\square$, describes the empty set of literals and is by definition unsatisfiable.

Notation

- The union of two clauses (C_1 and C_2) results in a new clause (C_3) that contains all the literals of both previous clauses:

$$C_3 = C_1 \cup C_2 = \{L \mid (L \in C_1) \vee (L \in C_2)\}$$

  Literals that occur in both C_1 and C_2, only appear once in C_3. This is a form of simplification.

- The difference of two clauses is defined as follows:

$$C_1 - C_2 = \{L \mid (L \in C_1) \wedge (L \notin C_2)\}$$

Notation

- A CNF formula $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ can be regarded as a set of clauses: $F = \{C_1, C_2, \ldots, C_m\}$
- An empty Formula describes an empty set of clauses, and by definition is satisfiable.
- The union of two CNF formulas (F_1 and F_2) results in a CNF formula F_3 that contains all the clauses from both previous formulas:

$$F_3 = F_1 \cup F_2 = \{C \,|\, (C \in F_1) \vee (C \in F_2)\}$$

Again, clauses that appear in both F_1 and F_2 will only be represented once in F_3.

Resolution

Definition (Resolution)

Given two clauses C_1 and C_2, and a literal L with the following property: $L \in C_1$ and $\neg L \in C_2$, then it is possible to build a clause R:

$$R = (C_1 - \{L\}) \cup (C_2 - \{\neg L\})$$

R is referred to as the resolvent of the clauses C_1 and C_2 on literal L. Using our notation, this is represented by:

$$R = C_1 \otimes_L C_2$$

Resolution

Example 3

- $C_1 = (x_1, x_2, x_3),\ C_2 = (x_4, \neg x_2) \Rightarrow R_1 = C_1 \otimes_{x_2} C_2 = (x_1, x_3, x_4)$
- $C_3 = (x_4, x_2, x_3),\ C_4 = (x_4, \neg x_2) \Rightarrow R_2 = C_3 \otimes_{x_2} C_4 = (x_3, x_4)$
- $C_5 = (x_4, x_2),\ C_6 = (\neg x_4, \neg x_2) \Rightarrow R_3 = C_5 \otimes_{x_2} C_6 = (x_4, \neg x_4)$
- $(x_4, \neg x_4)$ is for every assignment of x_4 satisfied, and is therefore referred to as a tautological clause.

Resolution

Lemma (Resolution Lemma)

Given a CNF formula F and the resolvent R of two clauses C_1 and C_2 from F, then it must be the cast that F and $F \cup \{R\}$ are equivalent: $F \equiv F \cup \{R\}$.

Proof.

Given a set of assignments $\mathscr{A}$ that satisfies the formula $F \cup \{R\}$: $\mathscr{A} \models F \cup \{R\}$. Then it must also be the case that $\mathscr{A} \models F$.

So assume that the assignments $\mathscr{A}$ satisfies the formula F. This means that all the clauses $C_i \in F$ are also satisfied. Furthermore, assume the resolvent R was constructed as $R = (C_1 - \{L\}) \cup (C_2 - \{\neg L\})$, with $C_1, C_2 \in F$, $L \in C_1$ and $\neg L \in C_2$.

To prove the equivalence of the two, we have to distinguishes between two cases in $\mathscr{A} \models F$. Either $\mathscr{A} \models L$ or $\mathscr{A} \models \neg L$.

1. $\mathscr{A} \models L$. Because $\mathscr{A} \models C_2$ and $\mathscr{A} \not\models \neg L$ it follows that $\mathscr{A} \models (C_2 - \{\neg L\})$. As such, the resolvent R is satisfied by $\mathscr{A}$, and then of course $F \cup \{R\}$ is also satisfied.
2. $\mathscr{A} \models \neg L$. Because $\mathscr{A} \models C_1$ and $\mathscr{A} \not\models L$ it follows that $\mathscr{A} \models (C_1 - \{L\})$. As such, the resolvent R is satisfied by $\mathscr{A}$, and then of course $F \cup \{R\}$ is also satisfied.

$\square$

Resolution

Definition

Given a formula F in CNF form, we define $Res(F)$ as:

$$Res(F) = F \cup \{R \,|\, R \text{ is the Resolvent of two clauses in } F\}.$$

Furthermore, we define:

$$\begin{aligned} Res^0(F) &= F \\ Res^{t+1}(F) &= Res(Res^t(F)) \text{ for } t \geq 0 \\ Res^*(F) &= \textstyle\bigcup_{t \geq 0} Res^t(F) \end{aligned}$$

Resolution

Theorem (Resolutions Theorem)

A CNF formula F is unsatisfiable when $\Box \in Res^*(F)$.

Proof.

Assume $\Box \in Res^*(F)$. In this case it is enough to prove their that resolution is correct, and therefore, F is unsatisfiable. First, the empty clause can only be produced from two clauses of the form $C_1 = (L)$ and $C_2 = (\neg L)$. Since $\Box$ is contained in $Res^*(F)$, it must be the case that for some $t \geq 0$:

$$\Box, C_1, C_2 \in Res^{t+1}(F) \text{ and } C_1, C_2 \in Res^t(F)$$

Obviously there is no assignment to the literals that can solve both C_1 and C_2, and as such, $Res^t(F)$ is unsatisfiable. Furthermore, with the help of the Resolutions-Lemmas, you can argue that:

$$F \equiv Res^1(F) \equiv Res^2(F) \equiv \ldots \equiv Res^t(F) \equiv Res^{t+1}(F) \equiv \ldots$$

Which allows us to reason that the unsatisfiability of $Res^t(F)$ is equal to the unsatisfiability of F.

Now, all that is left is to show that resolution is complete for all possible CNF formulas. Using induction, it can be shown that for any unsatisfiable CNF formula F, we can recursively apply the resolution rule to arrive at the empty clause...

$\Box$

Resolution

Using the resolutions lemmas and proofs described earlier, it is now possible to construct a simple complete SAT solver.

- Given:
  - A CNF formula F
- Procedure:
  - Calculate $F = Res^0(F)$ for $t > 0$, and keep increasing t until the empty clause is produced, or there are no clauses left to resolve.
- Result:
  - In the case that $t > 0$: $\square \in Res^t(F) \Rightarrow F$ is unsatisfiable.
  - Or, in the case that $t > 0$: $\square \notin Res^t(F) = Res^{t+1}(F) \Rightarrow F$ is satisfiable.

Resolution

Complexity of this naive procedure:

- Since variables can only appear as positive Literals, negative Literals, or not at all in a clause, the run time of this algorithm for a formula with n Variables is in the worst case $O(3^n)$. In other words, with n variables there is a maximum of 3^n clauses that can be produced.

Example 4

- Is the following CNF formula F satisfiable?

  $F = (x_1, x_2) \wedge (x_1, \neg x_3) \wedge (\neg x_1, x_3) \wedge (\neg x_1, \neg x_2) \wedge (x_3, \neg x_2) \wedge (\neg x_3, x_2)$

- Using the procedure outlined earlier:

  $Res^0(F) = F$

  $Res^1(F) = Res^0(F) \cup \{(x_2, x_3), (x_1, x_3), (\neg x_2, \neg x_3), (x_1, \neg x_2), (\neg x_1, x_2), (\neg x_1, \neg x_3)\}$

  $Res^2(F) = Res^1(F) \cup \{\ldots, (x_1), \ldots, (\neg x_1), \ldots\}$

  $Res^3(F) = Res^2(F) \cup \{\square\}$

Resolution

Example 4

- Is the following CNF formula F satisfiable?

  $F = (x_1, x_2) \wedge (x_1, \neg x_3) \wedge (\neg x_1, x_3) \wedge (\neg x_1, \neg x_2) \wedge (x_3, \neg x_2) \wedge (\neg x_3, x_2)$

- Using the procedure outlined earlier:

  $Res^0(F) = F$

  $Res^1(F) = Res^0(F) \cup \{(x_2, x_3), (x_1, x_3), (\neg x_2, \neg x_3), (x_1, \neg x_2), (\neg x_1, x_2), (\neg x_1, \neg x_3)\}$

  $Res^2(F) = Res^1(F) \cup \{\ldots, (x_1), \ldots, (\neg x_1), \ldots\}$

  $Res^3(F) = Res^2(F) \cup \{\Box\}$

$\Rightarrow$ F is unsatisfiable!

Resolution

Example 5

- Is the following CNF formula F satisfiable?

  $F = (x_1, x_2, x_3) \wedge (x_2, \neg x_3, \neg x_4) \wedge (\neg x_2, x_5)$

- Using the procedure outlined earlier:

  $Res^0(F) = F$

  $Res^1(F) = Res^0(F) \cup \{(x_1, x_3, x_5), (\neg x_3, \neg x_4, x_5), (x_1, x_2, \neg x_4)\}$

  $Res^2(F) = Res^1(F) \cup \{(x_1, \neg x_4), (x_1, \neg x_4, x_5), (x_1, \neg x_4, x_2, x_5)\}$

  $Res^3(F) = Res^2(F) = Res^*(F)$

Resolution

Example 5

- Is the following CNF formula F satisfiable?

  $F = (x_1, x_2, x_3) \wedge (x_2, \neg x_3, \neg x_4) \wedge (\neg x_2, x_5)$

- Using the procedure outlined earlier:

  $Res^0(F) = F$

  $Res^1(F) = Res^0(F) \cup \{(x_1, x_3, x_5), (\neg x_3, \neg x_4, x_5), (x_1, x_2, \neg x_4)\}$

  $Res^2(F) = Res^1(F) \cup \{(x_1, \neg x_4), (x_1, \neg x_4, x_5), (x_1, \neg x_4, x_2, x_5)\}$

  $Res^3(F) = Res^2(F) = Res^*(F)$

$\Rightarrow$ F is satisfiable!

DP Algorithm

The SAT algorithm introduced by M. Davis und H. Putnam in 1960 was based on the previous procedures but included the following optimizations:

- Subsumption
- Pure Literal
- Variable Elimination

Note: these optimizations improve the run time of the solver, and can decrease the average complexity. However, the worst case complexity remains the same as the naive approach.

DP Algorithm

"The superiority of the present procedure over those previously available is indicated in part by the fact that a formula on which Gilmore's routine for the IBM 704[1] *causes the machine to compute for 21 minutes without obtaining a result was worked successfully by* ***hand computation*** *using the present method in 30 minutes."*

– M. Davis and H. Putnam

[1] The IBM 704 was one of the first commercial scientific computers. It had a maximum memory capacity of 4096×36 bit words (excluding magnetic tape storage), and could execute up to 40,000 instructions per second. Between 1955 and 1960, IBM sold over 120 of these machines.

DP Algorithm

Definition (Subsumption)

Assume we have two clauses C_1 and C_2. C_1 then subsumes C_2 when all the literals in C_1 are also in C_2: $C_1 \subseteq C_2$.

- Idea: To satisfy a CNF formula F, we must satisfy all the clauses. Therefore, if F is satisfiable, both C_1 and C_2. Since $C_1 \subseteq C_2$, every satisfying assignment for C_1 will automatically solve C_2. This means that we can delete C_2 from F without changing the satisfiability of the formula.
- The idea of subsumption is used in all modern SAT solvers. Most only perform subsumption checks during preprocessing, but other do it continually.

Re-examine Example 5

- Is the following CNF formula F satisfiable?

  $F = (x_1, x_2, x_3) \wedge (x_2, \neg x_3, \neg x_4) \wedge (\neg x_2, x_5)$

- Using the naive procedure outlined earlier:

  $Res^0(F) = F$

  $Res^1(F) = Res^0(F) \cup \{(x_1, x_3, x_5), (\neg x_3, \neg x_4, x_5), (x_1, x_2, \neg x_4)\}$

  $Res^2(F) = Res^1(F) \cup \{(x_1, \neg x_4), \underbrace{(x_1, \neg x_4, x_5), (x_1, \neg x_4, x_2, x_5)}_{\text{subsummed from } (x_1, \neg x_4)}\}$

  $Res^3(F) = Res^2(F) = Res^*(F)$

$\Rightarrow$ F is satisfiable!

DP Algorithm

Definition (Pure Literal)

Let F be a CNF formula and L a literal contained in F. We say L is pure literal iff it is only present in its positive or negative form in F. In other words, F contains L or $\neg L$, but not both.

- Idea: Remove all the clauses from F that contain the pure literal L. This can be done because L will only satisfy clauses if assigned correctly. $\neg L$ on the other hand will only "unsatisfy" clauses.
- Normally, this step is only used during the preprocessing of a CNF formula for a SAT solver, and is no longer used during the solving process. However, on harder QBF problems, this technique is still used.

DP Algorithm

Again, repeating example 5

- Is the following CNF formula F satisfiable?

  $F = (x_1, x_2, x_3) \wedge (x_2, \neg x_3, \neg x_4) \wedge (\neg x_2, x_5)$

$\Rightarrow$ x_1, $\neg x_4$ und x_5 are pure literals.

$\Rightarrow$ Delete clauses containing x_1, $\neg x_4$ or x_5.

$\Rightarrow$ $F = \{\}$

$\Rightarrow$ F is satisfiable!

DP Algorithm

Within the DP algorithm resolution is used to completely remove a variable x_i from the formula (i.e. delete all positive and negative occurances of x_i from a CNF formula F). $\Rightarrow$ Variable Elimination

Goal: Reduce the number of variables occuring in the CNF formula F, while maintaining a relatively constant number of clauses.

DP Algorithm

Definition

Let F be a CNF formula, and x_i the variable we wish to eliminate (where $L = x_i$ and $\neg L = \neg x_i$). Then we need to define P, N and W as follows:

- Let P be the set of all clauses in F that include L:

$$P = \{C \mid (L \in C) \wedge (C \in F)\}$$

- Let N be the set of all clauses in F that include $\neg L$:

$$N = \{C \mid (\neg L \in C) \wedge (C \in F)\}$$

- Let W be the set of all clauses in F that do not contain L or $\neg L$:

$$W = \{C \mid (L \notin C) \wedge (\neg L \notin C) \wedge (C \in F)\}$$

As such: $F = P \wedge N \wedge W$.

DP Algorithm

Definition

Given the clause partitioning mentioned previously, $P \otimes_{x_i} N$ defines the set of clauses that are generated through the pairwise resolution on variable x_i from all combinations of clauses from P and N:

$$P \otimes_{x_i} N = \{R \mid (R = C_1 \otimes_{x_i} C_2) \wedge (C_1 \in P) \wedge (C_2 \in N)\}$$

DP Algorithm

Theorem

Let F be a CNF formula, and let x_i be a Variable. Assume the possitve occurance ($L = x_i$) and the negative occurance $\neg L = \neg x_i$ of the variable x_i appear in the formula F. Futhermore, let the clause sets P, N and W be used as defined earlier. Then, it must be the case that $F = P \wedge N \wedge W$ and $F' = (P \otimes_{x_i} N) \wedge W$ are equally satisfiable.

DP Algorithm

Conclusion of the previous sentence:

The question of the satisfiability of a formula F can be attributed to the satisfiability of F', where F' is the constructed from F through the elimination of the variable x_i. As such, if F' is unsatisfiable, so is F. Otherwise both are satisfiable.

DP Algorithm

Basic procedure for variable elimination:

- Select a variable x_i, then perform resolution between all the pairs of clauses containing x_i and $\neg x_i$. Then replace all the clauses in the sets P and N with the new clauses generated during resolution.
- If done blindly, the amount of new clauses that are produced normally is much greater than the sum of P and N. As such, the total number of clauses in the formula usually increases.
- Variable elimination is done in modern SAT solvers during preprocessing. Furthermore, using heuristics, variables are only selected for removal when they results in the total number of clauses remaining the same or decreasing.

