SAT Solving

Armin Biere

Hausdorff School for Advanced Studies in Mathematics
Computational Combinatorial Optimization

Universität Bonn
September 12 - 16, 2022
propositional logic:

- variables tie shirt
- negation ¬ (not)
- disjunction ∨ (or)
- conjunction ∧ (and)

clauses (conditions / constraints)

1. clearly one should not wear a tie without a shirt ¬tie ∨ shirt
2. not wearing a tie nor a shirt is impolite tie ∨ shirt
3. wearing a tie and a shirt is overkill ¬(tie ∧ shirt) ≡ ¬tie ∨ ¬shirt

Is this formula in conjunctive normal form (CNF) satisfiable?

(¬tie ∨ shirt) ∧ (tie ∨ shirt) ∧ (¬tie ∨ ¬shirt)
All Time Winners on SAT Competition 2020 Benchmarks

- kissat–2020
- maple–lcm–dist–cb–2018
- maple–lcm–dist–2017
- lingeling–2014
- abcdsat–2015
- lingeling–2013
- glucose–2012
- glucose–2011
- precosat–2009
- cryptominisat–2010
- minisat–2008
- minisat–2006
- satelite–gti–2005
- rsat–2007
- berkmin–2003
- zchaff–2004
- chaff–2001
- limmat–2002
- grasp–1997

All Time Winners on SAT Competition 2020...
some recent Tweets

SAT solvers get faster and faster: all-time winners of the SAT Competition on 2020 instances, featuring our new solver Kissat (fmv.jku.at/kissat), which won in 2020. The web page also has runtime CDFs for 2011 and 2019.

The largest ones have millions of variables and clauses. The planning track had even larger ones. See the variable and clause distribution plot for the main track:

Hi,

We are trying to verify Deep Neural Networks with our verification machine ESBMC, that uses Boolelter. Our experiments are getting the following error:

- internal error in 'lglib.c': more than 134217724 variables.
  Could we increase this variable number? Since we are performing our experiments in a huge RAM memory.

You are receiving this because you are subscribed to this thread. Reply to this email directly, view it on GitHub, or unsubscribe.

Andrew V. Jones 13:40
an Boolelter/boolelter, S...

Can you try compiling Boolelter with a different SAT solver? I believe that CaDiCaL has a much higher limit (maybe INT_MAX vars).

Zitierten Text anzeigen

Aina Niemetz 18:16
an Boolelter/boolelter, S...

As @andrewvaughan points out, this is a limitation in the SAT solver that we can not control. Let me add that CaDiCaL typically outperforms Lingeling in combination with Boolelter, so it might be a good idea to switch to CaDiCaL anyways.
Top 20 solvers main sequential track
SAT Competition 2022
Top 20 Solver Sequential Anniversary Track
SAT Competition 2022
Cloud vs Parallel vs Sequential SAT Competition 2022

- mallob-kicaliglu (cloud winner)
- mallob-ki (parallel winner)
- kissat-sc2022-bulky (sequential winner)
Satisfiability (SAT) related topics have attracted researchers from various disciplines. Logic, applied areas such as planning, scheduling, operations research and combinatorial optimization, but also theoretical issues on the theme of complexity, and much more, are all connected through SAT.

My personal interest in SAT stems from actual solving. The increase in power of modern SAT solvers over the past 15 years has been phenomenal. It has become the key enabling technology in automated verification of both computer hardware and software. Boothed Model Checking (BMC) of computer hardware is now probably the most widely used model checking technique. The counterexamples that it finds are just satisfying instances of a Boolean formula obtained by unraveling to some fixed depth a sequential circuit and its specification in Linear Temporal Logic.

Extending model checking to software verification is a much more difficult problem on the frontier of current research. One promising approach for languages like C with finite word-length integers is to use the same idea as in BMC but with a decision procedure for the theory of bit-vectors instead of SAT. All decision procedures for bit-vectors that I am familiar with ultimately make use of a fast SAT solver to handle complex formulas.

Decision procedures for more complicated theories, like linear real and integer arithmetic, are also used in program verification. Most of them use powerful SAT solvers in an essential way.

Clearly, efficient SAT solving is a key technology for 21st century computer science. I expect this collection of papers on all theoretical and practical aspects of SAT solving will be extremely useful to both students and researchers and will lead to many further advances in the field.

Edmund Clarke

Edmund M. Clarke, FORE Systems University Professor of Computer Science and Professor of Electrical and Computer Engineering at Carnegie Mellon University, is one of the initiators and main contributors to the field of Model Checking, for which he also received the 2007 ACM Turing Award.

In the late 90s, Professor Clarke was one of the first researchers to realize that SAT solving has the potential to become one of the most important tools in model checking.

Editors:
Armin Biere
Marijn Heule
Hans van Maaren
Toby Walsh
Part I. Theory and Algorithms

- John Franco, John Martin: *A History of Satisfiability*. 3-74
- Steven David Prestwich: *CNF Encodings*. 75-97
- Adnan Darwiche, Knot Pipatsrisawat: *Complete Algorithms*. 99-130
- Marlijn Heule, Hans van Maaren: *Look-Ahead Based SAT Solvers*. 155-184
- Oliver Kullmann: *Fundamentals of Branching Heuristics*. 205-244
- Dimitris Achlioptas: *Random Satisfiability*. 245-270
- Carla P. Gomes, Ashish Sabharwal: *Exploiting Runtime Variation in Complete Solvers*. 271-288
- Karem A. Sakallah: *Symmetry and Satisfiability*. 289-338
- Hans Kleine Büning, Oliver Kullmann: *Minimal Unsatisfiability and Autarkies*. 339-401
- Marko Samer, Stefan Szeider: *Fixed-Parameter Tractability*. 425-454

Part II. Applications and Extensions

- Armin Biere: *Bounded Model Checking*. 457-481
- Jussi Rintanen: *Planning and SAT*. 483-504
- Daniel Kroening: *Software Verification*. 505-532
- Hantao Zhang: *Combinatorial Designs by SAT Solvers*. 533-568
- Fabrizio Albertelli, Rémi Monasson, Guilhem Semerjian, Francesco Zamponi: *Connections to Statistical Physics*. 569-611
- Chu Min Li, Felip Manyà: *MaxSAT, Hard and Soft Constraints*. 613-631
- Rolf Drechsler, Tommi A. Junttila, Ilkka Niemelä: *Non-Clausal SAT and ATPG*. 655-693
- Olivier Roussel, Vasco M. Manquinho: *Pseudo-Boolean and Cardinality Constraints*. 695-733
- Hans Kleine Büning, Uwe Bubeck: *Theory of Quantified Boolean Formulas*. 735-760
- Enrico Giunchiglia, Paolo Marin, Massimo Narizzano: *Reasoning with Quantified Boolean Formulas*. 761-780
- Roberto Sebastiani, Armando Tacchella: *SAT Techniques for Modal and Description Logics*. 781-824
- Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, Cesare Tinelli: *Satisfiability Modulo Theories*. 825-885
- Stephen M. Majercik: *Stochastic Boolean Satisfiability*. 887-925
Special thanks are due to Armin Biere, Randy Bryant, Sam Buss, Niklas Eén, Ian Gent, Marijn Heule, Holger Hoos, Svante Janson, Peter Jeavons, Daniel Kroening, Oliver Kullmann, Massimo Lauria, Wes Pegden, Will Shortz, Carsten Sinz, Niklas Sörensson, Udo Wermuth, Ryan Williams, and ... for their detailed comments on my early attempts at exposition, as well as to numerous other correspondents who have contributed crucial corrections. Thanks also to Stanford’s Information Systems Laboratory for providing extra computer power when my laptop machine was inadequate.

*   *   *

Wow—Section 7.2.2.2 has turned out to be the longest section, by far, in The Art of Computer Programming. The SAT problem is evidently a “killer app,” because it is key to the solution of so many other problems. Consequently I can only hope that my lengthy treatment does not also kill off my faithful readers! As I wrote this material, one topic always seemed to flow naturally into another, so there was no neat way to break this section up into separate subsections. (And anyway the format of TAOCP doesn’t allow for a Section 7.2.2.2.1.)

I’ve tried to ameliorate the reader’s navigation problem by adding subheadings at the top of each right-hand page. Furthermore, as in other sections, the exercises appear in an order that roughly parallels the order in which corresponding topics are taken up in the text. Numerous cross-references are provided.
with many updated chapters and the following 7 new chapters:

**Proof Complexity** Jakob Nordström and Sam Buss

**Preprocessing** Armin Biere, Matti Järvisalo and Benjamin Kiesl

**Tuning and Configuration**
Holger Hoos, Frank Hutter and Kevin Leyton-Brown

**Proofs of Unsatisfiability** Marijn Heule

**Core-Based MaxSAT**
Fahiem Bacchus, Matti Järvisalo and Ruben Martins

**Proof Systems for Quantified Boolean Formulas**
Olaf Beyersdorff, Mikoláš Janota, Florian Lonsing and Martina Seidl

**Approximate Model Counting** Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi
Chapter 17. Fixed-Parameter Tractability

Chapter 16. Worst-Case Upper Bounds

Chapter 15. Proofs of Unsatisfiability

Chapter 14. Minimal Unsatisfiability and Autarkies

Chapter 13. Symmetry and Satisfiability

Chapter 12. Automated Configuration and Selection of SAT Solvers

Chapter 11. Exploiting Runtime Variation in Complete Solvers

Chapter 10. Random Satisfiability

Chapter 9. Preprocessing in SAT Solving

Chapter 8. Fundamentals of Branching Heuristics

Chapter 7. Proof Complexity and SAT Solving

Chapter 6. Incompleteness Algorithms

Chapter 5. Look-Ahead Based SAT Solvers

Chapter 4. Conflict-Driven Clause Learning SAT Solvers

Chapter 3. Complete Algorithms

Chapter 2. CNF Encodings

Chapter 1. A History of Satisfiability
What is Practical SAT Solving?

1st part
- encoding

2nd part
- search
- inprocessing
- other talks

Steps involved:
- Simplifying
- Reencoding
- Inprocessing
Equivalence Checking If-Then-Else Chains

original C code

```c
if(!a && !b) h();
else if(!a) g();
else f();
```

```c
if(!a)
{
  if(a) f();
  if(!b) h();
  else
  {
    else g();
    if(!b) h();
  }
else f();
} else g();
```

optimized C code

```c
if(a) f();
else if(b) g();
else h();
```

```c
if(a) f();
else if(b) g();
else h();
```

How to check that these two versions are equivalent?
Compilation

original \equiv \text{if } \lnot a \land \lnot b \text{ then } h \text{ else if } \lnot a \text{ then } g \text{ else } f
\equiv (\lnot a \land \lnot b) \land h \lor \lnot(\lnot a \land \lnot b) \land \text{if } \lnot a \text{ then } g \text{ else } f
\equiv (\lnot a \land \lnot b) \land h \lor \lnot(\lnot a \land \lnot b) \land (\lnot a \land g \lor a \land f)

optimized \equiv \text{if } a \text{ then } f \text{ else if } b \text{ then } g \text{ else } h
\equiv a \land f \lor \lnot a \land \text{if } b \text{ then } g \text{ else } h
\equiv a \land f \lor \lnot a \land (b \land g \lor \lnot b \land h)

(\lnot a \land \lnot b) \land h \lor \lnot(\lnot a \land \lnot b) \land (\lnot a \land g \lor a \land f) \not\equiv a \land f \lor \lnot a \land (b \land g \lor \lnot b \land h)

satisfying assignment gives counter-example to equivalence
Tseitin Transformation: Circuit to CNF

\[ o \wedge (x \leftrightarrow a \land c) \wedge (y \leftrightarrow b \lor x) \wedge (u \leftrightarrow a \lor b) \wedge (v \leftrightarrow b \lor c) \wedge (w \leftrightarrow u \land v) \wedge (o \leftrightarrow y \oplus w) \]

\[ o \land (x \rightarrow a) \land (x \rightarrow c) \land (x \leftarrow a \land c) \land \ldots \]

\[ o \land (\overline{x} \lor a) \land (\overline{x} \lor c) \land (x \lor \overline{a} \lor \overline{c}) \land \ldots \]
### Tseitin Transformation: Gate Constraints

**Negation:**
\[ x \leftrightarrow \bar{y} \Leftrightarrow (x \rightarrow \bar{y}) \land (\bar{y} \rightarrow x) \]
\[ \Leftrightarrow (\bar{x} \lor \bar{y}) \land (y \lor x) \]

**Disjunction:**
\[ x \leftrightarrow (y \lor z) \Leftrightarrow (y \rightarrow x) \land (z \rightarrow x) \land (x \rightarrow (y \lor z)) \]
\[ \Leftrightarrow (\bar{y} \lor x) \land (\bar{z} \lor x) \land (\bar{x} \lor y \lor z) \]

**Conjunction:**
\[ x \leftrightarrow (y \land z) \Leftrightarrow (x \rightarrow y) \land (x \rightarrow z) \land ((y \land z) \rightarrow x) \]
\[ \Leftrightarrow (\bar{x} \lor y) \land (\bar{x} \lor z) \land ((\bar{y} \land z) \lor x) \]
\[ \Leftrightarrow (\bar{x} \lor y) \land (\bar{x} \lor z) \land (\bar{y} \lor \bar{z} \lor x) \]

**Equivalence:**
\[ x \leftrightarrow (y \leftrightarrow z) \Leftrightarrow (x \rightarrow (y \leftrightarrow z)) \land ((y \leftrightarrow z) \rightarrow x) \]
\[ \Leftrightarrow (x \rightarrow ((y \rightarrow z) \land (z \rightarrow y)) \land ((y \leftrightarrow z) \rightarrow x) \]
\[ \Leftrightarrow (x \rightarrow (y \rightarrow z)) \land (x \rightarrow (z \rightarrow y)) \land ((y \leftrightarrow z) \rightarrow x) \]
\[ \Leftrightarrow (\bar{x} \lor \bar{y} \lor z) \land (\bar{x} \lor \bar{z} \lor y) \land ((\bar{y} \land z) \lor x) \]
\[ \Leftrightarrow (\bar{x} \lor \bar{y} \lor z) \land (\bar{x} \lor \bar{z} \lor y) \land ((y \land z) \lor (\bar{y} \land \bar{z})) \rightarrow x) \]
\[ \Leftrightarrow (\bar{x} \lor \bar{y} \lor z) \land (\bar{x} \lor \bar{z} \lor y) \land ((y \land z) \lor x) \land ((\bar{y} \land \bar{z}) \rightarrow x) \]
\[ \Leftrightarrow (\bar{x} \lor \bar{y} \lor z) \land (\bar{x} \lor \bar{z} \lor y) \land (\bar{y} \lor \bar{z} \lor x) \land (y \lor z \lor x) \]
Bit-Blasting of Bit-Vector Addition

addition of 4-bit numbers $x, y$ with result $s$ also 4-bit: \[ s = x + y \]

\[
[s_3, s_2, s_1, s_0]_4 = [x_3, x_2, x_1, x_0]_4 + [y_3, y_2, y_1, y_0]_4
\]

\[
[s_3, \cdot]_2 \quad = \quad \text{FullAdder}(x_3, y_3, c_2)
\]
\[
[s_2, c_2]_2 \quad = \quad \text{FullAdder}(x_2, y_2, c_1)
\]
\[
[s_1, c_1]_2 \quad = \quad \text{FullAdder}(x_1, y_1, c_0)
\]
\[
[s_0, c_0]_2 \quad = \quad \text{FullAdder}(x_0, y_0, \text{false})
\]

where

\[
[s, o]_2 \quad = \quad \text{FullAdder}(x, y, i) \quad \text{with}
\]
\[
  s \quad = \quad x \text{ xor } y \text{ xor } i
\]
\[
  o \quad = \quad (x \wedge y) \vee (x \wedge i) \vee (y \wedge i) \quad = \quad ((x + y + i) \geq 2)
\]
Boolector Architecture