DP Algorithm

Albert-Ludwigs-Universität Freiburg

Example 6

- Is the following CNF formula F satisfiable?

  $F = (x_1, x_2) \wedge (x_1, \neg x_3) \wedge (\neg x_1, x_3) \wedge (\neg x_1, \neg x_2) \wedge (x_3, \neg x_2) \wedge (\neg x_3, x_2)$

- Elimination of x_1 from formula F:

$$\begin{aligned}
P &= \{(x_1, x_2), (x_1, \neg x_3)\} \\
N &= \{(\neg x_1, x_3), (\neg x_1, \neg x_2)\} \\
W &= \{(x_3, \neg x_2), (\neg x_3, x_2)\} \\
P \otimes_{x_1} N &= \{(x_2, x_3), (x_2, \neg x_2), (\neg x_3, x_3), (\neg x_3, \neg x_2)\} \\
F' &= (P \otimes_{x_1} N) \wedge W = (x_2, x_3) \wedge (\neg x_3, \neg x_2) \wedge (x_3, \neg x_2) \wedge (\neg x_3, x_2)
\end{aligned}$$

- Elimination of x_2 from formula F:

$$\begin{aligned}
P' &= \{(x_2, x_3), (\neg x_3, x_2)\} \\
N' &= \{(\neg x_3, \neg x_2), (x_3, \neg x_2)\} \\
W' &= \emptyset \\
P' \otimes_{x_2} N' &= \{(x_3, \neg x_3), (x_3), (\neg x_3), (\neg x_3, x_3)\} \\
F'' &= (P' \otimes_{x_2} N') \wedge W' = (x_3) \wedge (\neg x_3)
\end{aligned}$$

Example 6

- Is the following CNF formula F satisfiable?

  $F = (x_1, x_2) \wedge (x_1, \neg x_3) \wedge (\neg x_1, x_3) \wedge (\neg x_1, \neg x_2) \wedge (x_3, \neg x_2) \wedge (\neg x_3, x_2)$

- Elimination of x_1 from formula F:

$$\begin{aligned}
P &= \{(x_1, x_2), (x_1, \neg x_3)\} \\
N &= \{(\neg x_1, x_3), (\neg x_1, \neg x_2)\} \\
W &= \{(x_3, \neg x_2), (\neg x_3, x_2)\} \\
P \otimes_{x_1} N &= \{(x_2, x_3), (x_2, \neg x_2), (\neg x_3, x_3), (\neg x_3, \neg x_2)\} \\
F' &= (P \otimes_{x_1} N) \wedge W = (x_2, x_3) \wedge (\neg x_3, \neg x_2) \wedge (x_3, \neg x_2) \wedge (\neg x_3, x_2)
\end{aligned}$$

- Elimination of x_2 from formula F:

$$\begin{aligned}
P' &= \{(x_2, x_3), (\neg x_3, x_2)\} \\
N' &= \{(\neg x_3, \neg x_2), (x_3, \neg x_2)\} \\
W' &= \emptyset \\
P' \otimes_{x_2} N' &= \{(x_3, \neg x_3), (x_3), (\neg x_3), (\neg x_3, x_3)\} \\
F'' &= (P' \otimes_{x_2} N') \wedge W' = (x_3) \wedge (\neg x_3) \Rightarrow F'' \text{ and } F \text{ are unsatisfiable!}
\end{aligned}$$

Davis-Putnam Algorithm

```
bool DP(CNF $F$)
{
   if ($F = \emptyset$) { return SATISFIABLE; }                       // Empty clause set.
   if ($\square \in F$) { return UNSATISFIABLE; }                     // Empty clause.
   if ($F$ contains a unit clause $(L)$)                              // Unit clause rule.
      {
         // Unit Subsumption.
         $F' = F - \{C \mid (L \in C) \wedge (C \in F) \wedge (C \neq (L))\}$;
         // Unit Resolution.
         $P = \{(L)\}$;
         $N = \{C \mid (\neg L \in C) \wedge (C \in F')\}$;
         $W = F' - P - N$;
         return DP($[P \otimes_L N] \wedge W$);
      }
   if ($F$ contains a pure literal $L$)                               // Pure literal rule.
      {
         // Delete from $F$ every clause containing $L$.
         $F' = F - \{C \mid (L \in C) \wedge (C \in F)\}$;
         return DP($F'$);
      }
   $L$ = SELECTLITERAL($F$);                                          // Select a literal.
   $P = \{C \mid (L \in C) \wedge (C \in F)$;                         // Variablen elimination.
   $N = \{C \mid (\neg L \in C) \wedge (C \in F)\}$;
   $W = F - P - N$;
   return DP($[P \otimes_L N] \wedge W$);
}
```

DLL Algorithm

- Due to the possibly exponential growth in memory requirements, the basic DP Algorithm has only seen little use.
- However, two years later in 1962, M. Davis, G. Logemann und D. Loveland introduced the DLL-Algorithm, which replaced the physical variable elimination with a depth first search.
- Idea: If a CNF formula F is satisfiable, a satisfying assignment of the variables in F must included either $x_i = 1$ oder $x_i = 0 \Rightarrow$ Check both paths one after another.
- In literature, the DLL algorithm is often referred to as the DPLL algorithm.

Davis-Logemann-Loveland Algorithm

```
bool DLL(CNF $F$)
{
   if ($F = \emptyset$) { return SATISFIABLE; }                          // Empty clause set.
   if ($\square \in F$) { return UNSATISFIABLE; }                        // Empty clause.
   if ($F$ contains a unit clause $(L)$)                                 // Unit clause rule.
      {
         // Unit Subsumption.
         $F' = F - \{C \mid (L \in C) \wedge (C \in F) \wedge (C \neq (L))\}$;
         // Unit Resolution.
         $P = \{(L)\}$;
         $N = \{C \mid (\neg L \in C) \wedge (C \in F')\}$;
         $W = F' - P - N$;
         return DLL($[P \otimes_L N] \wedge W$);
      }
   if ($F$ contains a pure literal $L$)                                  // Pure literal rule.
      {
         // Delete from $F$ every clause containing $L$.
         $F' = F - \{C \mid (L \in C) \wedge (C \in F)\}$;
         return DLL($F'$);
      }
   $L$ = SELECTLITERAL($F$);                                             // Select a literal.
   if (DLL($F \cup \{(L)\}$) $==$ SATISFIABLE)                           // Path selection.
      { return SATISFIABLE; }
   else
      { return DLL($F \cup \{(\neg L)\}$); }
}
```

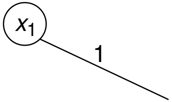
DLL Algorithm

Albert-Ludwigs-Universität Freiburg

$(\neg x_1, \neg x_2, \neg x_3) \wedge (\neg x_1, \neg x_2, x_3) \wedge (\neg x_1, x_2, \neg x_3) \wedge (\neg x_1, x_2, x_3) \wedge (x_1, \neg x_2, \neg x_3)$

DLL Algorithm

Albert-Ludwigs-Universität Freiburg

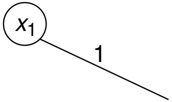


$(\neg x_1, \neg x_2, \neg x_3) \wedge (\neg x_1, \neg x_2, x_3) \wedge (\neg x_1, x_2, \neg x_3) \wedge (\neg x_1, x_2, x_3) \wedge (x_1, \neg x_2, \neg x_3)$

Select path $x_1 = 1$

DLL Algorithm

Albert-Ludwigs-Universität Freiburg

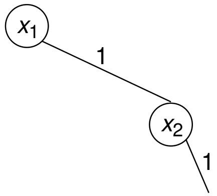


$(\quad, \neg x_2, \neg x_3) \wedge (\quad, \neg x_2, x_3) \wedge (\quad, x_2, \neg x_3) \wedge (\quad, x_2, x_3) \wedge$

Select path $x_1 = 1$

DLL Algorithm

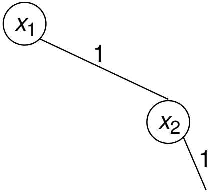
Albert-Ludwigs-Universität Freiburg



$(\quad , \neg x_2, \neg x_3) \wedge (\quad , \neg x_2, x_3) \wedge (\quad , x_2, \neg x_3) \wedge (\quad , x_2, x_3) \wedge$

Select path $x_2 = 1$

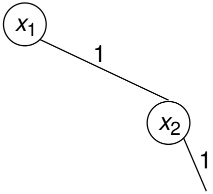
DLL Algorithm



$(\quad , \quad , \neg x_3) \wedge (\quad , \quad , x_3) \wedge \qquad \wedge \qquad \wedge$

Select path $x_2 = 1$

DLL Algorithm

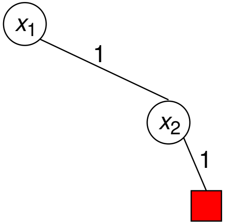


$(\quad , \quad , \neg x_3) \wedge (\quad , \quad , x_3) \wedge \qquad \wedge \qquad \wedge$

Unit clause rule $x_3 = 0$ and $x_3 = 1$

DLL Algorithm

Albert-Ludwigs-Universität Freiburg

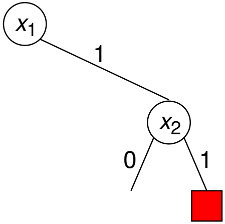


$(\quad , \quad , \neg x_3) \wedge (\quad , \quad , x_3) \wedge \qquad \wedge \qquad \wedge$

Try opposite path

DLL Algorithm

Albert-Ludwigs-Universität Freiburg

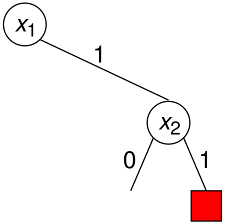


$(\quad , \neg x_2, \neg x_3) \wedge (\quad , \neg x_2, x_3) \wedge (\quad , x_2, \neg x_3) \wedge (\quad , x_2, x_3) \wedge$

Select path $x_2 = 0$

DLL Algorithm

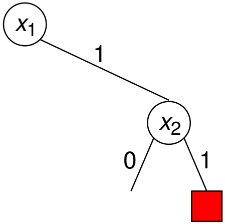
Albert-Ludwigs-Universität Freiburg



$\wedge \qquad\qquad \wedge (\quad , \quad , \neg x_3) \wedge (\quad , \quad , x_3) \wedge$

Select path $x_2 = 0$

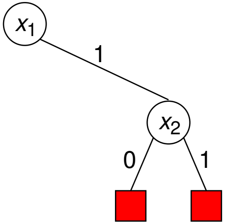
DLL Algorithm



$\wedge \qquad \wedge (\quad , \quad , \neg x_3) \wedge (\quad , \quad , x_3) \wedge$

Unit clause rule $x_3 = 0$ and $x_3 = 1$

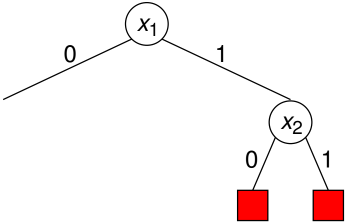
DLL Algorithm



$\wedge \qquad \wedge(\quad,\quad,\neg x_3)\wedge(\quad,\quad,x_3)\wedge$

Try opposite path

DLL Algorithm

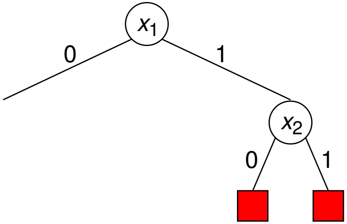


$(\neg x_1, \neg x_2, \neg x_3) \wedge (\neg x_1, \neg x_2, x_3) \wedge (\neg x_1, x_2, \neg x_3) \wedge (\neg x_1, x_2, x_3) \wedge (x_1, \neg x_2, \neg x_3)$

Select path $x_1 = 0$

DLL Algorithm

Albert-Ludwigs-Universität Freiburg

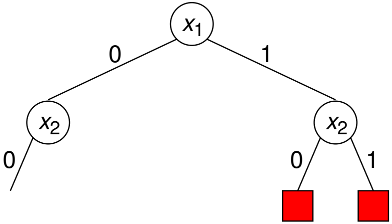


$\wedge \qquad \wedge \qquad \wedge \qquad \wedge(\quad, \neg x_2, \neg x_3)$

Select path $x_1 = 0$

DLL Algorithm

Albert-Ludwigs-Universität Freiburg

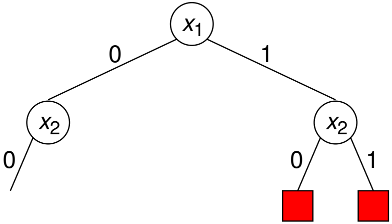


$\wedge \quad \wedge \quad \wedge \quad \wedge (\quad , \neg x_2, \neg x_3)$

Pure literal rule $x_2 = 0$

DLL Algorithm

Albert-Ludwigs-Universität Freiburg

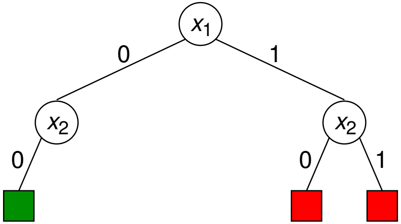


$\wedge$ $\wedge$ $\wedge$ $\wedge$

Pure literal rule $x_2 = 0$

DLL Algorithm

Albert-Ludwigs-Universität Freiburg



$\wedge$ $\wedge$ $\wedge$ $\wedge$

Formula is satisfiable with $x_1 = x_2 = 0$

Whats next?

- Data structures and algorithmic implementation details have so far been largely omitted.

⇒ Modern SAT algorithms

  - Preprocessing
  - Decision heuristics and strategies
  - Boolean constraint propagation
  - Conflict analysis & non-chronological backtracking
  - Conflict clause deletion
  - Restarts

Summary: DLL-Algorithm

It is a recursive procedure where at every recursive step the following are checked:

1. Termination:
   - Empty clause set $\Rightarrow$ formula is satisfiable
   - Empty clause $\Rightarrow$ Current (partial) formula unsatisfiable
2. Unit clause / pure literal rule
   - In the case that there is the unit clause (L) in the current sub-formula F, we can simplify F to F' by deleting all clauses containing L, and by removing all occurrences of $\neg L$ in the formula.
   - In the case that the sub-formula F contains a pure literal, we can delete the occurrence of these variables from all clauses in F, and in the process, produce F'.
3. Search both branches
   - For the current formula F which still contains the literal L, we must call the DLL algorithm recursively on for both $F \cup \{(L)\}$ and $F \cup \{(\neg L)\}$.

From the DLL to modern SAT Algorithms

Approach:

- DLL Algorithm
  - Recursive procedure
  - From one recursive level r to $r+1$ the algorithm modifies the formula (satisfied clauses are deleted, and falsified literals are removed from clauses).
  - When jumping back from recursive level $r+1$ to r, the algorithm has to re-insert all deleted clauses and literal occurrences in the formula.

From the DLL to modern SAT Algorithms

Approach:

- Modern SAT Algorithms
  - No longer a recursive procedure
  - Except for special cases, clauses and literals are not physically removed from the CNF formula during the search process.
  - In general, the pure literal rule is no longer used (QBF?)

From the DLL to modern SAT Algorithms

Unit Clause

- DLL Algorithm
    - A clause which contains exactly one literal.