O1 = bottom up simplification
O2 = global but almost linear
O3 = normalizing (often non-linear) [default]

Lingeling / PicoSAT / MiniSAT / CaDiCaL
Intermediate Representations

- encoding directly into CNF is hard, so we use intermediate levels:
  
  1. application level
  2. bit-precise semantics world-level operations (bit-vectors)
  3. bit-level representations such as And-Inverter Graphs (AIGs)
  4. conjunctive normal form (CNF)

- encoding “logical” constraints is another story
XOR as AIG

\[ \begin{align*}
\text{x xor y} & \equiv (\overline{x} \land y) \lor (x \land \overline{y}) \\
& \equiv (\overline{x} \land y) \land (x \land \overline{y})
\end{align*} \]
4-bit adder

8-bit adder
bit-vector of length 16 shifted by bit-vector of length 4
Encoding Logical Constraints

- Tseitin construction suitable for most kinds of “model constraints”
  - assuming simple operational semantics: encode an interpreter
  - small domains: one-hot encoding  large domains: binary encoding

- harder to encode properties or additional constraints
  - temporal logic / fix-points
  - environment constraints

- example for fix-points / recursive equations: \( x = (a \lor y), \quad y = (b \lor x) \)
  - has unique least fix-point \( x = y = (a \lor b) \)
  - and unique largest fix-point \( x = y = true \)  but unfortunately …
  - … only largest fix-point can be (directly) encoded in SAT
    otherwise need stable models / logical programming / ASP
Example of Logical Constraints: Cardinality Constraints

- given a set of literals \( \{l_1, \ldots, l_n\} \)
  - constraint the number of literals assigned to \textit{true}
  - \( l_1 + \cdots + l_n \geq k \) or \( l_1 + \cdots + l_n \leq k \) or \( l_1 + \cdots + l_n = k \)
  - combined make up exactly all fully symmetric boolean functions

- multiple encodings of cardinality constraints
  - naïve encoding exponential: \textit{at-most-one} quadratic, \textit{at-most-two} cubic, etc.
  - quadratic \( O(k \cdot n) \) encoding goes back to Shannon
  - linear \( O(n) \) parallel counter encoding \[\text{[Sinz'05]}\]

- many variants even for \textit{at-most-one} constraints
  - for an \( O(n \cdot \log n) \) encoding see Prestwich’s chapter in Handbook of SAT

- \textbf{Pseudo-Boolean} constraints (PB) or 0/1 ILP constraints have many encodings too

\[
2 \cdot \overline{a} + b + c + d + 2 \cdot e \geq 3
\]

actually used to handle MaxSAT in SAT4J for configuration in Eclipse
BDD-Based Encoding of Cardinality Constraints

\[ 2 \leq l_1 + \cdots + l_9 \leq 3 \]

If-Then-Else gates (MUX) with “then” edge downward, dashed “else” edge to the right
Tseitin Encoding of If-Then-Else Gate

\[ x \leftrightarrow (c \ ? \ t \ : \ e) \iff (x \rightarrow (c \rightarrow t)) \land (x \rightarrow (\bar{c} \rightarrow e)) \land (\bar{x} \rightarrow (c \rightarrow \bar{t})) \land (\bar{x} \rightarrow (\bar{c} \rightarrow \bar{e})) \]
\[ \iff (\bar{x} \lor \bar{c} \lor t) \land (\bar{x} \lor c \lor e) \land (x \lor \bar{c} \lor \bar{t}) \land (x \lor c \lor \bar{e}) \]

minimal but not arc consistent:

- if \( t \) and \( e \) have the same value then \( x \) needs to have that too
- possible additional clauses

\( (\bar{t} \land \bar{e} \rightarrow \bar{x}) \equiv (t \lor e \lor \bar{x}) \)
\( (t \land e \rightarrow x) \equiv (\bar{t} \lor \bar{e} \lor x) \)

- but can be learned or derived through preprocessing (ternary resolution) keeping those clauses redundant is better in practice
$ cat example.cnf

c comments start with 'c' and extend until the end of the line
c
c variables are encoded as integers:
c
c 'tie' becomes '1'
c 'shirt' becomes '2'
c
c header 'p cnf <variables> <clauses>'
c
p cnf 2 3
-1 2 0 c !tie or shirt
1 2 0 c tie or shirt
-1 -2 0 c !tie or !shirt

$ picosat example.cnf
s SATISFIABLE
v -1 2 0
SAT Application Programmatic Interface (API)

- incremental usage of SAT solvers
  - add facts such as clauses incrementally
  - call SAT solver and get satisfying assignments
  - optionally retract facts

- retracting facts
  - remove clauses explicitly: complex to implement
  - push / pop: stack like activation, no sharing of learned facts
  - MiniSAT assumptions [EénSörensson’03]

- assumptions
  - unit assumptions: assumed for the next SAT call
  - easy to implement: force SAT solver to decide on assumptions first
  - shares learned clauses across SAT calls

- IPASIR: Reentrant Incremental SAT API
  - used in the SAT competition / race since 2015 [BalyoBiereIserSinz’16]
```c
#include "ipasir.h"
#include <assert.h>
#include <stdio.h>
#define ADD(LIT) ipasir_add (solver, LIT)
#define PRINT(LIT) \
  printf (ipasir_val (solver, LIT) < 0 ? " -" #LIT : " " #LIT)
int main () {
  void * solver = ipasir_init ();
  enum { tie = 1, shirt = 2 };
  ADD (-tie); ADD ( shirt); ADD (0);
  ADD ( tie); ADD ( shirt); ADD (0);
  ADD (-tie); ADD (-shirt); ADD (0);
  int res = ipasir_solve (solver);
  assert (res == 10);
  printf ("satisfiable:"); PRINT (shirt); PRINT (tie); printf ("
") ;
  printf ("assuming now: tie shirt\n" );
  ipasir_assume (solver, tie); ipasir_assume (solver, shirt);
  res = ipasir_solve (solver);
  assert (res == 20);
  printf ("unsatisfiable, failed:");
  if (ipasir_failed (solver, tie)) printf (" tie");
  if (ipasir_failed (solver, shirt)) printf (" shirt");
  printf ("\n");
  ipasir_release (solver);
  return res;
}
```

```
$ ./example
satisfiable: shirt -tie
assuming now: tie shirt
unsatisfiable, failed: tie
```
IPASIR Functions

const char * ipasir_signature ();

void * ipasir_init ();

void ipasir_release (void * solver);

void ipasir_add (void * solver, int lit_or_zero);

void ipasir_assume (void * solver, int lit);

int ipasir_solve (void * solver);

int ipasir_val (void * solver, int lit);

int ipasir_failed (void * solver, int lit);

void ipasir_set_terminate (void * solver, void * state,
                           int (*terminate)(void * state));
```cpp
#include "cadical.hpp"
#include <cassert>
#include <iostream>
using namespace std;
#define ADD(LIT) solver.add (LIT)
#define PRINT(LIT)
  (solver.val (LIT) < 0 ? " -" #LIT : " " #LIT)

int main () {
  CaDiCaL::Solver solver; solver.set ("quiet", 1);
  enum { tie = 1, shirt = 2 };
  ADD (-tie), ADD ( shirt), ADD (0);
  ADD ( tie), ADD ( shirt), ADD (0);
  ADD (-tie), ADD (-shirt), ADD (0);
  int res = solver.solve ();
  assert (res == 10);
  cout << "satisfiable:" << PRINT (shirt) << PRINT (tie) << endl;
  cout << "assuming now: tie shirt" << endl;
  solver.assume (tie), solver.assume (shirt);
  res = solver.solve ();
  assert (res == 20);
  cout << "unsatisfiable, failed:";
  if (solver.failed (tie)) cout << " tie";
  if (solver.failed (shirt)) cout << " shirt";
  cout << endl;
  return res;
}
```