Unit Clause

- Modern SAT Algorithms
  - In addition to the previous definition, a clauses that contains only falsified literals and one unassigned literal under the current search space evaluation is also a unit clause.
  - Example: The assignments $x_1 = 0, x_2 = 1$ turns $(x_1, \neg x_2, x_3)$ into a unit clause.
  - In the example, by adding the assignment $x_3 = 1$ to the previous assignments $x_1 = 0, x_2 = 1$, the clause $(x_1, \neg x_2, x_3)$ becomes satisfied. This use of the unit clause rule in this case implies $x_3 = 1$. As such $x_3 = 1$ is referred to as an Implication.

From the DLL to modern SAT Algorithms

Unit Clause

- Modern SAT Algorithms
    - …
    - Determining all the implications (i.e. unit propagations) of assigning a variable a value is in modern solvers done by the Boolean Constraint Propagation (BCP) procedure. In its original form, the DLL would recursively call itself after every implication.
    - Example: In $F = (x_1, \neg x_2) \wedge (x_1, x_2, x_3) \wedge (\neg x_3, x_4)$ the assignment $x_1 = 0$ results in the implications $x_2 = 0, x_3 = 1, x_4 = 1$

From the DLL to modern SAT Algorithms

Unsatisfiable paths / Conflicts

- DLL Algorithm:
  - Empty clause.
- Modern SAT Algorithm:
  - A clause where all its literals are falsely assigned.
  - Example: The assignments $x_1 = 0, x_2 = 1, x_3 = 0$ result in the clause $(x_1, \neg x_2, x_3)$ becoming falsified. Since all our formulas are in CNF form, the entire formulas under this assignment is also unsatisfied.

From the DLL to modern SAT Algorithms

Unsatisfiable paths / Conflicts

- DLL Algorithm
  - A conflict is always the result of the previous variable selection, and the resulting unit implications.
  - Backtracking, to a previous recursion level, in which both possible cases of a variable have not been checked, allows the solver to remove the existing conflict.
  - In the case that their exist no unchecked path, and the solver must recursive backtrack to its first call DLL call, the CNF formula is unsatisfiable.

From the DLL to modern SAT Algorithms

Unsatisfiable paths / Conflicts

- Modern SAT Algorithms
    - Current solvers perform a more indepth analysis of every conflict as it is often the case that multiple variable selections play a role in each conflict.
    - Generation (by resolution) and addition of new conflict clauses to the formula allow the solver to learn important information about the problem. These conflict clauses contain a list of literals that are responsible for the current conflict.
    - With the use of the conflict claues, the algorithm can in many cases backtrack past multiple variable selections. The procedure can also produce the the empty or null clause resulting in a final evaluation of *UNSATISFIABLE*.

Modern SAT Algorithms

Basic procedures of a moderns SAT-Solver

- Preprocessing
- Main routines:…
  - Selection of decision variables
  - Boolean constraint propagation / unit propagation
  - Conflict analysis & backtracking
- Ever now and then during the search:
  - Reduce size of conflict clause set (delete clauses)
  - Restarts
- If the formula is satisfiable:
  - Output a variable assignment that satisfies all the clauses (i.e. a model)

Main procedure of a modern sequential SAT algorithm

```
bool SequentialSatEngine(CNF F)
{
  if (PreprocessCNF(F) == CONFLICT)                          // Simplify the CNF formula.
    { return UNSATISFIABLE; }                                // Problem is unsatisfiable.
  while (true)
    {
      if (DecideNextBranch())                      // Select a free variable and assign it a value.
        {
          while (BCP() == CONFLICT)                          // Boolean constraint propagation.
            {
              BLevel = AnalyzeConflict();                    // Conflict analysis.
              if (BLevel > 0)
                { Backtrack(BLevel); }                       // Backtrack to a previous decision.
              else
                { return UNSATISFIABLE; }                    // Problem unsatisfiable.
            }
        }
      else
        { return SATISFIABLE; }                    // All variables are assigned, problem satisfiable.
    }
}
```

Not explicitly shown: Deletion of conflict clauses, restarts, or outputted model.

Main procedure of a modern sequential SAT algorithm

```
bool SequentialSatEngine(CNF F)
{
  if (PreprocessCNF(F) == CONFLICT)                          // Simplify the CNF formula.
    { return UNSATISFIABLE; }                                // Problem is unsatisfiable.
  while (true)
    {
      if (DecideNextBranch())                                // Select a free variable and assign it a value.
        {
          while (BCP() == CONFLICT)                          // Boolean constraint propagation.
            {
              BLevel = AnalyzeConflict();                    // Conflict analysis.
              if (BLevel > 0)
                { Backtrack(BLevel); }                       // Backtrack to a previous decision.
              else
                { return UNSATISFIABLE; }                    // Problem unsatisfiable.
            }
        }
      else
        { return SATISFIABLE; }                              // All variables are assigned, problem satisfiable.
    }
}
```

Not explicitly shown: Deletion of conflict clauses, restarts, or outputted model.

Preprocessing

- Goal:
  - Prior to actually starting the search, try to simplify the formula as much as possible.
- Practical observations:
  - In many cases, the size of the input formula directly correlates to the run time of the SAT Algorithm.
  - A reduction of more than 75% in the number of clauses & variables in the input formula can be achieved.
- Identification and processing of unit clauses that are contained with the original clause set has always been a part of a modern SAT algorithm.
- The trick is to find a balance between: the simplification that preprocessing is able to achieve;the time required by the preprocessor; and the performance increases gained by the SAT search algorithm.

Preprocessing

Unit Propagation Lookahead (UPLA)

- For a variable x_i test $x_i = 0$ and $x_i = 1$. During the test monitor what each assignment leads to:
  - $(x_i = 0 \rightarrow \mathit{conflict}) \wedge (x_i = 1 \rightarrow \mathit{conflict}) \Rightarrow \mathit{UNSAT}$
  - $(x_i = 0 \rightarrow \mathit{conflict}) \Rightarrow x_i = 1$
  - $(x_i = 1 \rightarrow \mathit{conflict}) \Rightarrow x_i = 0$
  - $(x_i = 0 \rightarrow x_j = 1) \wedge (x_i = 1 \rightarrow x_j = 1) \Rightarrow x_j = 1$
  - $(x_i = 0 \rightarrow x_j = 0) \wedge (x_i = 1 \rightarrow x_j = 0) \Rightarrow x_j = 0$
  - $(x_i = 0 \rightarrow x_j = 0) \wedge (x_i = 1 \rightarrow x_j = 1) \Rightarrow x_i \equiv x_j$

Unit Propagation Lookahead (UPLA)

- Advantages
  - Uses procedures that are already implemented in most SAT solvers.
- Disadvantages
  - Formula needs to have binary clauses.
  - Model extraction is can be more complicated (e.g. when $x_i \equiv x_j$ is found, and all x_i's are replaced with x_j's).
  - Can be time consuming if ALL variables are tested.

Preprocessing

Applying the resolution rule:

- Advantages:
  - Can be performed on any formula in CNF form.
  - Possible to achieve far-reaching simplifications in reasonable time.
- Disadvantages:
  - Model expansion necessary.
- Techniques (SatELite)
  - Self-Subsuming Resolution
  - Elimination by Clause Distribution
  - Variable Elimination by Substitution
  - Forward Subsumption
  - Backward Subsumption

Preprocessing

Self-Subsuming Resolution

- Given formula:
  - $F = (x_1 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge \ldots$
- Applying resolution to the first two clauses yields:
  - $(x_1 \vee \neg x_3) \otimes_{x_3} (x_1 \vee x_2 \vee x_3) = (x_1 \vee x_2)$

  $\Rightarrow$ $(x_1 \vee x_2)$ subsumes $(x_1 \vee x_2 \vee x_3)$

  $\Rightarrow$ Replace $(x_1 \vee x_2 \vee x_3)$ with $(x_1 \vee x_2)$
- Simplified formula:
  - $F' = (x_1 \vee \neg x_3) \wedge (x_1 \vee x_2) \wedge \ldots$
- Savings:
  - 1 literal

Preprocessing

Elimination by Clause Distribution

- Referred to as variable elimination earlier.
- Given formula:
  - $F = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_2)$
- Performing variablen elimination on x_1 leads to:
  - $F' = (x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_2)$
- Savings:
  - 1 variable, 2 clauses, and 4 literals.
- Only used if it actually simplifies the formula. A modern SAT solver makes many checks before actually eliminating a variable.

Preprocessing

Variable Elimination by Substitution

- Given formula:
  - $F = (\neg x_5 \vee x_1) \wedge (\neg x_5 \vee x_2) \wedge (x_5 \vee \neg x_1 \vee \neg x_2) \wedge$ $(x_4 \vee \neg x_5) \wedge (\neg x_4 \vee x_5 \vee x_6)$
- The first three clauses represent a logical AND gate:
  - $[(\neg x_5 \vee x_1) \wedge (\neg x_5 \vee x_2) \wedge (x_5 \vee \neg x_1 \vee \neg x_2)] \leftrightarrow [x_5 \equiv x_1 \wedge x_2]$
- Delete the first three clauses by substituting the variable x_5 with $x_1 \wedge x_2$ in the remaining clauses:
  - $F' = (x_4 \vee \neg(x_1 \wedge x_2)) \wedge (\neg x_4 \vee (x_1 \wedge x_2) \vee x_6)$
- Restoring the CNF representation leads to:
  - $F'' = (x_4 \vee \neg x_1 \vee \neg x_2) \wedge (\neg x_4 \vee x_1 \vee x_6) \wedge (\neg x_4 \vee x_2 \vee x_6)$
- Savings: 1 variable, 2 clauses, and 3 literals.
- Again, only used if it actually simplifies the formula.
- For other gates we can do similar things.

Preprocessing

Albert-Ludwigs-Universität Freiburg

Forward Subsumption

- Test to see if any of the newly generated clauses from the preprocessing steps are already subsumed by existing clauses in our current clause set.

Backward Subsumption

- Test to see if the newly generated clauses from the preprocessing steps subsumes any existing clause from the current clause set.

$\Rightarrow$ Delete all subsumed clauses.

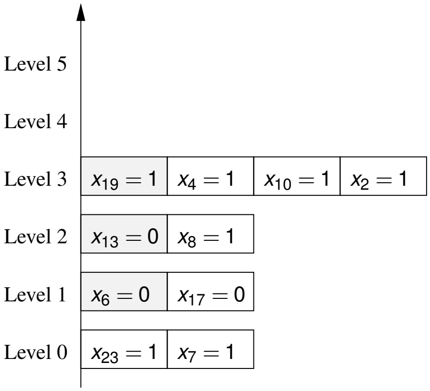
Main procedure of a modern sequential SAT algorithm

```
bool SequentialSatEngine(CNF F)
{
  if (PreprocessCNF(F) == CONFLICT)                                  // Simplify the CNF formula.
    { return UNSATISFIABLE; }                                        // Problem is unsatisfiable.
  while (true)
    {
      if (DecideNextBranch())                    // Select a free variable and assign it a value.
        {
          while (BCP() == CONFLICT)                           // Boolean Constraint Propagation.
            {
              BLevel = AnalyzeConflict();                                  // Conflict analysis.
              if (BLevel > 0)
                { Backtrack(BLevel); }                      // Backtrack to a previous decision.
              else
                { return UNSATISFIABLE; }                              // Problem unsatisfiable.
            }
        }
      else
        { return SATISFIABLE; }               // All variables are assigned, problem satisfiable.
    }
}
```

Not explicitly shown: Deletion of conflict clauses, restarts, or outputted model.

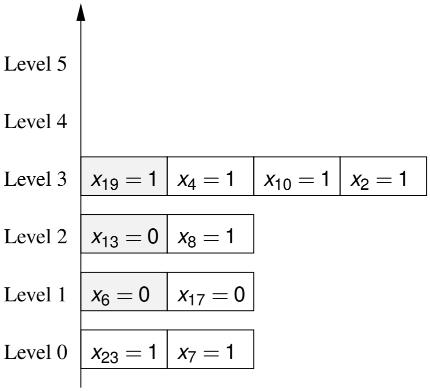
Decision Stack

Albert-Ludwigs-Universität Freiburg



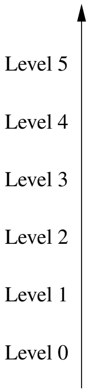
- Central data structure of modern SAT solvers.
- Decision stack saves the order of the assignments and implications.
- In a CNF formula is satisfiable, the decision stack stores the model (i.e. the satisfying assignment).

Decision Stack



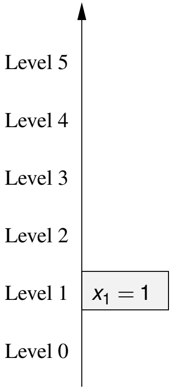
- Each variable assignment is associated with a decision level
- The decision level variables starts at 0, and for every decision variable it is incremented. For backtracking, it is decremented by one for every decision the solver backtracks past.
- Decision level 0 is important, as it stores all implications that directly result from unit claues (i.e. does not contain decision variables).
- A conflict on decision level 0 means that the entire CNF formula is unsatisfiable.

Decision Stack



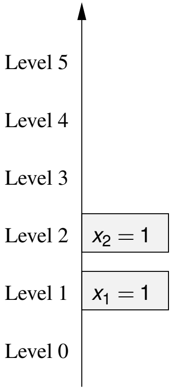
$(\neg x_1, \neg x_2, \neg x_3) \wedge (\neg x_1, \neg x_2, x_3) \wedge (\neg x_1, x_2, \neg x_3) \wedge (\neg x_1, x_2, x_3) \wedge (x_1, \neg x_2, \neg x_3)$

Decision Stack



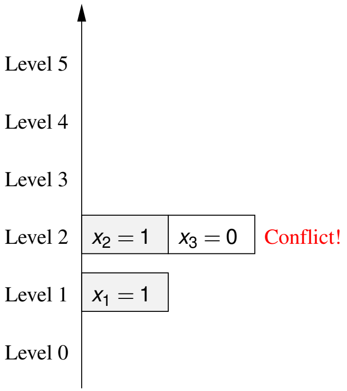
$(\neg x_1, \neg x_2, \neg x_3) \wedge (\neg x_1, \neg x_2, x_3) \wedge (\neg x_1, x_2, \neg x_3) \wedge (\neg x_1, x_2, x_3) \wedge (x_1, \neg x_2, \neg x_3)$

Decision Stack



$(\neg x_1, \neg x_2, \neg x_3) \land (\neg x_1, \neg x_2, x_3) \land (\neg x_1, x_2, \neg x_3) \land (\neg x_1, x_2, x_3) \land (x_1, \neg x_2, \neg x_3)$

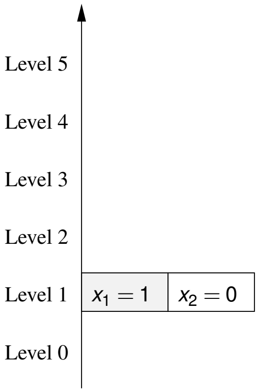
Decision Stack



$(\neg x_1, \neg x_2, \neg x_3) \land (\neg x_1, \neg x_2, x_3) \land (\neg x_1, x_2, \neg x_3) \land (\neg x_1, x_2, x_3) \land (x_1, \neg x_2, \neg x_3)$

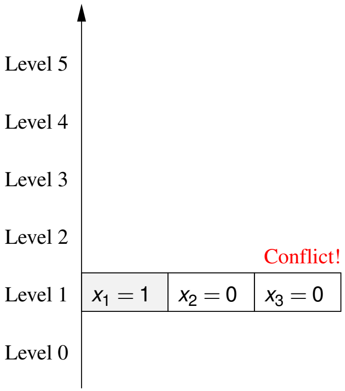
Decision Stack

Albert-Ludwigs-Universität Freiburg



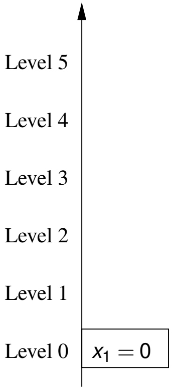
$(\neg x_1, \neg x_2, \neg x_3) \wedge (\neg x_1, \neg x_2, x_3) \wedge (\neg x_1, x_2, \neg x_3) \wedge (\neg x_1, x_2, x_3) \wedge (x_1, \neg x_2, \neg x_3)$