$ ./example
satisfiable: shirt -tie
assuming now: tie shirt
unsatisfiable, failed: tie

DP / DPLL

- dates back to the 50’ies:
  - 1st version DP is resolution based ⇒ preprocessing
  - 2nd version D(P)LL splits space for time ⇒ CDCL

- ideas:
  - 1st version: eliminate the two cases of assigning a variable in space or
  - 2nd version: case analysis in time, e.g. try \( x = 0,1 \) in turn and recurse

- most successful SAT solvers are based on variant (CDCL) of the second version
  works for very large instances

- recent (≤ 25 years) optimizations:
  backjumping, learning, UIPs, dynamic splitting heuristics, fast data structures
DP Procedure

forever

if $F = \top$ return satisfiable

if $\bot \in F$ return unsatisfiable

pick remaining variable $x$

add all resolvents on $x$

remove all clauses with $x$ and $\neg x$

$\Rightarrow$ Bounded Variable Elimination
D(P)LL Procedure

\[ DPLL(F) \]

\[ F := BCP(F) \]

if \( F = \top \) return **satisfiable**

if \( \bot \in F \) return **unsatisfiable**

pick remaining variable \( x \) and literal \( l \in \{x, \neg x\} \)

if \( DPLL(F \land \{l\}) \) returns **satisfiable** return **satisfiable**

return \( DPLL(F \land \{\neg l\}) \)

⇒ **CDCL**
DPLL Example

\[
\begin{align*}
\neg a \lor \neg b \lor \neg c & \\
\neg a \lor b \lor \neg c & \\
\neg a \lor \neg b \lor c & \\
\neg a \lor b \lor \neg c & \\
a \lor \neg b \lor \neg c & \\
a \lor b \lor \neg c & \\
a \lor \neg b \lor c & \\
a \lor b \lor c & \\
\end{align*}
\]
Conflict Driven Clause Learning (CDCL)  
[MarqueSilvaSakallah'96]

- first implemented in the context of GRASP SAT solver
  - name given later to distinguish it from DPLL
  - not recursive anymore

- essential for SMT

- learning clauses as no-goods

- notion of implication graph

- (first) unique implication points
Conflict Driven Clause Learning (CDCL)

\[ a = 1 \]
\[ b = 1 \]
\[ c = 0 \]

BCP

\[ \neg c \]

decision

\[ a \]

\[ \neg a \lor \neg b \lor \neg c \]

\[ \neg a \lor \neg b \lor c \]

\[ \neg a \lor b \lor c \]

\[ a \lor \neg b \lor c \]

\[ a \lor b \lor c \]

\[ a \lor b \lor c \]

\[ a \lor b \lor c \]

\[ a \lor b \lor c \]

\[ \neg a \lor \neg b \]

learn
Conflict Driven Clause Learning (CDCL)

$a = 1$

$b = 0$

$c = 0$

Decision $a$

$\neg b$ BCP

$\neg c$ BCP

Clauses

$\neg a \lor \neg b \lor \neg c$

$\neg a \lor \neg b \lor c$

$\neg a \lor \neg b \lor c$

$a \lor \neg b \lor c$

$a \lor b \lor \neg c$

$a \lor b \lor c$

$a \lor \neg c$

learn

$\neg a$
Conflict Driven Clause Learning (CDCL)

\[ a = 1 \]
\[ b = 0 \]
\[ c = 0 \]

\[ \neg a \quad \neg c \quad \text{decision} \]

\[ \neg b \quad \text{BCP} \]

\[ \neg a \lor \neg b \lor \neg c \]
\[ \neg a \lor \neg b \lor c \]
\[ \neg a \lor b \lor \neg c \]
\[ \neg a \lor b \lor c \]
\[ a \lor \neg b \lor \neg c \]
\[ a \lor b \lor c \]
\[ \neg a \lor \neg b \]
\[ \neg a \]

\[ c \]

\[ \text{clauses} \]
Conflict Driven Clause Learning (CDCL)

\[ a = 1 \]
\[ b = 0 \]
\[ c = 0 \]

\[ \neg a \lor \neg b \lor \neg c \]
\[ \neg a \lor \neg b \lor c \]
\[ \neg a \lor b \lor \neg c \]
\[ \neg a \lor b \lor c \]
\[ \neg a \lor \neg b \]
\[ \neg a \]

learn
empty clause

clauses

\[ \neg a \lor \neg b \lor \neg c \]
\[ \neg a \lor \neg b \lor c \]
\[ \neg a \lor b \lor \neg c \]
\[ \neg a \lor b \lor c \]
Implication Graph

d = 1 @ 1
e = 1 @ 1
b = 1 @ 0
a = 1 @ 0
f g = 1 @ 2
h = 1 @ 2
i = 1 @ 2
l = 1 @ 3
c
k = 1 @ 3
r = 1 @ 4
s = 1 @ 4
t = 1 @ 4
y = 1 @ 4
x = 1 @ 4
z = 1 @ 4
κ

Top-level decision

Unit

Conflicts
Antecedents / Reasons

$d \land g \land s \rightarrow t \equiv (\bar{d} \lor \bar{g} \lor \bar{s} \lor t)$
Conflicting Clauses

\[ \neg(y \land z) \equiv (\bar{y} \lor \bar{z}) \]
Resolving Antecedents 1\textsuperscript{st} Time

\[
\begin{align*}
\text{top-level} & \quad \text{unit} \quad a = 1 @ 0 \quad \text{unit} \quad b = 1 @ 0 \\
\text{decision} & \quad c = 1 @ 1 \quad d = 1 @ 1 \quad e = 1 @ 1 \\
\text{decision} & \quad f = 1 @ 2 \quad g = 1 @ 2 \\
\text{decision} & \quad k = 1 @ 3 \quad l = 1 @ 3 \\
\text{decision} & \quad r = 1 @ 4 \quad s = 1 @ 4 \\
\quad x = 1 @ 4 \\
\quad z = 1 @ 4 \quad \kappa \quad \text{conflict}
\end{align*}
\]

\[
(\overline{h} \vee i \vee \overline{i} \vee y) \quad (\overline{y} \vee \overline{z})
\]
Resolving Antecedents 1\textsuperscript{st} Time

\[ (\bar{h} \lor \bar{i} \lor \bar{j} \lor y) \land (\bar{y} \lor \bar{z}) \]

\[ (\bar{h} \lor \bar{i} \lor \bar{j} \lor \bar{z}) \]
Resolvents = **Cuts** = Potential Learned Clauses

$$\top \rightarrow a = 1 @ 0 \rightarrow b = 1 @ 0$$

$$\text{decision } c = 1 @ 1 \rightarrow d = 1 @ 1 \rightarrow e = 1 @ 1$$

$$\text{decision } f = 1 @ 2 \rightarrow g = 1 @ 2$$

$$\text{decision } k = 1 @ 3 \rightarrow l = 1 @ 3$$

$$\text{decision } r = 1 @ 4 \rightarrow s = 1 @ 4$$

$$x = 1 @ 4 \rightarrow z = 1 @ 4 \rightarrow \kappa$$

$$\frac{(\overline{h} \lor \overline{i} \lor \overline{t} \lor y)}{(\overline{y} \lor \overline{z})}$$

$$\text{conflict}$$
Potential Learned Clause After 1 Resolution

\[
\begin{align*}
\text{top–level} & \quad \text{unit} \quad a = 1 @ 0 & \quad \text{unit} \quad b = 1 @ 0 \\
\text{decision} & \quad c = 1 @ 1 & \quad d = 1 @ 1 & \quad e = 1 @ 1 \\
\text{decision} & \quad f = 1 @ 2 & \quad g = 1 @ 2 & \quad h = 1 @ 2 & \quad i = 1 @ 2 \\
\text{decision} & \quad k = 1 @ 3 & \quad l = 1 @ 3 \\
\text{decision} & \quad r = 1 @ 4 & \quad s = 1 @ 4 & \quad t = 1 @ 4 & \quad y = 1 @ 4 \\
\quad x = 1 @ 4 & \quad z = 1 @ 4 & \quad \kappa \quad \text{conflict} \\
\end{align*}
\]

\[\overline{h} \lor \overline{i} \lor \overline{t} \lor \overline{z}\]
Resolving Antecedents 2nd Time

\[
(d \lor \bar{g} \lor \bar{s} \lor t) (\bar{h} \lor \bar{i} \lor \bar{t} \lor \bar{z})
\]

\[
(d \lor \bar{g} \lor \bar{s} \lor \bar{h} \lor \bar{i} \lor \bar{t} \lor \bar{z})
\]

\[
(d \lor \bar{g} \lor \bar{s} \lor \bar{h} \lor \bar{i} \lor \bar{t} \lor \bar{z})
\]
Resolving Antecedents 3rd Time

\[(\overline{x} \lor z) (d \lor \overline{g} \lor \overline{s} \lor \overline{h} \lor \overline{i} \lor \overline{z})\]

\[(\overline{x} \lor \overline{d} \lor \overline{g} \lor \overline{s} \lor \overline{h} \lor \overline{i})\]
Resolving Antecedents 4\textsuperscript{th} Time

<table>
<thead>
<tr>
<th>Decision</th>
<th>( a = 1 @ 0 )</th>
<th>( b = 1 @ 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( d = 1 @ 1 )</td>
<td>( e = 1 @ 1 )</td>
</tr>
<tr>
<td></td>
<td>( g = 1 @ 2 )</td>
<td>( h = 1 @ 2 )</td>
</tr>
<tr>
<td></td>
<td>( s = 1 @ 4 )</td>
<td>( t = 1 @ 4 )</td>
</tr>
<tr>
<td></td>
<td>( x = 1 @ 4 )</td>
<td>( z = 1 @ 4 )</td>
</tr>
</tbody>
</table>

\[
(\overline{s} \lor x) \quad (\overline{x} \lor \overline{d} \lor \overline{g} \lor \overline{s} \lor \overline{h} \lor i) \\
(\overline{d} \lor \overline{g} \lor \overline{s} \lor \overline{h} \lor i)
\]

self subsuming resolution
1st UIP Clause after 4 Resolutions

\[(d \lor g \lor s \lor h \lor i)\]

UIP = unique implication point dominates conflict on the last level
If $y$ has never been used to derive a conflict, then skip $\bar{y}$ case.

Immediately jump back to the $\bar{x}$ case – assuming $x$ was used.
Resolving Antecedents 5^{th} Time

\begin{align*}
\text{top-level} & \quad \text{unit} \quad a = 1 @ 0 \quad \text{unit} \quad b = 1 @ 0 \\
\text{decision} \quad c = 1 @ 1 & \quad d = 1 @ 1 & \quad e = 1 @ 1 \\
\text{decision} \quad f = 1 @ 2 & \quad g = 1 @ 2 & \quad h = 1 @ 2 & \quad i = 1 @ 2 \\
\text{decision} \quad k = 1 @ 3 & \quad l = 1 @ 3 \\
\text{decision} \quad r = 1 @ 4 & \quad s = 1 @ 4 & \quad t = 1 @ 4 & \quad y = 1 @ 4 \\
\end{align*}

\[ (\bar{l} \lor \bar{r} \lor s) \quad (\bar{d} \lor \bar{g} \lor \bar{s} \lor \bar{h} \lor \bar{i}) \quad (\bar{l} \lor \bar{r} \lor \bar{d} \lor \bar{g} \lor \bar{h} \lor \bar{i}) \]

\[ \kappa \quad \text{conflict} \]
(\overline{d} \lor \overline{g} \lor \overline{l} \lor \overline{r} \lor \overline{h} \lor \overline{i})
1st UIP Clause after 4 Resolutions

decision $c = 1 @ 1$ \quad \rightarrow \quad d = 1 @ 1 \quad \rightarrow \quad e = 1 @ 1$

decision $f = 1 @ 2$ \quad \rightarrow \quad g = 1 @ 2 \quad \rightarrow \quad h = 1 @ 2 \quad \rightarrow \quad i = 1 @ 2$

decision $k = 1 @ 3$ \quad \rightarrow \quad l = 1 @ 3$

decision $r = 1 @ 4$ \quad \rightarrow \quad s = 1 @ 4 \quad \rightarrow \quad t = 1 @ 4 \quad \rightarrow \quad y = 1 @ 4$

$x = 1 @ 4 \quad \rightarrow \quad z = 1 @ 4 \quad \rightarrow \quad \kappa$ conflict

$(\overline{d} \lor \overline{g} \lor \overline{s} \lor \overline{h} \lor \overline{i})$
Locally Minimizing 1st UIP Clause

\[
\begin{align*}
\text{top-level} & \quad a = 1 @ 0 \quad \text{unit} \quad b = 1 @ 0 \\
\text{decision} & \quad c = 1 @ 1 \quad \rightarrow \quad d = 1 @ 1 \quad \rightarrow \quad e = 1 @ 1 \\
\text{decision} & \quad f = 1 @ 2 \quad \rightarrow \quad g = 1 @ 2 \quad \rightarrow \quad h = 1 @ 2 \quad \rightarrow \quad i = 1 @ 2 \\
\text{decision} & \quad k = 1 @ 3 \quad \rightarrow \quad l = 1 @ 3 \\
\text{decision} & \quad r = 1 @ 4 \quad \rightarrow \quad s = 1 @ 4 \quad \rightarrow \quad t = 1 @ 4 \quad \rightarrow \quad y = 1 @ 4 \\
\quad x = 1 @ 4 \quad \rightarrow \quad z = 1 @ 4 \quad \rightarrow \quad \kappa \quad \text{conflict}
\end{align*}
\]

\[
\frac{(h \lor i) \quad (d \lor \overline{g} \lor \overline{s} \lor \overline{h} \lor i)}{(d \lor \overline{g} \lor \overline{s} \lor \overline{h})}
\]

self subsuming resolution
Locally Minimized Learned Clause

\[
(d \lor \overline{g} \lor \overline{s} \lor \overline{h})
\]
Minimizing Locally Minimized Learned Clause Further?