Decision Stack



$(\neg x_1, \neg x_2, \neg x_3) \wedge (\neg x_1, \neg x_2, x_3) \wedge (\neg x_1, x_2, \neg x_3) \wedge (\neg x_1, x_2, x_3) \wedge (x_1, \neg x_2, \neg x_3)$

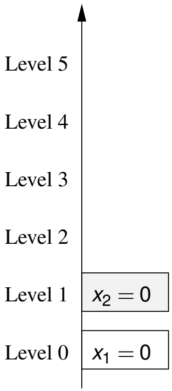
Decision Stack



$(\neg x_1, \neg x_2, \neg x_3) \wedge (\neg x_1, \neg x_2, x_3) \wedge (\neg x_1, x_2, \neg x_3) \wedge (\neg x_1, x_2, x_3) \wedge (x_1, \neg x_2, \neg x_3)$

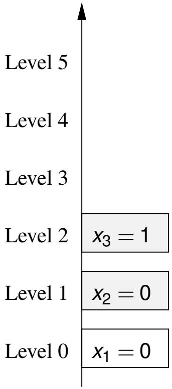
Decision Stack

Albert-Ludwigs-Universität Freiburg



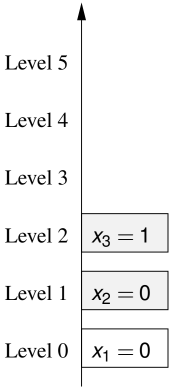
$(\neg x_1, \neg x_2, \neg x_3) \wedge (\neg x_1, \neg x_2, x_3) \wedge (\neg x_1, x_2, \neg x_3) \wedge (\neg x_1, x_2, x_3) \wedge (x_1, \neg x_2, \neg x_3)$

Decision Stack



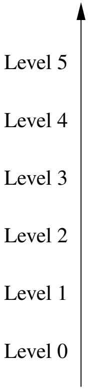
$(\neg x_1, \neg x_2, \neg x_3) \wedge (\neg x_1, \neg x_2, x_3) \wedge (\neg x_1, x_2, \neg x_3) \wedge (\neg x_1, x_2, x_3) \wedge (x_1, \neg x_2, \neg x_3)$

Decision Stack



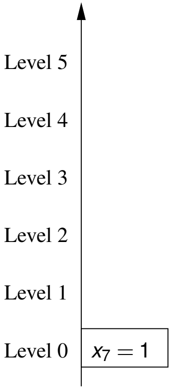
⇒ Formula is satisfiable with assignments: $x_1 = 0, x_2 = 0, x_3 = 1$.

Decision Stack



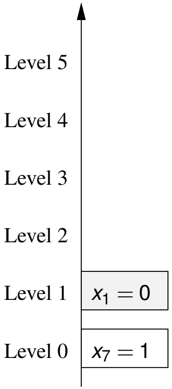
$$(x_1, x_2) \wedge (x_1, \neg x_3) \wedge (\neg x_1, x_3) \wedge (\neg x_1, \neg x_2) \wedge (x_3, \neg x_2) \wedge (\neg x_3, x_2) \wedge (x_7)$$

Decision Stack



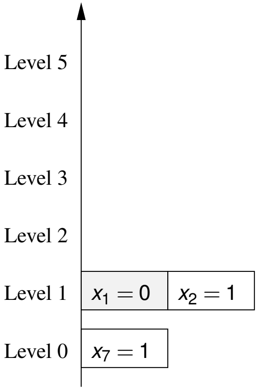
$$(x_1, x_2) \wedge (x_1, \neg x_3) \wedge (\neg x_1, x_3) \wedge (\neg x_1, \neg x_2) \wedge (x_3, \neg x_2) \wedge (\neg x_3, x_2) \wedge (x_7)$$

Decision Stack



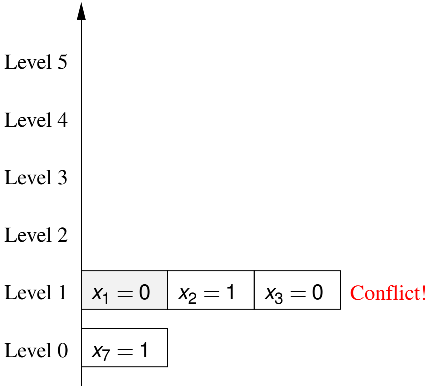
$$(x_1, x_2) \wedge (x_1, \neg x_3) \wedge (\neg x_1, x_3) \wedge (\neg x_1, \neg x_2) \wedge (x_3, \neg x_2) \wedge (\neg x_3, x_2) \wedge (x_7)$$

Decision Stack



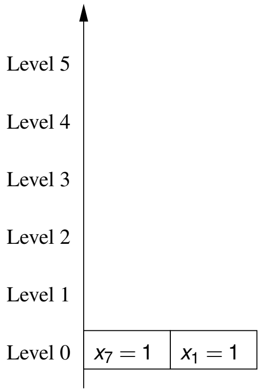
$(x_1, x_2) \wedge (x_1, \neg x_3) \wedge (\neg x_1, x_3) \wedge (\neg x_1, \neg x_2) \wedge (x_3, \neg x_2) \wedge (\neg x_3, x_2) \wedge (x_7)$

Decision Stack



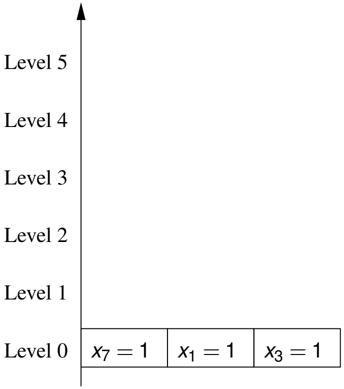
$(x_1, x_2) \wedge (x_1, \neg x_3) \wedge (\neg x_1, x_3) \wedge (\neg x_1, \neg x_2) \wedge (x_3, \neg x_2) \wedge (\neg x_3, x_2) \wedge (x_7)$

Decision Stack



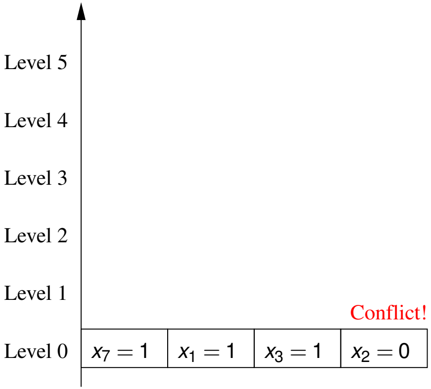
$$(x_1, x_2) \land (x_1, \neg x_3) \land (\neg x_1, x_3) \land (\neg x_1, \neg x_2) \land (x_3, \neg x_2) \land (\neg x_3, x_2) \land (x_7)$$

Decision Stack



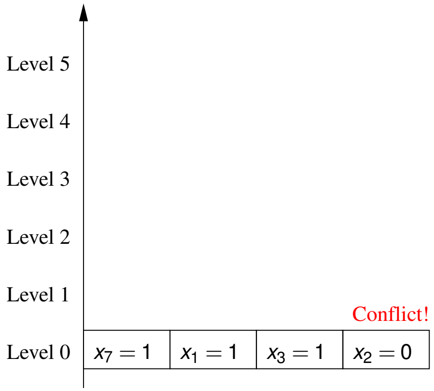
$$(x_1, x_2) \wedge (x_1, \neg x_3) \wedge (\neg x_1, x_3) \wedge (\neg x_1, \neg x_2) \wedge (x_3, \neg x_2) \wedge (\neg x_3, x_2) \wedge (x_7)$$

Decision Stack



$$(x_1, x_2) \wedge (x_1, \neg x_3) \wedge (\neg x_1, x_3) \wedge (\neg x_1, \neg x_2) \wedge (x_3, \neg x_2) \wedge (\neg x_3, x_2) \wedge (x_7)$$

Decision Stack



$\Rightarrow$ Formula is unsatisfiable as there is a conflict on decision level 0.

Main procedure of a modern sequential SAT algorithm

```
bool SequentialSatEngine(CNF F)
{
   if (PreprocessCNF(F) == CONFLICT)                       // Simplify the CNF formula.
      { return UNSATISFIABLE; }                            // Problem is unsatisfiable.
   while (true)
      {
         if (DecideNextBranch())                 // Select a free variable and assign it a value.
            {
               while (BCP() == CONFLICT)                   // Boolean Constraint Propagation.
                  {
                     BLevel = AnalyzeConflict();                         // Conflict analysis.
                     if (BLevel > 0)
                        { Backtrack(BLevel); }             // Backtrack to a previous decision.
                     else
                        { return UNSATISFIABLE; }                   // Problem unsatisfiable.
                  }
            }
         else
            { return SATISFIABLE; }           // All variables are assigned, problem satisfiable.
      }
}
```

Not explicitly shown: Deletion of conflict clauses, restarts, or outputted model.

Decision Heuristics

- Job: Select a free variable and assign it a value.
  Selected variable is referred to as a Decision Variable.
- Comparable to the branch selection in the DLL Algorithm.
- Has a significant impact on the search process.
- Modern SAT algorithms do not test if every clause in the CNF formula is satisfied during the search. Instead, if the are no more free variable to select as decision, and no conflicts exist, the problem is satisfiable.
  - Example: $F = (x_1, x_2, x_3) \wedge (\neg x_1, x_4)$

  $\Rightarrow$ Satisfying assignment: $x_1 = 1, x_4 = 1$

  $\Rightarrow$ Current solver do not test if $x_1 = x_4 = 1$ satisfies all the clauses, instead they will continue until an are variables are assigned (e.g. $x_2 = x_3 = 0$). Only then will they output *SATISFIABLE*.

Decision Heuristics

"Classical" Decision Heuristics

- Many variants:
  - Dynamic Largest Individual/Combined Sum
  - Maximum Occurrences on Clauses of Minimal Size
- Selection criteria:
  - "How often does a unassigned variable appear in the current remaining formula?"
  - Select the unassigned with the highest count as the next decision variable.
  - Can always weigh each variables score with the size of the clauses it appears in.
- These are termed computationally expensive heuristics as they must keep track of the current variable distributions as clauses are deleted/added/re-added/shortened/...

  $\Rightarrow$ Computational complexity is determined by # of clauses

Decision Heuristics

Variable State Independent Decaying Sum (zChaff)

- Standard heuristic used by most modern SAT solvers.
- Computational complexity is determined by # of variables.
- No computation required due to backtracking.
- Every variable x_i has two activity counters: P_{x_i} and N_{x_i}.
- Each of these counters is incremented for every literal L that appears in a clause C that is part of any new clause:

$$\begin{aligned} P_{x_i} &= P_{x_i} + 1, \text{ case } L = x_i \\ N_{x_i} &= N_{x_i} + 1, \text{ case } L = \neg x_i \end{aligned}$$

- The decision variable is selected to be the variable x_i with the largest activity (P_{x_i} or N_{x_i}).
- The positive or negative assignment of this variable depends on if $P_{x_i} > N_{x_i}$.

Decision Heuristics

Variable State Independent Decaying Sum (zChaff)

- The variable activity counters are periodically "normalized" (e.g. divide by a constant).
  - $\Rightarrow$ Because of normalization, newly generated conflict clauses have a larger impact on the current decision process than older clauses.
  - $\Rightarrow$ The "history" of the search process is taken into account.
- Many optimization opportunities:
  - By what amount should the activities be incremented?
  - How often should the activities be normalized?
  - During normalization, what division factor should be used?

Main procedure of a modern sequential SAT Algorithm

```
bool SequentialSatEngine(CNF F)
{
  if (PreprocessCNF(F) == CONFLICT)                          // Simplify the CNF formula.
    { return UNSATISFIABLE; }                                // Problem is unsatisfiable.
  while (true)
    {
      if (DecideNextBranch())                                // Select a free variable and assign it a value.
        {
          while (BCP() == CONFLICT)                          // Boolean Constraint Propagation.
            {
              BLevel = AnalyzeConflict();                    // Conflict analysis.
              if (BLevel > 0)
                { Backtrack(BLevel); }                       // Backtrack to a previous decision.
              else
                { return UNSATISFIABLE; }                    // Problem unsatisfiable.
            }
        }
      else
        { return SATISFIABLE; }                              // All variables are assigned, problem satisfiable.
    }
}
```

Not explicitly shown: Deletion of conflict clauses, restarts, or outputted model.

Boolean Constraint Propagation

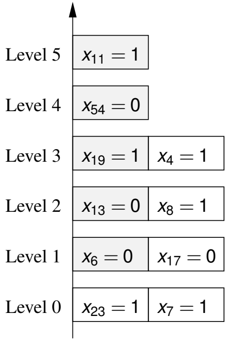
- Task:
  - Find all the implications that are the result of the current decision variable.
  - Detect conflicts if they exist under the current assignment.
- Comparable to the repeatedly called Unit clause rule in the original DLL Algorithm.
- An efficient implementation is required. Even today, with special data structures and techniques, the BCP procedure accounts for ≈ 80 of the total run time of the solver.

Boolean Constraint Propagation

General procedure:

- After ever variable assignment it must identify every resulting implications. The resulting implications are then stored in an Implication Queue, and processed one after another.
- As long as the implication queue is not empty:
  1. Delete the first element in the queue.
  2. Assign the the implied value of the variable in the decision stack.
  3. Check to see if this new assignment forces more implications. If so, add them to the implication queue.

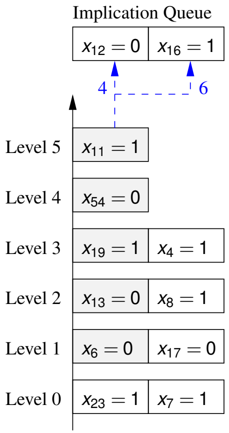
Boolean Constraint Propagation



$$F = \underbrace{(x_{23})}_{1} \wedge \underbrace{(x_7, \neg x_{23})}_{2} \wedge \underbrace{(x_6, \neg x_{17})}_{3} \wedge \underbrace{(x_6, \neg x_{11}, \neg x_{12})}_{4} \wedge \underbrace{(x_{13}, x_8)}_{5} \wedge \underbrace{(\neg x_{11}, x_{13}, x_{16})}_{6} \wedge \underbrace{(x_{12}, \neg x_{16}, \neg x_2)}_{7} \wedge \underbrace{(x_2, \neg x_4, \neg x_{10})}_{8} \wedge$$
$$\underbrace{(\neg x_{19}, x_4)}_{9} \wedge \underbrace{(x_{10}, \neg x_5)}_{10} \wedge \underbrace{(x_{10}, x_3)}_{11} \wedge \underbrace{(x_{10}, \neg x_8, x_1)}_{12} \wedge \underbrace{(\neg x_{19}, \neg x_{18}, \neg x_3)}_{13} \wedge \underbrace{(x_{17}, \neg x_1, x_{18}, \neg x_3, x_5)}_{14} \wedge \dots$$

Boolean Constraint Propagation

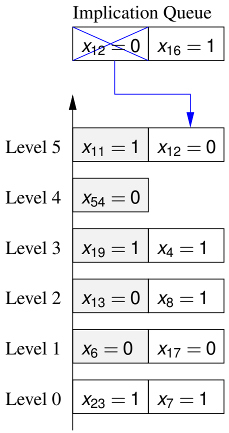
Albert-Ludwigs-Universität Freiburg



$$F = \underbrace{(x_{23})}_{1} \wedge \underbrace{(x_7, \neg x_{23})}_{2} \wedge \underbrace{(x_6, \neg x_{17})}_{3} \wedge \underbrace{(x_6, \neg x_{11}, \neg x_{12})}_{4} \wedge \underbrace{(x_{13}, x_8)}_{5} \wedge \underbrace{(\neg x_{11}, x_{13}, x_{16})}_{6} \wedge \underbrace{(x_{12}, \neg x_{16}, \neg x_2)}_{7} \wedge \underbrace{(x_2, \neg x_4, \neg x_{10})}_{8} \wedge$$

$$\underbrace{(\neg x_{19}, x_4)}_{9} \wedge \underbrace{(x_{10}, \neg x_5)}_{10} \wedge \underbrace{(x_{10}, x_3)}_{11} \wedge \underbrace{(x_{10}, \neg x_8, x_1)}_{12} \wedge \underbrace{(\neg x_{19}, \neg x_{18}, \neg x_3)}_{13} \wedge \underbrace{(x_{17}, \neg x_1, x_{18}, \neg x_3, x_5)}_{14} \wedge \ldots$$

Boolean Constraint Propagation

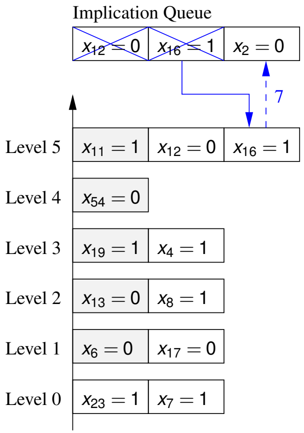


$$F = \underbrace{(x_{23})}_{1} \wedge \underbrace{(x_7, \neg x_{23})}_{2} \wedge \underbrace{(x_6, \neg x_{17})}_{3} \wedge \underbrace{(x_6, \neg x_{11}, \neg x_{12})}_{4} \wedge \underbrace{(x_{13}, x_8)}_{5} \wedge \underbrace{(\neg x_{11}, x_{13}, x_{16})}_{6} \wedge \underbrace{(x_{12}, \neg x_{16}, \neg x_2)}_{7} \wedge \underbrace{(x_2, \neg x_4, \neg x_{10})}_{8} \wedge$$

$$\underbrace{(\neg x_{19}, x_4)}_{9} \wedge \underbrace{(x_{10}, \neg x_5)}_{10} \wedge \underbrace{(x_{10}, x_3)}_{11} \wedge \underbrace{(x_{10}, \neg x_8, x_1)}_{12} \wedge \underbrace{(\neg x_{19}, \neg x_{18}, \neg x_3)}_{13} \wedge \underbrace{(x_{17}, \neg x_1, x_{18}, \neg x_3, x_5)}_{14} \wedge \ldots$$

Boolean Constraint Propagation

Albert-Ludwigs-Universität Freiburg

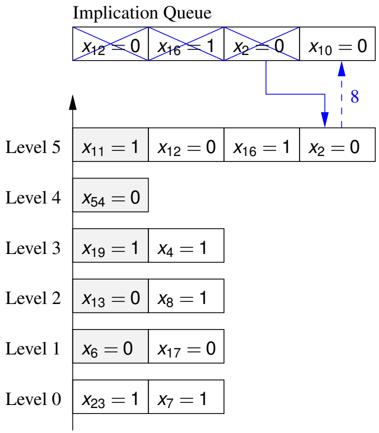


$$F = \underbrace{(x_{23})}_{1} \wedge \underbrace{(x_7, \neg x_{23})}_{2} \wedge \underbrace{(x_6, \neg x_{17})}_{3} \wedge \underbrace{(x_6, \neg x_{11}, \neg x_{12})}_{4} \wedge \underbrace{(x_{13}, x_8)}_{5} \wedge \underbrace{(\neg x_{11}, x_{13}, x_{16})}_{6} \wedge \underbrace{(x_{12}, \neg x_{16}, \neg x_2)}_{7} \wedge \underbrace{(x_2, \neg x_4, \neg x_{10})}_{8} \wedge$$

$$\underbrace{(\neg x_{19}, x_4)}_{9} \wedge \underbrace{(x_{10}, \neg x_5)}_{10} \wedge \underbrace{(x_{10}, x_3)}_{11} \wedge \underbrace{(x_{10}, \neg x_8, x_1)}_{12} \wedge \underbrace{(\neg x_{19}, \neg x_{18}, \neg x_3)}_{13} \wedge \underbrace{(x_{17}, \neg x_1, x_{18}, \neg x_3, x_5)}_{14} \wedge \ldots$$

Boolean Constraint Propagation

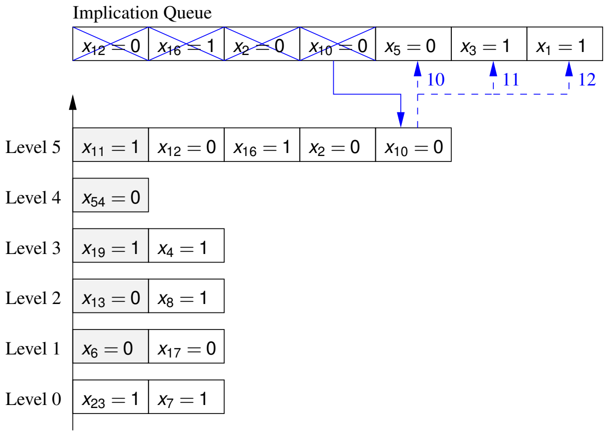
Albert-Ludwigs-Universität Freiburg



$$F = \underbrace{(x_{23})}_{1} \wedge \underbrace{(x_7, \neg x_{23})}_{2} \wedge \underbrace{(x_6, \neg x_{17})}_{3} \wedge \underbrace{(x_6, \neg x_{11}, \neg x_{12})}_{4} \wedge \underbrace{(x_{13}, x_8)}_{5} \wedge \underbrace{(\neg x_{11}, x_{13}, x_{16})}_{6} \wedge \underbrace{(x_{12}, \neg x_{16}, \neg x_2)}_{7} \wedge \underbrace{(x_2, \neg x_4, \neg x_{10})}_{8} \wedge$$

$$\underbrace{(\neg x_{19}, x_4)}_{9} \wedge \underbrace{(x_{10}, \neg x_5)}_{10} \wedge \underbrace{(x_{10}, x_3)}_{11} \wedge \underbrace{(x_{10}, \neg x_8, x_1)}_{12} \wedge \underbrace{(\neg x_{19}, \neg x_{18}, \neg x_3)}_{13} \wedge \underbrace{(x_{17}, \neg x_1, x_{18}, \neg x_3, x_5)}_{14} \wedge \dots$$

Boolean Constraint Propagation

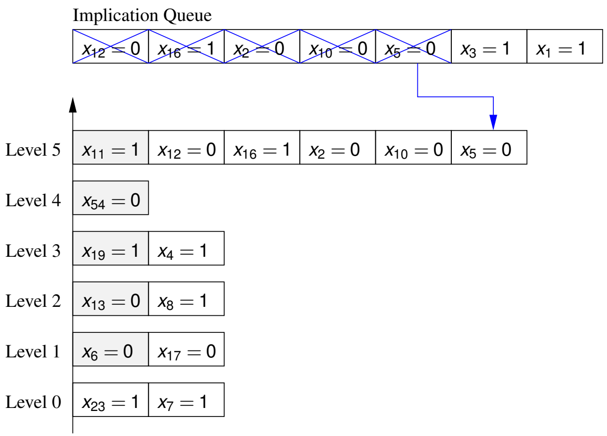


$$F = \underbrace{(x_{23})}_{1} \wedge \underbrace{(x_7, \neg x_{23})}_{2} \wedge \underbrace{(x_6, \neg x_{17})}_{3} \wedge \underbrace{(x_6, \neg x_{11}, \neg x_{12})}_{4} \wedge \underbrace{(x_{13}, x_8)}_{5} \wedge \underbrace{(\neg x_{11}, x_{13}, x_{16})}_{6} \wedge \underbrace{(x_{12}, \neg x_{16}, \neg x_2)}_{7} \wedge \underbrace{(x_2, \neg x_4, \neg x_{10})}_{8} \wedge$$

$$\underbrace{(\neg x_{19}, x_4)}_{9} \wedge \underbrace{(x_{10}, \neg x_5)}_{10} \wedge \underbrace{(x_{10}, x_3)}_{11} \wedge \underbrace{(x_{10}, \neg x_8, x_1)}_{12} \wedge \underbrace{(\neg x_{19}, \neg x_{18}, \neg x_3)}_{13} \wedge \underbrace{(x_{17}, \neg x_1, x_{18}, \neg x_3, x_5)}_{14} \wedge \ldots$$

Boolean Constraint Propagation

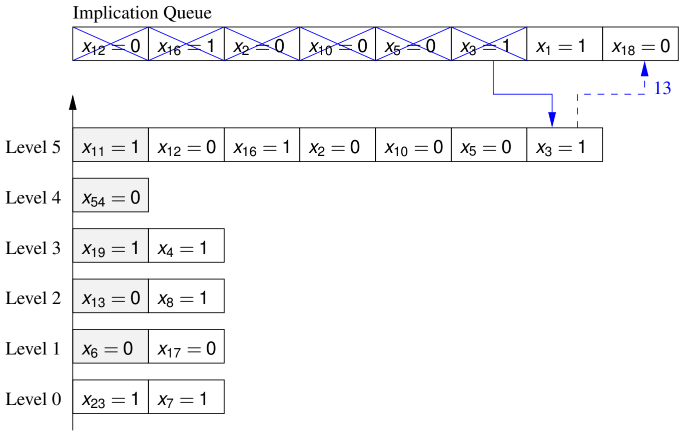
Albert-Ludwigs-Universität Freiburg



$$F = \underbrace{(x_{23})}_{1} \wedge \underbrace{(x_7, \neg x_{23})}_{2} \wedge \underbrace{(x_6, \neg x_{17})}_{3} \wedge \underbrace{(x_6, \neg x_{11}, \neg x_{12})}_{4} \wedge \underbrace{(x_{13}, x_8)}_{5} \wedge \underbrace{(\neg x_{11}, x_{13}, x_{16})}_{6} \wedge \underbrace{(x_{12}, \neg x_{16}, \neg x_2)}_{7} \wedge \underbrace{(x_2, \neg x_4, \neg x_{10})}_{8} \wedge$$

$$\underbrace{(\neg x_{19}, x_4)}_{9} \wedge \underbrace{(x_{10}, \neg x_5)}_{10} \wedge \underbrace{(x_{10}, x_3)}_{11} \wedge \underbrace{(x_{10}, \neg x_8, x_1)}_{12} \wedge \underbrace{(\neg x_{19}, \neg x_{18}, \neg x_3)}_{13} \wedge \underbrace{(x_{17}, \neg x_1, x_{18}, \neg x_3, x_5)}_{14} \wedge \ldots$$

Boolean Constraint Propagation

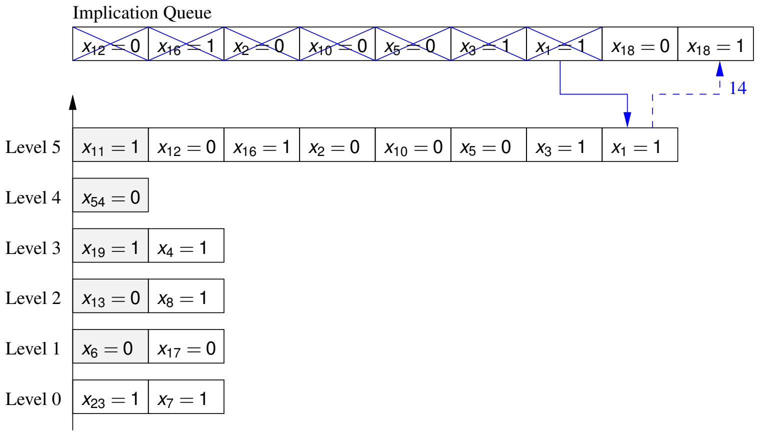


$$F = \underbrace{(x_{23})}_{1} \wedge \underbrace{(x_7, \neg x_{23})}_{2} \wedge \underbrace{(x_6, \neg x_{17})}_{3} \wedge \underbrace{(x_6, \neg x_{11}, \neg x_{12})}_{4} \wedge \underbrace{(x_{13}, x_8)}_{5} \wedge \underbrace{(\neg x_{11}, x_{13}, x_{16})}_{6} \wedge \underbrace{(x_{12}, \neg x_{16}, \neg x_2)}_{7} \wedge \underbrace{(x_2, \neg x_4, \neg x_{10})}_{8} \wedge$$

$$\underbrace{(\neg x_{19}, x_4)}_{9} \wedge \underbrace{(x_{10}, \neg x_5)}_{10} \wedge \underbrace{(x_{10}, x_3)}_{11} \wedge \underbrace{(x_{10}, \neg x_8, x_1)}_{12} \wedge \underbrace{(\neg x_{19}, \neg x_{18}, \neg x_3)}_{13} \wedge \underbrace{(x_{17}, \neg x_1, x_{18}, \neg x_3, x_5)}_{14} \wedge \ldots$$

Boolean Constraint Propagation

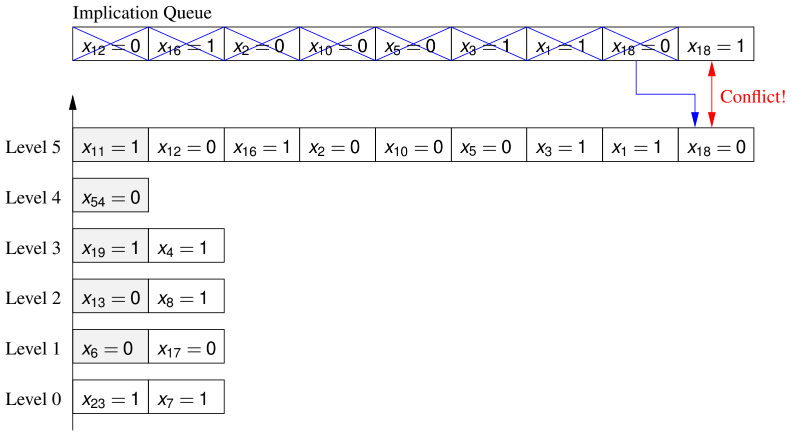
Albert-Ludwigs-Universität Freiburg



$$F = \underbrace{(x_{23})}_{1} \wedge \underbrace{(x_7, \neg x_{23})}_{2} \wedge \underbrace{(x_6, \neg x_{17})}_{3} \wedge \underbrace{(x_6, \neg x_{11}, \neg x_{12})}_{4} \wedge \underbrace{(x_{13}, x_8)}_{5} \wedge \underbrace{(\neg x_{11}, x_{13}, x_{16})}_{6} \wedge \underbrace{(x_{12}, \neg x_{16}, \neg x_2)}_{7} \wedge \underbrace{(x_2, \neg x_4, \neg x_{10})}_{8} \wedge$$

$$\underbrace{(\neg x_{19}, x_4)}_{9} \wedge \underbrace{(x_{10}, \neg x_5)}_{10} \wedge \underbrace{(x_{10}, x_3)}_{11} \wedge \underbrace{(x_{10}, \neg x_8, x_1)}_{12} \wedge \underbrace{(\neg x_{19}, \neg x_{18}, \neg x_3)}_{13} \wedge \underbrace{(x_{17}, \neg x_1, x_{18}, \neg x_3, x_5)}_{14} \wedge \ldots$$

Boolean Constraint Propagation



$$F = \underbrace{(x_{23})}_{1} \wedge \underbrace{(x_7, \neg x_{23})}_{2} \wedge \underbrace{(x_6, \neg x_{17})}_{3} \wedge \underbrace{(x_6, \neg x_{11}, \neg x_{12})}_{4} \wedge \underbrace{(x_{13}, x_8)}_{5} \wedge \underbrace{(\neg x_{11}, x_{13}, x_{16})}_{6} \wedge \underbrace{(x_{12}, \neg x_{16}, \neg x_2)}_{7} \wedge \underbrace{(x_2, \neg x_4, \neg x_{10})}_{8} \wedge$$

$$\underbrace{(\neg x_{19}, x_4)}_{9} \wedge \underbrace{(x_{10}, \neg x_5)}_{10} \wedge \underbrace{(x_{10}, x_3)}_{11} \wedge \underbrace{(x_{10}, \neg x_8, x_1)}_{12} \wedge \underbrace{(\neg x_{19}, \neg x_{18}, \neg x_3)}_{13} \wedge \underbrace{(x_{17}, \neg x_1, x_{18}, \neg x_3, x_5)}_{14} \wedge \dots$$

Boolean Constraint Propagation

- Task:
  - Find all the implications that are the result of the current decision variable.
  - Detect conflicts if they exist under the current assignment.
- Possible approaches to implement the BCP procedure:
  - Counter-Based schemes
  - Head/Tail list scheme
  - Watched literals / 2-literal watching scheme

Counter-Based Schemes

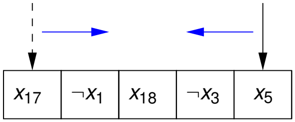
- 2-counter scheme
  - Two counters per clause:
    - One for the literals that satisfy the clause.
    - One for the literals that are still unassigned.
- 1-counter scheme
  - One counter per clause counting the number of falsely assigned literals.
- Disadvantages
  - “Unneeded” counter updates.
  - Counters must be updated during backtracking.
  - Requires a list for every polarity of every variable, that maintains where each variable is stored in each clauses. This list must be updated as clauses are added and removed.