\[(\overline{d} \lor \overline{g} \lor \overline{s} \lor \overline{h})\]
Recursively Minimizing Learned Clause

\[
\begin{align*}
\top & \quad a = 1 \quad \text{(top-level decision)} \\
\text{decision} & \quad c = 1 \quad \text{(decision)} \\
\text{decision} & \quad f = 1 \quad \text{(decision)} \\
\text{decision} & \quad k = 1 \quad \text{(decision)} \\
\text{decision} & \quad r = 1 \quad \text{(decision)} \\
\text{unit} & \quad b = 1 \quad \text{(unit)} \\
\text{unit} & \quad d = 1 \quad \text{(unit)} \\
\text{unit} & \quad e = 1 \quad \text{(unit)} \\
\text{unit} & \quad g = 1 \quad \text{(unit)} \\
\text{unit} & \quad h = 1 \quad \text{(unit)} \\
\text{unit} & \quad i = 1 \quad \text{(unit)} \\
\text{unit} & \quad l = 1 \quad \text{(unit)} \\
\text{unit} & \quad s = 1 \quad \text{(unit)} \\
\text{unit} & \quad t = 1 \quad \text{(unit)} \\
\text{unit} & \quad y = 1 \quad \text{(unit)} \\
\text{unit} & \quad z = 1 \quad \text{(unit)} \\
\text{unit} & \quad x = 1 \quad \text{(unit)} \\
\text{unit} & \quad \kappa = 1 \quad \text{(conflict)} \\
\nonumber
\end{align*}
\]

\[
\begin{align*}
(y \lor g \lor h) & \quad (\overline{d} \lor \overline{g} \lor \overline{s} \lor h) \\
(d \lor \overline{g} \lor \overline{s} \lor h) & \quad (\overline{e} \lor \overline{d} \lor \overline{g} \lor \overline{s}) \\
(e \lor \overline{d} \lor \overline{g} \lor \overline{s}) & \quad (\overline{d} \lor \overline{g} \lor \overline{s}) \\
(b) & \quad (b \lor \overline{d} \lor \overline{g} \lor \overline{s}) \\
\nonumber
\end{align*}
\]
Recursively Minimized Learned Clause

\[(d \vee \overline{g} \vee s)\]
Decision Heuristics

- number of variable occurrences in (remaining unsatisfied) clauses (LIS)
  - eagerly satisfy many clauses with many variations studied in the 90ies
  - actually expensive to compute

- dynamic heuristics
  - focus on variables which were useful recently in deriving learned clauses
  - can be interpreted as reinforcement learning
  - started with the VSIDS heuristic
    - [MoskewiczMadiganZhaoZhangMalik'01]
  - most solvers rely on the exponential variant in MiniSAT (EVSIDS)
  - recently showed VMTF as effective as VSIDS
    - [BiereFröhlich-SAT’15] survey

- look-ahead
  - spent more time in selecting good variables (and simplification)
  - related to our Cube & Conquer paper
    - [HeuleKullmanWieringaBiere-HVC’11]
  - “The Science of Brute Force”
    - [Heule & Kullman CACM August 2017]

- EVSIDS during stabilization VMTF otherwise
  - [Biere-SAT-Race-2019]
Fast VMTF Implementation

- Siege SAT solver [Ryan Thesis 2004] used variable move to front (VMTF)
  - bumped variables moved to head of doubly linked list
  - search for unassigned variable starts at head
  - variable selection is an online sorting algorithm of scores
  - classic “move-to-front” strategy achieves good amortized complexity

- fast simple implementation for caching searches in VMTF [BiereFröhlich’SAT15]
  - doubly linked list does not have positions as an ordered array
  - bump = move-to-front = dequeue then insertion at the head

- time-stamp list entries with “insertion-time”
  - maintained invariant: all variables right of next-search are assigned
  - requires (constant time) update to next-search while unassigning variables
  - occasionally (32-bit) time-stamps will overflow: update all time stamps
# Variable Scoring Schemes

[BiereFröhlich-SAT’15]

<table>
<thead>
<tr>
<th>variable score $s'$ after $i$ conflicts</th>
<th>bumped</th>
<th>not-bumped</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STATIC</strong></td>
<td>$s$</td>
<td>$s$</td>
</tr>
<tr>
<td><strong>INC</strong></td>
<td>$s + 1$</td>
<td>$s$</td>
</tr>
<tr>
<td><strong>SUM</strong></td>
<td>$s + i$</td>
<td>$s$</td>
</tr>
<tr>
<td><strong>VSIDS</strong></td>
<td>$h_i^{256} \cdot s + 1$</td>
<td>$h_i^{256} \cdot s$</td>
</tr>
<tr>
<td><strong>NVSIDS</strong></td>
<td>$f \cdot s + (1 - f)$</td>
<td>$f \cdot s$</td>
</tr>
<tr>
<td><strong>EVSIDS</strong></td>
<td>$s + g^i$</td>
<td>$s$</td>
</tr>
<tr>
<td><strong>ACIDS</strong></td>
<td>$(s + i)/2$</td>
<td>$s$</td>
</tr>
<tr>
<td><strong>VMTF$_1$</strong></td>
<td>$i$</td>
<td>$s$</td>
</tr>
<tr>
<td><strong>VMTF$_2$</strong></td>
<td>$b$</td>
<td>$s$</td>
</tr>
</tbody>
</table>

- **STATIC**: static decision order
- **INC**: increment scores
- **SUM**: sum of conflict-indices
- **VSIDS**: original implementation in Chaff
- **NVSIDS**: normalized variant of VSIDS
- **EVSIDS**: exponential MiniSAT dual of NVSIDS
- **ACIDS**: average conflict-index decision scheme
- **VMTF$_1$**: variable move-to-front
- **VMTF$_2$**: variable move-to-front variant

<table>
<thead>
<tr>
<th>$0 &lt; f &lt; 1$</th>
<th>$g = 1/f$</th>
<th>$h_i^m = 0.5$ if $m$ divides $i$</th>
<th>$h_i^m = 1$ otherwise</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>$b$</td>
<td>conflict index</td>
<td>bumped counter</td>
</tr>
</tbody>
</table>
Basic CDCL Loop

```c
int basic_cdcl_loop () {
  int res = 0;

  while (!res)
    if (unsat) res = 20;  // analyze propagated conflict
    else if (!propagate ()) analyze ();         // all variables satisfied
    else if (satisfied ()) res = 10;            // otherwise pick next decision
    else decide ();

  return res;
}
```
Reducing Learned Clauses

- keeping all learned clauses slows down BCP
  - so SATO and RelSAT just kept only “short” clauses

- better periodically delete “useless” learned clauses
  - keep a certain number of learned clauses
  - if this number is reached MiniSAT reduces (deletes) half of the clauses
  - then maximum number kept learned clauses is increased geometrically

- LBD (glucose level / glue) prediction for usefulness
  - LBD = number of decision-levels in the learned clause
  - allows arithmetic increase of number of kept learned clauses
  - keep clauses with small LBD forever \( \leq 2 \ldots 5 \)
  - three Tier system by

- eagerly reduce hyper-binary resolvents derived in inprocessing
Restarts

- often it is a good strategy to abandon what you do and restart
  - for satisfiable instances the solver may get stuck in the unsatisfiable part
  - for unsatisfiable instances focusing on one part might miss short proofs
  - restart after the number of conflicts reached a restart limit

- avoid to run into the same dead end
  - by randomization (either on the decision variable or its phase)
  - and/or just keep all the learned clauses during restart

- for completeness dynamically increase restart limit
  - arithmetically, geometrically, Luby, Inner/Outer

- **Glucose restarts** [AudemardSimon-CP’12]
  - short vs. large window exponential moving average (EMA) over LBD
  - if recent LBD values are larger than long time average then restart

- interleave “stabilizing” (no restarts) and “non-stabilizing” phases [Chanseok Oh]
  - call it now “stabilizing mode” and “focused mode”
Luby’s Restart Intervals
70 restarts in 104448 conflicts
Luby Restart Scheduling

```c
unsigned luby (unsigned i)
{
    unsigned k;

    for (k = 1; k < 32; k++)
        if (i == (1 << k) - 1)
            return 1 << (k - 1);

    for (k = 1;; k++)
        if ((1 << (k - 1)) <= i && i < (1 << k) - 1)
            return luby (i - (1 << (k-1)) + 1);
}

limit = 512 * luby (++restarts);
... // run SAT core loop for 'limit' conflicts
```
(u_1, v_1) = (1, 1)