Head/Tail List Scheme

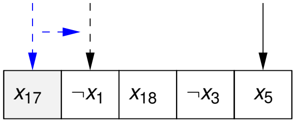
- Two pointer per clause:
  - Head pointer
  - Tail pointer
- Invariants:
  - In a clause, the literals contained left of the head pointer and right of the tail pointer must be falsely assigned.
  - Literals that are pointed to by either the Head or Tail pointers must be unassigned, or properly assigned and fulfill the clause.
- Advantages over counter based schemes:
  - Update operations are only needed when the invariants about for the clause are broken.
  - For both polarities of each variable a list is needed so that each clause knows its current head and tail pointers.
- Disadvantages:
  - Pointers must be updated during backtracking.

Head/Tail List Scheme

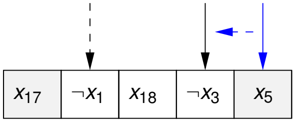
Albert-Ludwigs-Universität Freiburg



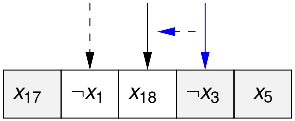
(a) Initial state



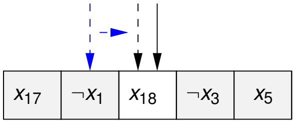
(b) $x_{17} = 0$



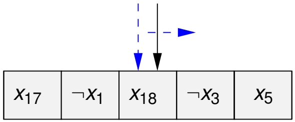
(c) $x_5 = 0$



(d) $x_3 = 1$



(e) $x_1 = 1 \Rightarrow x_{18} = 1$

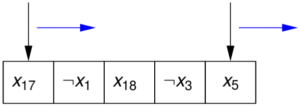


(f) $x_{18} = 0 \Rightarrow$ Conflict!

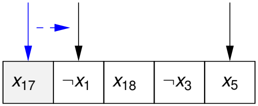
Watched Literals

- For every clause we “watch” 2 literals.
- Invariant:
  - The two watched literals in a clause must either be unassigned, or at least one must be properly assigned.
- Advantages over counter based schemes:
  - Update operations are only needed when the invariant about a clause is broken.
  - For both polarities of each variable a list is needed so that each clause knows its current head and tail pointers.
- Advantages over Head/Tail list scheme:
  - No work to do during backtracking.
- Disadvantages
  - Literals in every clauses are normally evaluated more than once.

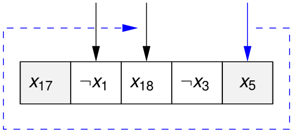
Watched Literals



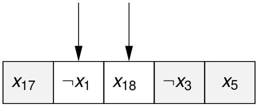
(a) Initial state



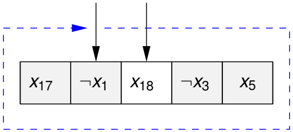
(b) $x_{17} = 0$



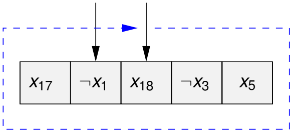
(c) $x_5 = 0$



(d) $x_3 = 1$



(e) $x_1 = 1 \Rightarrow x_{18} = 1$



(f) $x_{18} = 0 \Rightarrow$ Conflict!

Watched Literals

- Possible optimizations:
  - Save the first two watched literals in the first two locations of every clause.
    - Fast access to the “other” watched literal, so that its status can be checked.
    - If the “other” watched literal is satisfied, the BCP can skip its processing of this clause.
- Watched literals have been used for quite some time and are now standard in every modern SAT algorithm.

Main procedure of a modern sequential SAT Algorithm

```
bool SequentialSatEngine(CNF F)
{
  if (PreprocessCNF(F) == CONFLICT)                              // Simplify the CNF formula.
    { return UNSATISFIABLE; }                                    // Problem is unsatisfiable.
  while (true)
    {
      if (DecideNextBranch())                    // Select a free variable and assign it a value.
        {
          while (BCP() == CONFLICT)                         // Boolean Constraint Propagation.
            {
              BLevel = AnalyzeConflict();                                // Conflict analysis.
              if (BLevel > 0)
                { Backtrack(BLevel); }                      // Backtrack to a previous decision.
              else
                { return UNSATISFIABLE; }                             // Problem unsatisfiable.
            }
        }
      else
        { return SATISFIABLE; }              // All variables are assigned, problem satisfiable.
    }
}
```

Not explicitly shown: Deletion of conflict clauses, restarts, or outputted model.

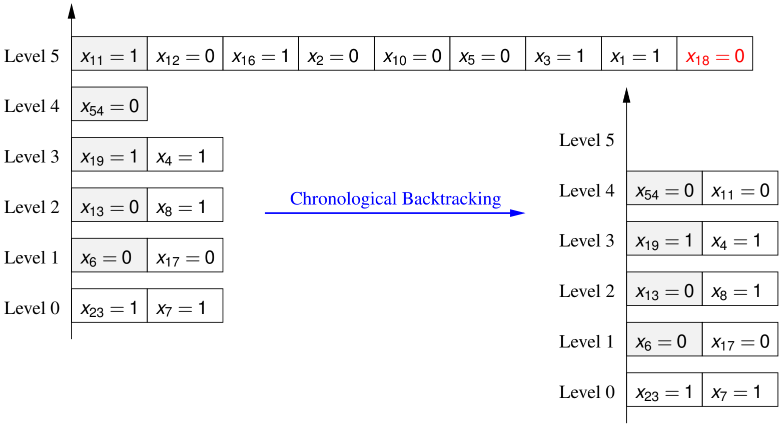
Conflict Analysis and Backtracking

DLL Algorithm

- Conflicts are always directly related to the current selected branch.
- Backtracking to the last branch, in which only one path was searched (called Chronological Backtracking).
- If all both cases of every variables have been tried, the current CNF formula is unsatisfiable.

Conflict Analysis and Backtracking

Albert-Ludwigs-Universität Freiburg



$$F = \underbrace{(x_{23})}_{1} \wedge \underbrace{(x_7,\neg x_{23})}_{2} \wedge \underbrace{(x_6,\neg x_{17})}_{3} \wedge \underbrace{(x_6,\neg x_{11},\neg x_{12})}_{4} \wedge \underbrace{(x_{13},x_8)}_{5} \wedge \underbrace{(\neg x_{11},x_{13},x_{16})}_{6} \wedge \underbrace{(x_{12},\neg x_{16},\neg x_2)}_{7} \wedge \underbrace{(x_2,\neg x_4,\neg x_{10})}_{8} \wedge$$
$$\underbrace{(\neg x_{19},x_4)}_{9} \wedge \underbrace{(x_{10},\neg x_5)}_{10} \wedge \underbrace{(x_{10},x_3)}_{11} \wedge \underbrace{(x_{10},\neg x_8,x_1)}_{12} \wedge \underbrace{(\neg x_{19},\neg x_{18},\neg x_3)}_{13} \wedge \underbrace{(x_{17},\neg x_1,x_{18},\neg x_3,x_5)}_{14} \wedge \ldots$$

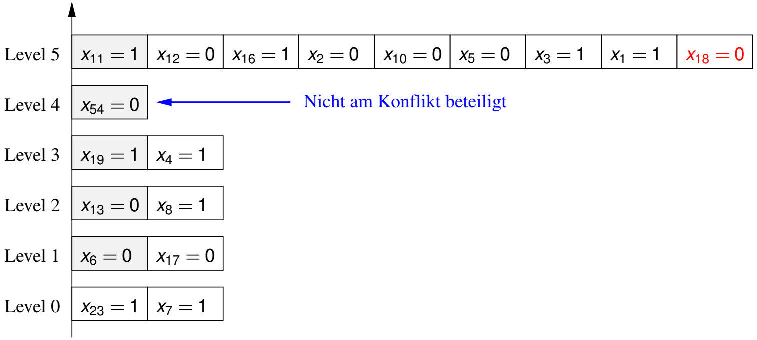
Conflict Analysis and Backtracking

Modern SAT algorithms:

- Do a deeper analysis of the current conflict situation to find out which decisions and implications are actually involved in the conflict.
- Generate (by resolution) and add a conflict clause to the current formula. The conflict clause contains all the literals that were responsible for the current conflict. The conflict clause can now be used to alleviate the current conflict, and possibly future conflicts.
- Using the conflict clause, backtrack to a previous decision level. In many cases, this is significantly earlier than the current decision level). If the conflict clause cannot be satisfied, the problem is *UNSATISFIABLE*. This process is referred to as Non-chronological Backtracking.

Conflict Analysis and Backtracking

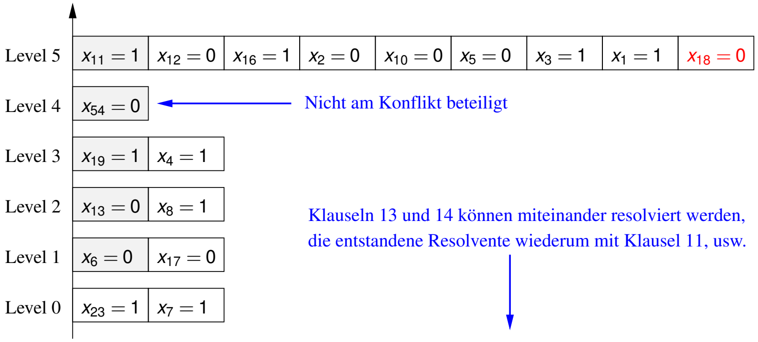
Albert-Ludwigs-Universität Freiburg



$$F = \underbrace{(x_{23})}_{1} \wedge \underbrace{(x_7, \neg x_{23})}_{2} \wedge \underbrace{(x_6, \neg x_{17})}_{3} \wedge \underbrace{(x_6, \neg x_{11}, \neg x_{12})}_{4} \wedge \underbrace{(x_{13}, x_8)}_{5} \wedge \underbrace{(\neg x_{11}, x_{13}, x_{16})}_{6} \wedge \underbrace{(x_{12}, \neg x_{16}, \neg x_2)}_{7} \wedge \underbrace{(x_2, \neg x_4, \neg x_{10})}_{8} \wedge$$
$$\underbrace{(\neg x_{19}, x_4)}_{9} \wedge \underbrace{(x_{10}, \neg x_5)}_{10} \wedge \underbrace{(x_{10}, x_3)}_{11} \wedge \underbrace{(x_{10}, \neg x_8, x_1)}_{12} \wedge \underbrace{(\neg x_{19}, \neg x_{18}, \neg x_3)}_{13} \wedge \underbrace{(x_{17}, \neg x_1, x_{18}, \neg x_3, x_5)}_{14} \wedge \ldots$$

Conflict Analysis and Backtracking

Albert-Ludwigs-Universität Freiburg



Klauseln 13 und 14 können miteinander resolviert werden, die entstandene Resolvente wiederum mit Klausel 11, usw.

$$F = \underbrace{(x_{23})}_{1} \wedge \underbrace{(x_7, \neg x_{23})}_{2} \wedge \underbrace{(x_6, \neg x_{17})}_{3} \wedge \underbrace{(x_6, \neg x_{11}, \neg x_{12})}_{4} \wedge \underbrace{(x_{13}, x_8)}_{5} \wedge \underbrace{(\neg x_{11}, x_{13}, x_{16})}_{6} \wedge \underbrace{(x_{12}, \neg x_{16}, \neg x_2)}_{7} \wedge \underbrace{(x_2, \neg x_4, \neg x_{10})}_{8} \wedge$$

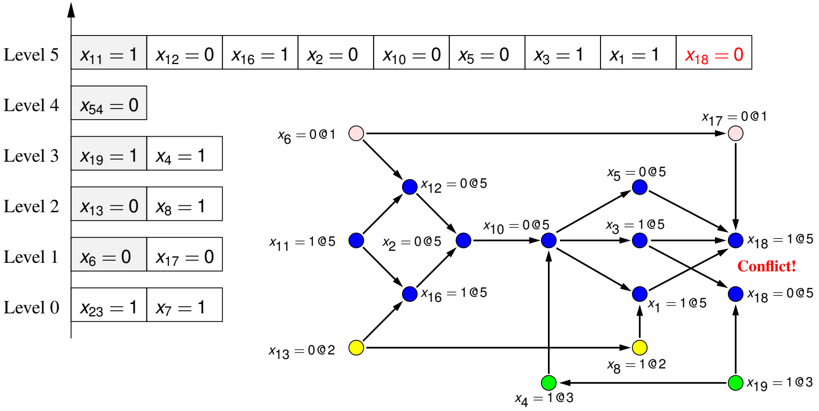
$$\underbrace{(\neg x_{19}, x_4)}_{9} \wedge \underbrace{(x_{10}, \neg x_5)}_{10} \wedge \underbrace{(x_{10}, x_3)}_{11} \wedge \underbrace{(x_{10}, \neg x_8, x_1)}_{12} \wedge \underbrace{(\neg x_{19}, \neg x_{18}, \neg x_3)}_{13} \wedge \underbrace{(x_{17}, \neg x_1, x_{18}, \neg x_3, x_5)}_{14} \wedge \ldots$$

Conflict Analysis and Backtracking

- To perform the conflict analysis in modern SAT algorithms we use the implication graph:
  - Directed, acyclic graph.
  - Nodes represent variable assignments.
  - Edges reflect the relationship between decision and implications.
- The implication graph changes during the search process and with every variable assignment an backtrack operation. However, the decision stack contains all the information we need to produce it when a conflict happens.

Conflict Analysis and Backtracking

Albert-Ludwigs-Universität Freiburg



Level 5: $x_{11}=1$ | $x_{12}=0$ | $x_{16}=1$ | $x_2=0$ | $x_{10}=0$ | $x_5=0$ | $x_3=1$ | $x_1=1$ | $x_{18}=0$
Level 4: $x_{54}=0$
Level 3: $x_{19}=1$ | $x_4=1$
Level 2: $x_{13}=0$ | $x_8=1$
Level 1: $x_6=0$ | $x_{17}=0$
Level 0: $x_{23}=1$ | $x_7=1$
$x_6=0@1$
$x_{17}=0@1$
$x_{12}=0@5$
$x_5=0@5$
$x_{11}=1@5$
$x_2=0@5$
$x_{10}=0@5$
$x_3=1@5$
$x_{18}=1@5$
Conflict!
$x_{16}=1@5$
$x_1=1@5$
$x_{18}=0@5$
$x_{13}=0@2$
$x_8=1@2$
$x_4=1@3$
$x_{19}=1@3$

$$F=\underbrace{(x_{23})}_{1}\wedge\underbrace{(x_7,\neg x_{23})}_{2}\wedge\underbrace{(x_6,\neg x_{17})}_{3}\wedge\underbrace{(x_6,\neg x_{11},\neg x_{12})}_{4}\wedge\underbrace{(x_{13},x_8)}_{5}\wedge\underbrace{(\neg x_{11},x_{13},x_{16})}_{6}\wedge\underbrace{(x_{12},\neg x_{16},\neg x_2)}_{7}\wedge\underbrace{(x_2,\neg x_4,\neg x_{10})}_{8}\wedge$$

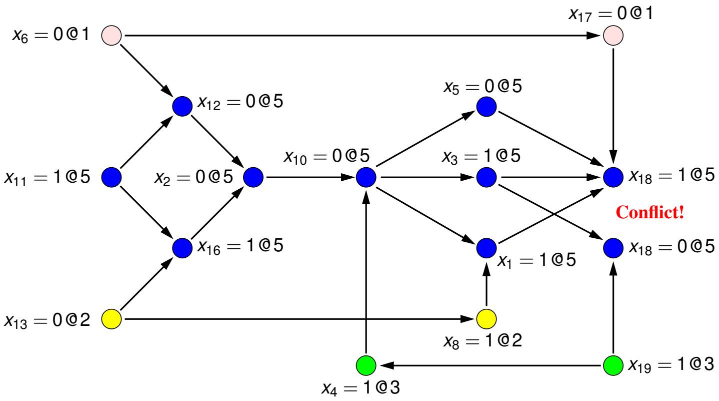
$$\underbrace{(\neg x_{19},x_4)}_{9}\wedge\underbrace{(x_{10},\neg x_5)}_{10}\wedge\underbrace{(x_{10},x_3)}_{11}\wedge\underbrace{(x_{10},\neg x_8,x_1)}_{12}\wedge\underbrace{(\neg x_{19},\neg x_{18},\neg x_3)}_{13}\wedge\underbrace{(x_{17},\neg x_1,x_{18},\neg x_3,x_5)}_{14}\wedge\ldots$$

Conflict Analysis and Backtracking

- During the conflict analysis routine, the implication graph is generated starting at the “conflict location”, and then working backwards in a chronological fashion with respect to the decision stack. The first clause to be examined, is the one that is conflicting, and as such, it is called the conflicting clause. Following all these resolution steps to the focal point of the problem allows us to generate a so called conflict clause.
- Various “stopping criteria” used to end the analysis process can result in different conflict clauses.
- Approaches:
  - 1UIP (Standard method, shown next)
  - RelSat
  - Grasp
  - ...

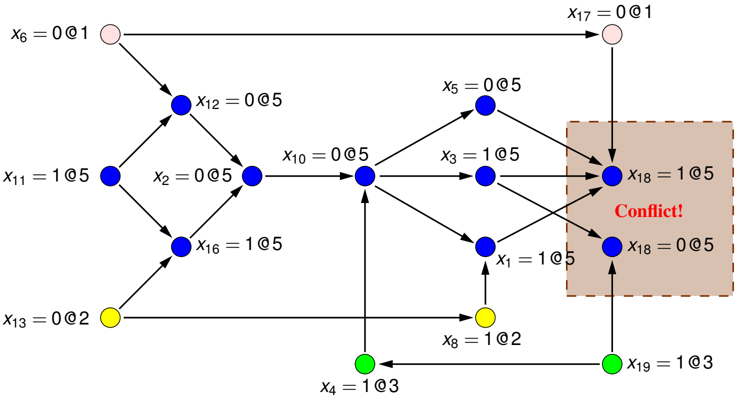
Conflict Analysis and Backtracking

Albert-Ludwigs-Universität Freiburg



$F = (x_{23}) \wedge (x_7, \neg x_{23}) \wedge (x_6, \neg x_{17}) \wedge (x_6, \neg x_{11}, \neg x_{12}) \wedge (x_{13}, x_8) \wedge (\neg x_{11}, x_{13}, x_{16}) \wedge (x_{12}, \neg x_{16}, \neg x_2) \wedge (x_2, \neg x_4, \neg x_{10}) \wedge$
$(\neg x_{19}, x_4) \wedge (x_{10}, \neg x_5) \wedge (x_{10}, x_3) \wedge (x_{10}, \neg x_8, x_1) \wedge (\neg x_{19}, \neg x_{18}, \neg x_3) \wedge (x_{17}, \neg x_1, x_{18}, \neg x_3, x_5) \wedge \ldots$

Conflict Analysis and Backtracking

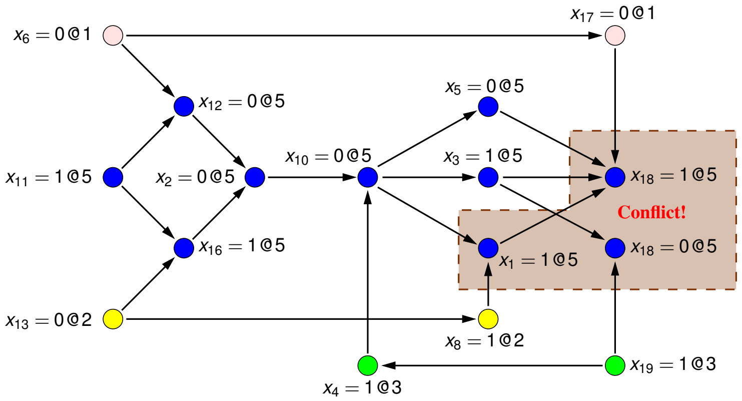


$F = (x_{23}) \wedge (x_7, \neg x_{23}) \wedge (x_6, \neg x_{17}) \wedge (x_6, \neg x_{11}, \neg x_{12}) \wedge (x_{13}, x_8) \wedge (\neg x_{11}, x_{13}, x_{16}) \wedge (x_{12}, \neg x_{16}, \neg x_2) \wedge (x_2, \neg x_4, \neg x_{10}) \wedge$
$(\neg x_{19}, x_4) \wedge (x_{10}, \neg x_5) \wedge (x_{10}, x_3) \wedge (x_{10}, \neg x_8, x_1) \wedge (\neg x_{19}, \neg x_{18}, \neg x_3) \wedge (x_{17}, \neg x_1, x_{18}, \neg x_3, x_5) \wedge \dots$

$R_1 = (x_{17}, \neg x_1, x_{18}, \neg x_3, x_5) \otimes_{x_{18}} (\neg x_{19}, \neg x_{18}, \neg x_3) = (x_{17}, \neg x_1, \neg x_3, x_5, \neg x_{19})$

Conflict Analysis and Backtracking

Albert-Ludwigs-Universität Freiburg

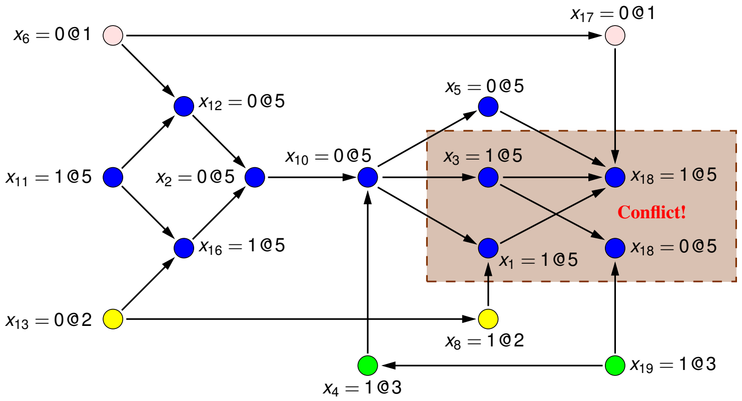


$F = (x_{23}) \wedge (x_7, \neg x_{23}) \wedge (x_6, \neg x_{17}) \wedge (x_6, \neg x_{11}, \neg x_{12}) \wedge (x_{13}, x_8) \wedge (\neg x_{11}, x_{13}, x_{16}) \wedge (x_{12}, \neg x_{16}, \neg x_2) \wedge (x_2, \neg x_4, \neg x_{10}) \wedge$
$(\neg x_{19}, x_4) \wedge (x_{10}, \neg x_5) \wedge (x_{10}, x_3) \wedge (x_{10}, \neg x_8, x_1) \wedge (\neg x_{19}, \neg x_{18}, \neg x_3) \wedge (x_{17}, \neg x_1, x_{18}, \neg x_3, x_5) \wedge \ldots$

$R_1 = (x_{17}, \neg x_1, x_{18}, \neg x_3, x_5) \otimes_{x_{18}} (\neg x_{19}, \neg x_{18}, \neg x_3) = (x_{17}, \neg x_1, \neg x_3, x_5, \neg x_{19})$

$R_2 = (x_{17}, \neg x_1, \neg x_3, x_5, \neg x_{19}) \otimes_{x_1} (x_1, x_{10}, \neg x_8) = (x_{17}, \neg x_3, x_5, \neg x_{19}, x_{10}, \neg x_8)$

Conflict Analysis and Backtracking



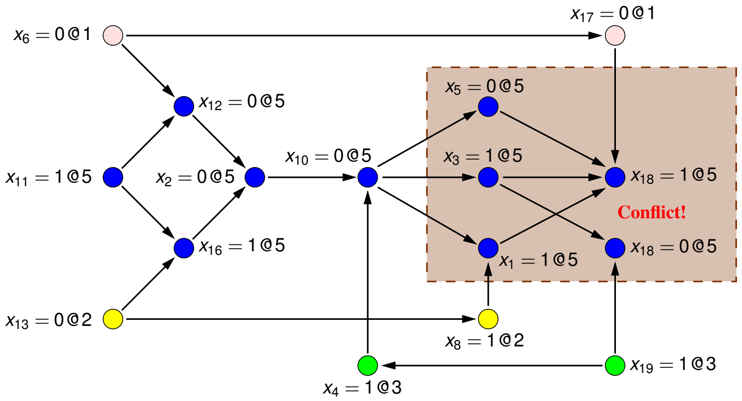
$F = (x_{23}) \wedge (x_7, \neg x_{23}) \wedge (x_6, \neg x_{17}) \wedge (x_6, \neg x_{11}, \neg x_{12}) \wedge (x_{13}, x_8) \wedge (\neg x_{11}, x_{13}, x_{16}) \wedge (x_{12}, \neg x_{16}, \neg x_2) \wedge (x_2, \neg x_4, \neg x_{10}) \wedge$
$(\neg x_{19}, x_4) \wedge (x_{10}, \neg x_5) \wedge (x_{10}, x_3) \wedge (x_{10}, \neg x_8, x_1) \wedge (\neg x_{19}, \neg x_{18}, \neg x_3) \wedge (x_{17}, \neg x_1, x_{18}, \neg x_3, x_5) \wedge \ldots$

$R_1 = (x_{17}, \neg x_1, x_{18}, \neg x_3, x_5) \otimes_{x_{18}} (\neg x_{19}, \neg x_{18}, \neg x_3) = (x_{17}, \neg x_1, \neg x_3, x_5, \neg x_{19})$

$R_2 = (x_{17}, \neg x_1, \neg x_3, x_5, \neg x_{19}) \otimes_{x_1} (x_1, x_{10}, \neg x_8) = (x_{17}, \neg x_3, x_5, \neg x_{19}, x_{10}, \neg x_8)$

$R_3 = (x_{17}, \neg x_3, x_5, \neg x_{19}, x_{10}, \neg x_8) \otimes_{x_3} (x_{10}, x_3) = (x_{17}, x_5, \neg x_{19}, x_{10}, \neg x_8)$

Conflict Analysis and Backtracking



$F = (x_{23}) \wedge (x_7, \neg x_{23}) \wedge (x_6, \neg x_{17}) \wedge (x_6, \neg x_{11}, \neg x_{12}) \wedge (x_{13}, x_8) \wedge (\neg x_{11}, x_{13}, x_{16}) \wedge (x_{12}, \neg x_{16}, \neg x_2) \wedge (x_2, \neg x_4, \neg x_{10}) \wedge$
$(\neg x_{19}, x_4) \wedge (x_{10}, \neg x_5) \wedge (x_{10}, x_3) \wedge (x_{10}, \neg x_8, x_1) \wedge (\neg x_{19}, \neg x_{18}, \neg x_3) \wedge (x_{17}, \neg x_1, x_{18}, \neg x_3, x_5) \wedge \dots$

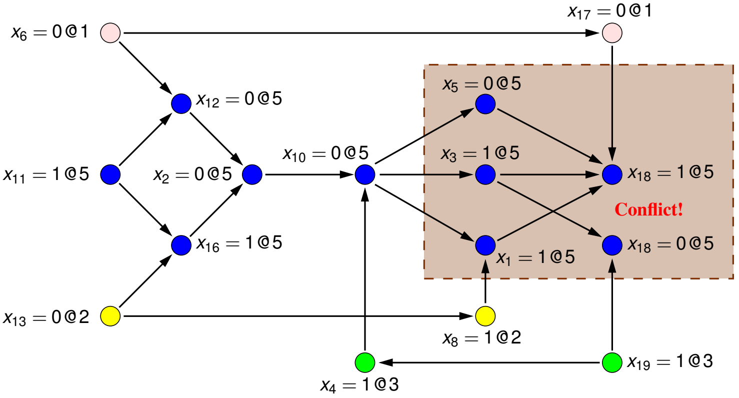
$R_1 = (x_{17}, \neg x_1, x_{18}, \neg x_3, x_5) \otimes_{x_{18}} (\neg x_{19}, \neg x_{18}, \neg x_3) = (x_{17}, \neg x_1, \neg x_3, x_5, \neg x_{19})$

$R_2 = (x_{17}, \neg x_1, \neg x_3, x_5, \neg x_{19}) \otimes_{x_1} (x_1, x_{10}, \neg x_8) = (x_{17}, \neg x_3, x_5, \neg x_{19}, x_{10}, \neg x_8)$

$R_3 = (x_{17}, \neg x_3, x_5, \neg x_{19}, x_{10}, \neg x_8) \otimes_{x_3} (x_{10}, x_3) = (x_{17}, x_5, \neg x_{19}, x_{10}, \neg x_8)$

$R_4 = (x_{17}, x_5, \neg x_{19}, x_{10}, \neg x_8) \otimes_{x_5} (x_{10}, \neg x_5) = (x_{17}, \neg x_{19}, x_{10}, \neg x_8)$