(u_{n+1}, v_{n+1}) = ((u_n \& -u_n == v_n) ? (u_{n} + 1, 1) : (u_n, 2v_n))

(1, 1), (2, 1), (2, 2), (3, 1), (4, 1), (4, 2), (4, 4), (5, 1), ...
Restart Scheduling with Exponential Moving Averages

[BiereFröhlich-POS’15]

- **LBD**
  - fast $EMA$ of LBD with $\alpha = 2^{-5}$
- **restart**
  - slow $EMA$ of LBD with $\alpha = 2^{-14}$ (ema-14)
- **inprocessing**
  - $CMA$ of LBD (average)
Phase Saving and Rapid Restarts

- phase assignment:
  - assign decision variable to 0 or 1?
  - "Only thing that matters in satisfiable instances" [Hans van Maaren]

- "phase saving" as in RSat [PipatsrisawatDarwiche’07]
  - pick phase of last assignment (if not forced to, do not toggle assignment)
  - initially use statically computed phase (typically LIS)
  - so can be seen to maintain a global full assignment

- rapid restarts
  - varying restart interval with bursts of restarts
  - not only theoretically avoids local minima
  - works nicely together with phase saving

- reusing the trail can reduce the cost of restarts [RamosVanDerTakHeule-JSAT’11]

- target phases of largest conflict free trail / assignment
  [Biere-SAT-Race-2019] [BiereFleury-POS-2020] [CaiZhang-SAT21] [CaiZhangFleuryBiere-JAIR22]
int basic_cdcl_loop_with_reduce_and_restart () {

  int res = 0;

  while (!res)
  {
    if (unsat) res = 20;  // analyze propagated conflict
    else if (!propagate ()) analyze ();  // analyze propagated conflict
    else if (satisfied ()) res = 10;  // all variables satisfied
    else if (restarting ()) restart ();  // restart by backtracking
    else if (reducing ()) reduce ();  // collect useless learned clauses
    else decide ();  // otherwise pick next decision
  }

  return res;
}
while (!res) {
    if (unsat) res = 20;
    else if (!propagate ()) analyze (); // propagate and analyze
    else if (iterating) iterate ();     // report learned unit
    else if (satisfied ()) res = 10;   // found model
    else if (search_limits_hit ()) break; // decision or conflict limit
    else if (terminated_asynchronously ()) // externally terminated
        break;
    else if (restarting ()) restart ();  // restart by backtracking
    else if (rephasing ()) rephase ();  // reset variable phases
    else if (reducing ()) reduce ();    // collect useless clauses
    else if (probing ()) probe ();      // failed literal probing
    else if (subsuming ()) subsume ();  // subsumption algorithm
    else if (eliminating ()) elim ();   // variable elimination
    else if (compacting ()) compact (); // collect variables
    else if (conditioning ()) condition (); // globally blocked clauses
    else res = decide ();               // next decision
}

https://github.com/arminbiere/cadical
https://fmv.jku.at/cadical
Two-Watched Literal Schemes

- original idea from SATO
  - invariant: always watch two non-false literals
  - if a watched literal becomes false replace it
  - if no replacement can be found clause is either unit or empty
  - original version used head and tail pointers on Tries

- improved variant from Chaff
  - watch pointers can move arbitrarily
  - no update needed during backtracking
  - one watch is enough to ensure correctness
    - but loses arc consistency

- reduces visiting clauses by 10x
  - particularly useful for large and many learned clauses

- blocking literals

- special treatment of short clauses (binary [PilarskiHu'02] or ternary [Ryan'04])

- cache start of search for replacement
Parallel SAT

- vector units, GPU, multi-core, cluster, cloud
- application level parallelism usually trivial
- classic work on guiding path principle
- portfolio (with sharing)
- (concurrent) cube & conquer
- control vs. data flow parallelism
- achieve low-level parallelism even though even already BCP is P-complete

⇒ Handbook of Parallel Constraint Reasoning

⇒ still many low-level programming issues left
resolution proofs (RES) are simple to check but large and hard(er) to produce directly

original idea for clausal proofs and checking them:
- proof traces are sequences of “learned clauses” \( C \)
- first check clause through unit propagation \( F \vdash_1 C \) then add \( C \) to \( F \)
- reverse unit implied clauses (RUP) \([GoldbergNovikov’03] [VanGelder’12]\)
- deletion information:
  - “deletion” lines tell checker to forget clause, decreases checking time substantially
- trace of added and deleted clauses (DRUP) \([HeuleHuntWetzler-FMCAD’13 / STVR’14]\)

RUP/RES tracks SAT Competition 2007, 2009, 2011, now DRUP/DRAT mandatory since 2013 to certify UNSAT

big certified proofs:
- Pythagorean Triples \([HeuleKullmannMarek-SAT’16]\) (200TB)
- Schur Number Five \([Heule-AAAI’18]\) (2PB)
- Certification: Coq \([CruzFilipeMarquesSilvaSchneiderKamp-TACAS’17 / JAR’19]\), similar papers for ACL2, Isabelle, …
<table>
<thead>
<tr>
<th>CNF</th>
<th>trace</th>
<th>extended trace</th>
<th>resolution trace</th>
<th>RUP</th>
<th>DRUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1 -2 -3 0</td>
<td>1 -2 -3 -1 0 0</td>
<td>1 -2 -3 -1 0 0</td>
<td>1 -1 -3 -2 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1 -2 3 0</td>
<td>2 -2 3 -1 0 0</td>
<td>2 -2 3 -1 0 0</td>
<td>2 -1 3 -2 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1 2 -3 0</td>
<td>3 2 -3 -1 0 0</td>
<td>3 2 -3 -1 0 0</td>
<td>3 2 -1 -3 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1 2 3 0</td>
<td>4 2 3 -1 0 0</td>
<td>4 2 3 -1 0 0</td>
<td>4 2 -1 3 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 -2 -3 0</td>
<td>5 1 -3 -2 0 0</td>
<td>5 1 -3 -2 0 0</td>
<td>5 -2 -3 1 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 -2 3 0</td>
<td>6 1 3 -2 0 0</td>
<td>6 1 3 -2 0 0</td>
<td>6 -2 3 1 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 2 -3 0</td>
<td>7 1 -3 2 0 0</td>
<td>7 1 -3 2 0 0</td>
<td>7 1 -3 2 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 2 3 0</td>
<td>8 1 3 2 0 0</td>
<td>8 1 3 2 0 0</td>
<td>8 1 3 2 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 * 7 8 0</td>
<td>9 1 2 0 7 8 0</td>
<td>9 1 2 0 7 8 0</td>
<td>-2 -3 0 -2 -3 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 * 9 5 6 0</td>
<td>10 1 0 9 5 6 0</td>
<td>10 -2 1 0 5 6 0</td>
<td>-3 0 d 1 -2 -3 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 * 1 10 2 0</td>
<td>11 -2 0 1 10 2 0</td>
<td>11 1 0 10 9 0</td>
<td>2 0 d -1 -2 -3 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 * 10 11 4 0</td>
<td>12 3 0 10 11 4 0</td>
<td>12 -1 -2 0 1 2 0</td>
<td>-1 0 -2 3 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 * 10 11 3 12 0</td>
<td>13 0 10 11 3 12 0</td>
<td>13 -2 0 12 11 0</td>
<td>0 d 1 -2 3 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

picosat -t    picosat -T    tracecheck -B    cadical    cadical -P1
Definition. Clause $C$ blocked on literal $\ell \in C$ w.r.t CNF $F$ if for all resolution candidates $D \in F$ with $\bar{\ell} \in D$ the resolvent $(C \setminus \ell) \lor (D \setminus \bar{\ell})$ is tautological.