Conflict Analysis and Backtracking



$F = (x_{23}) \wedge (x_7, \neg x_{23}) \wedge (x_6, \neg x_{17}) \wedge (x_6, \neg x_{11}, \neg x_{12}) \wedge (x_{13}, x_8) \wedge (\neg x_{11}, x_{13}, x_{16}) \wedge (x_{12}, \neg x_{16}, \neg x_2) \wedge (x_2, \neg x_4, \neg x_{10}) \wedge$
$(\neg x_{19}, x_4) \wedge (x_{10}, \neg x_5) \wedge (x_{10}, x_3) \wedge (x_{10}, \neg x_8, x_1) \wedge (\neg x_{19}, \neg x_{18}, \neg x_3) \wedge (x_{17}, \neg x_1, x_{18}, \neg x_3, x_5) \wedge \ldots$

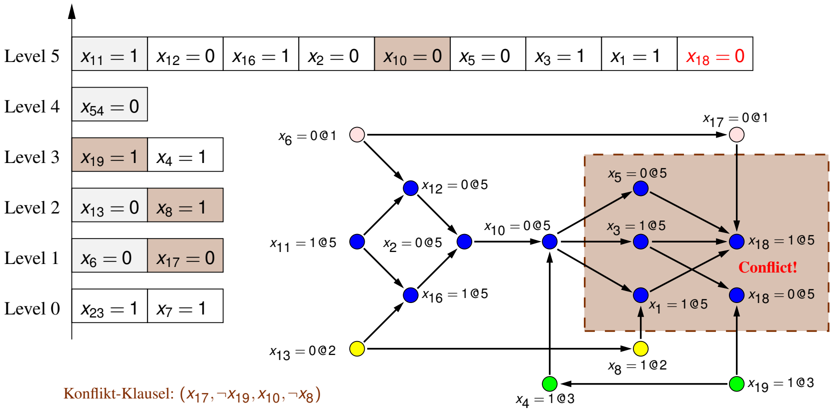
$R_1 = (x_{17}, \neg x_1, x_{18}, \neg x_3, x_5) \otimes_{x_{18}} (\neg x_{19}, \neg x_{18}, \neg x_3) = (x_{17}, \neg x_1, \neg x_3, x_5, \neg x_{19})$

$R_2 = (x_{17}, \neg x_1, \neg x_3, x_5, \neg x_{19}) \otimes_{x_1} (x_1, x_{10}, \neg x_8) = (x_{17}, \neg x_3, x_5, \neg x_{19}, x_{10}, \neg x_8)$

$R_3 = (x_{17}, \neg x_3, x_5, \neg x_{19}, x_{10}, \neg x_8) \otimes_{x_3} (x_{10}, x_3) = (x_{17}, x_5, \neg x_{19}, x_{10}, \neg x_8)$

$R_4 = (x_{17}, x_5, \neg x_{19}, x_{10}, \neg x_8) \otimes_{x_5} (x_{10}, \neg x_5) = (x_{17}, \neg x_{19}, x_{10}, \neg x_8) \Leftarrow$ Final conflict clause

Conflict Analysis and Backtracking



Konflikt-Klausel: $(x_{17}, \neg x_{19}, x_{10}, \neg x_8)$

$$F = \underbrace{(x_{23})}_{1} \wedge \underbrace{(x_7, \neg x_{23})}_{2} \wedge \underbrace{(x_6, \neg x_{17})}_{3} \wedge \underbrace{(x_6, \neg x_{11}, \neg x_{12})}_{4} \wedge \underbrace{(x_{13}, x_8)}_{5} \wedge \underbrace{(\neg x_{11}, x_{13}, x_{16})}_{6} \wedge \underbrace{(x_{12}, \neg x_{16}, \neg x_2)}_{7} \wedge \underbrace{(x_2, \neg x_4, \neg x_{10})}_{8} \wedge$$

$$\underbrace{(\neg x_{19}, x_4)}_{9} \wedge \underbrace{(x_{10}, \neg x_5)}_{10} \wedge \underbrace{(x_{10}, x_3)}_{11} \wedge \underbrace{(x_{10}, \neg x_8, x_1)}_{12} \wedge \underbrace{(\neg x_{19}, \neg x_{18}, \neg x_3)}_{13} \wedge \underbrace{(x_{17}, \neg x_1, x_{18}, \neg x_3, x_5)}_{14} \wedge \ldots$$

Conflict Analysis and Backtracking

Comments:

- The first UIP (First Unique Implication Point) conflict analysis strategy terminates it analysis when the resolvent clause only contains one literal from the current decision level (the so called UIP literal). this means all other literals must be falsely assigned on previous levels.
- The conflict clauses with the solvers current decision strategy inevitably lead to a conflict.

Conflict Analysis and Backtracking

Comments:

- The resolution lemmas allow us to add the conflict clauses directly to the CNF formula. This enables us to reduce the size of the total search space (i.e. the conflict clause will force implication allowing us to avoid searches in unsatisfiable parts of the search space).
- The 1UIP method has been compared to other approaches and is seen today as the most powerful in the case of SAT. This is because it produces shorter, more general clauses.

Conflict Analysis and Backtracking

Non-chronological backtracking

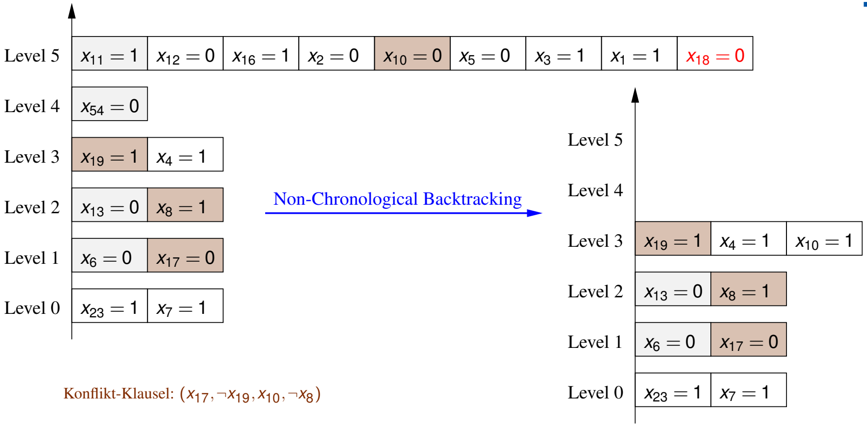
- In modern SAT algorithms, the conflict clause determines the backtrack level.
- The backtrack level is related to the literal with the highest decision level (with the exception of the UIP literal) in the conflict clause.
- Idea: “What would have happened had the conflict clause been part of the original formula?”

Conflict Analysis and Backtracking

Non-chronological backtracking

- Procedure:
  1. Backtrack to the backtrack level calculated as proposed.
  2. The conflict clause will then be a unit clause, and force the UIP literal.
  3. Continue the search process.
- If a conflict clause's UIP is already on decision level 0, the current CNF formula is unsatisfiable.

Conflict Analysis and Backtracking



Level 5 | $x_{11}=1$ | $x_{12}=0$ | $x_{16}=1$ | $x_2=0$ | $x_{10}=0$ | $x_5=0$ | $x_3=1$ | $x_1=1$ | $x_{18}=0$
Level 4 | $x_{54}=0$
Level 3 | $x_{19}=1$ | $x_4=1$
Level 2 | $x_{13}=0$ | $x_8=1$
Level 1 | $x_6=0$ | $x_{17}=0$
Level 0 | $x_{23}=1$ | $x_7=1$
Non-Chronological Backtracking
Level 5
Level 4
Level 3 | $x_{19}=1$ | $x_4=1$ | $x_{10}=1$
Level 2 | $x_{13}=0$ | $x_8=1$
Level 1 | $x_6=0$ | $x_{17}=0$
Level 0 | $x_{23}=1$ | $x_7=1$

Konflikt-Klausel: $(x_{17}, \neg x_{19}, x_{10}, \neg x_8)$

$$F = \underbrace{(x_{23})}_{1} \wedge \underbrace{(x_7, \neg x_{23})}_{2} \wedge \underbrace{(x_6, \neg x_{17})}_{3} \wedge \underbrace{(x_6, \neg x_{11}, \neg x_{12})}_{4} \wedge \underbrace{(x_{13}, x_8)}_{5} \wedge \underbrace{(\neg x_{11}, x_{13}, x_{16})}_{6} \wedge \underbrace{(x_{12}, \neg x_{16}, \neg x_2)}_{7} \wedge \underbrace{(x_2, \neg x_4, \neg x_{10})}_{8} \wedge$$

$$\underbrace{(\neg x_{19}, x_4)}_{9} \wedge \underbrace{(x_{10}, \neg x_5)}_{10} \wedge \underbrace{(x_{10}, x_3)}_{11} \wedge \underbrace{(x_{10}, \neg x_8, x_1)}_{12} \wedge \underbrace{(\neg x_{19}, \neg x_{18}, \neg x_3)}_{13} \wedge \underbrace{(x_{17}, \neg x_1, x_{18}, \neg x_3, x_5)}_{14} \wedge \ldots$$

Main procedure of a modern sequential SAT Algorithm

```
bool SequentialSatEngine(CNF F)
{
  if (PreprocessCNF(F) == CONFLICT)                         // Simplify the CNF formula.
    { return UNSATISFIABLE; }                               // Problem is unsatisfiable.
  while (true)
    {
      if (DecideNextBranch())                               // Select a free variable and assign it a value.
        {
          while (BCP() == CONFLICT)                         // Boolean Constraint Propagation.
            {
              BLevel = AnalyzeConflict();                   // Conflict analysis.
              if (BLevel > 0)
                { Backtrack(BLevel); }                      // Backtrack to a previous decision.
              else
                { return UNSATISFIABLE; }                   // Problem unsatisfiable.
            }
        }
      else
        { return SATISFIABLE; }                             // All variables are assigned, problem satisfiable.
    }
}
```

Not explicitly shown: Deletion of conflict clauses, restarts, or outputted model.

Deletion of Conflict Clauses

- A modern SAT solver generates and saves a conflict clause for every conflict it encounters.
- Problem:
  - Risk of memory requirements exploding.
  - Significant slowdown of the BCP procedure.
- Solution:
  - Periodically delete conflict clauses.
- When deleting clauses we must exclude:
  - Clauses that are part of the original CNF formula.
  - All clauses that are forcing implications in the current decision stack.
- We have to determine a balance between:
  - the deletion of information.
  - and the BCP and memory problems mention above.

Deletion of Conflict Clauses

Strategies:

- zChaff
  - Scheduled Lazy Clause Deletion / Relevance Based Learning
  - A static approach used to determine when all clauses are deleted.
  - For example: learnt clauses with more than 50 literals would be deleted when 30 literals become undefined.
- Grasp
  - Size-Bounded Learning / k-Bounded Learning
  - All clauses that exceed a predefined size would be deleted as soon as possible.

Deletion of Conflict Clauses

Strategies:

- BerkMin
  - Delete “old” and “inactive” clauses first.
  - Similar to variable activities used in decision strategies.
  - The activity of a clause is determined by how often it is used during resolution in the conflict analysis routine.
  - Idea: active clause are helping shrink the search space, and inactive clauses are just slowly down the BCP procedure.
  - The age of clause can have a similar effect, and can easily be calculated by its position in the clause set.

Deletion of Conflict Clauses

Strategies:

- MiniSat
  - Irrespective of a clauses age, inactive clauses are deleted.
  - The deletion procedure removes 50% of the learnt information after ever run.

Main procedure of a modern sequential SAT Algorithm

```
bool SEQUENTIALSATENGINE(CNF F)
{
  if (PREPROCESSCNF(F) == CONFLICT)                    // Simplify the CNF formula.
    { return UNSATISFIABLE; }                          // Problem is unsatisfiable.
  while (true)
    {
      if (DECIDENEXTBRANCH())                          // Select a free variable and assign it a value.
        {
          while (BCP() == CONFLICT)                    // Boolean Constraint Propagation.
            {
              BLevel = ANALYZECONFLICT();              // Conflict analysis.
              if (BLevel > 0)
                { BACKTRACK(BLevel); }                 // Backtrack to a previous decision.
              else
                { return UNSATISFIABLE; }              // Problem unsatisfiable.
            }
        }
      else
        { return SATISFIABLE; }                        // All variables are assigned, problem satisfiable.
    }
}
```

Not explicitly shown: Deletion of conflict clauses, restarts, or outputted model.

Restarts

- Method to move the SAT solver if it is “stuck” in an hard part of the search space.
- Basic idea:
    - The longer a SAT solver search for a model to a CNF problem, the higher the probability that:
        - The solver is in an unsatisfiable part of the search space.
        - On earlier decision, “bad” branches were taken.

Restarts

- Approach to restart:
  1. Stop the search process.
  2. Undo all variable assignments with the exception of those on decision level 0.
  3. Begin searching again on decision level 0.
- All previously learnt information is retained.
- Variable activities remain unchanged.
- Good chance that after a restart the solver will be in a different situation than before because:
  - The solver will choose other variables for the first decision levels.
  - The search process will be steered in other directions.

Restarts

- To prevent a SAT solver from repeating infinite loops because of restarting, the interval between each restart is usually slowly increased.
- Many optimization opportunities:
  - When should the first restart happen?
  - By how much should the interval between restart increase?
  - Can we intelligently decided when would be a good time to restart?
- Restarts not only aid the solvers performance on satisfiable instances, but on unsatisfiable ones as well.

Termination of Modern SAT Algorithms

- Consider:
  - Due to conflict analysis, learning, non-chronological backtracking, and restarts, the solvers can “wildly” jump around throughout the search space. So the question is: “Is a modern SAT solver guaranteed to terminated?”
- We need to consider the following when discussing termination:
  - Let F be a CNF formula with n variables.
    - ⇒ This problem requires at most $n+1$ Decision Level: $dl_0, \ldots, dl_n$
  - Let $k(dl_i)$ be the number of variables that are assigned on decision level dl_i.
    - ⇒ For all dl_i with $i \in \{0, \ldots, n\}$, it follows $k(dl_i) \leq n$ must be true.
    - ⇒ Similarly: $\sum_{i=0}^{n} k(dl_i) \leq n$
  - In the following ds will represent the current decision stack for the problem F.

Termination of Modern SAT Algorithms

- Continued ...
  - Let $f : ds \rightarrow \mathbb{N}$ be defined as $f(ds) = \sum_{d=0}^{n} \frac{k(dl_d)}{(n+1)^d}$

    $\Rightarrow$ Since the lower decision levels trim off larger parts of the search space, they are weighted higher in the formula. Moreover, $f(ds_x)$ has a higher weight than the sum of all decision levels following it. As such the following holds:

    $f(ds_1) > f(ds_2) \Leftrightarrow$

    $\exists i < n : k_{ds_1}(dl_i) > k_{ds_2}(dl_i) \wedge \forall j < i : k_{ds_1}(dl_j) = k_{ds_2}(dl_j)$

    $\Rightarrow$ Without restarts, $f(ds)$ will increase during the search. This is especially true with non-chronological backtracking.

    $\Rightarrow$ It does not depend on the deletion of clauses

    $\Rightarrow$ Because n is a fixed number, $f(ds)$ can only be increased so many times.

    $\Rightarrow$ So modern SAT-Solver without restarts do terminate.

    $\Rightarrow$ Restarts can be problematic in this regard. However, if the interval between restarts always increases, so too will the SAT solver.

Aspects not covered here

- Proofs of (un-)satisfiability of a CNF formula
- Incremental SAT solving
- Local search algorithms / incomplete SAT Algorithms
- Parallel SAT algorithms
- Quantified Boolean Formulae
- SAT Modulo Theory