Assume output true, thus single unit clause constraint $(x)$

PG encoding drops upward propagating clauses of only positively occurring gates.
PG encoding drops downward propagating clauses of only negatively occurring gates.

Unconstrained or monotone inputs can be removed too.
Resolution Asymmetric Tautologies (RAT)

"Inprocessing Rules"  [JärvisaloHeuleBiere-IJCAR’12]

- justify complex preprocessing algorithms in Lingeling  [Biere-TR’10]
  - examples are adding blocked clauses or variable elimination
  - interleaved with research (forgetting learned clauses = reduce)

- need more general notion of redundancy criteria
  - extension of blocked clauses
  - replace “resolvents on \( l \) are tautological” by “resolvents on \( l \) are RUP”

  example:  \((a \lor \lnot l)\)  RAT on \( l \)  w.r.t.  \((a \lor b) \land (l \lor c) \land (\lnot l \lor b)\)

- deletion information is again essential (DRAT)  [HeuleHuntWetzler-FMCAD’13 / STVR’14]
- now mandatory in the main track of the SAT competitions since 2013
- pretty powerful:  can for instance also cover symmetry breaking
"Clause Elimination for SAT and QSAT"

by Marijn Heule, Matti Järvisalo, Florian Lonsing, Martina Seidl and Armin Biere

has been selected as the winner of the

2019 IJCAI-JAIR Best Paper Prize

with the following citation:

This paper describes fundamental and practical results on a range of clause elimination procedures as preprocessing and simplification techniques for SAT and QBF solvers. Since its publication, the techniques described therein have been demonstrated to have profound impact on the efficiency of state-of-the-art SAT and QBF solvers. The work is elegant and extends beautifully some well-established theoretical concepts. In addition, the paper gives new emphasis and impulse to pre- and in-processing techniques - an emphasis that resonates beyond the two key problems, SAT and QBF, covered by the authors.

The IJCAI-JAIR Best Paper Prize is awarded to an outstanding paper published in the Journal of Artificial Intelligence Research in the preceding five calendar years.

Macao, 13 August 2019

Shaul Markovitch
Editor-in-Chief, JAIR

Holger H. Hoos
Chair, 2019 IJCAI-JAIR Best Paper Prize Selection Committee
Associate-Editor-in-Chief, JAIR
Structural Reasoning Methods for Satisfiability Solving and Beyond

DISSERTATION
submitted in partial fulfillment of the requirements for the degree of
Doktor der Technischen Wissenschaften

by
Dipl.-Ing. Benjamin Kiesl, BSc
Registration Number 1127227

to the Faculty of Informatics
at the TU Wien

Advisors: Assoc.-Univ.Prof. Dr. Martina Seidl
           a.o. Univ.-Prof. Dr. Hans Tompits

The dissertation has been reviewed by:

Olaf Beyersdorff          Christoph Weidenbach

Benjamin Kiesl

Vienna, 20th February, 2019
Set Blocked Clauses (SBC)
[KieslSeidlTompitsBiere-IJCAR’16] [KieslSeidlTompitsBiere-LMCS’18]

\[ C \text{ is set blocked on } L \subseteq C \text{ iff } (C \setminus L) \cup \bar{L} \cup D \text{ is a tautology for all } D \in F \text{ with a literal in } \bar{L} \]

- easy to check if the “witness” \( L \) is given
  - NP hard to check otherwise ("exponential" in \( |L| \))

- local redundancy property
  - only considering the resolution environment of a clause
  - in contrast to (R)AT / RUP

- strictly more powerful than blocked clauses (\( |L| = 1 \))

- most general local redundancy property super blocked clauses
  - strictly more powerful than blocked clauses
  - \( \Pi_2^P \) complete to check

Example:
\[ C = \{a \lor b\} \text{ set blocked in } F = (\bar{a} \lor b) \land (a \lor \bar{b}) \text{ by } L = \{a, b\} \]
Definition. A partial assignment $\alpha$ blocks a clause $C$ if $\alpha$ assigns the literals in $C$ to false (and no other literal).

Definition. A clause $C$ is redundant w.r.t. a formula $F$ if $F$ and $F \cup \{C\}$ are satisfiability equivalent.

Definition. A formula $F$ simplified by a partial assignment $\alpha$ is written as $F|\alpha$.

Theorem.
Let $F$ be a formula, $C$ a clause, and $\alpha$ the assignment blocked by $C$. Then $C$ is redundant w.r.t. $F$ iff exists an assignment $\omega$ such that

(i) $\omega$ satisfies $C$ and

(ii) $F|\alpha \models F|\omega$. 
Propagation Redundant (PR)

[HeuleKieslBiere-CADE’17] [HeuleKieslBiere-JAR’19]

- more general than RAT: short proofs for pigeon hole formulas without new variables

\[
C \text{ propagation redundant (PR) if exists } \omega \text{ satisfying } C \text{ with } F \mid \alpha \vdash_1 F \mid \omega
\]

so in essence replacing “|=” by “\( \vdash_1 \)” (implied by unit propagation)

where again \( \alpha \) is the clause that blocks \( C \)

- Satisfaction Driven Clause Learning (SDCL) [HeuleKieslSeidlBiere-HVC’17] best paper
  - first automatically generated PR proofs
  - prune assignments for which we have other at least as satisfiable assignments
  - (filtered) positive reduct in SaDiCaL [HeuleKieslBiere-TACAS’19] nom. best paper

- translate PR to DRAT [HeuleBiere-TACAS’18]
  - only one additional variable needed
  - shortest proofs for pigeon hole formulas

- translate DRAT to extended resolution [KieslRebolaPardoHeule-IJCAR’18] best paper

- recent separation results in [BussThapen-SAT’19]
  - but PR and can not simulate covered clauses [BarnettCernaBiere-IJCAR’20]
Mutilated Chessboard

CDCL

SDCL

[HeuleKieslBiere-NFM'19]
Landscape of Clausal Redundancy

\[ F | \alpha \models F | \omega \]

\[ F | \alpha \models F | \omega \]

satisfiability equivalence

logical equivalence

\[ F | \alpha \models \bot \]

[HeuleKieslBiere-JAR’19]
\textbf{CDCL} (formula $F$)

1 \hspace{1em} \alpha := \emptyset
2 \textbf{forever do}
3 \hspace{1em} \alpha := \text{UnitPropagate}(F, \alpha)
4 \hspace{1em} \textbf{if} \ \alpha \ \text{falsifies a clause in} \ F \ \textbf{then}
5 \hspace{1em} \quad C := \text{AnalyzeConflict}()
6 \hspace{1em} \quad F := F \land C
7 \hspace{1em} \textbf{if} \ C \ \text{is the empty clause} \ \perp \ \textbf{then return} \ \text{UNSAT}
8 \hspace{1em} \alpha := \text{BackJump}(C, \alpha)

13 \hspace{1em} \textbf{else}
14 \hspace{1em} \textbf{if} \ \text{all variables are assigned} \ \textbf{then return} \ \text{SAT}
15 \hspace{1em} l := \text{Decide}()
16 \hspace{1em} \alpha := \alpha \cup \{l\}
\textbf{SDCL} (formula $F$)

1. $\alpha := \emptyset$

2. \textbf{forever do}

3. \hspace{1em} $\alpha := \text{UnitPropagate}(F, \alpha)$

4. \hspace{1em} \textbf{if} $\alpha$ falsifies a clause in $F$ \textbf{then}

5. \hspace{2em} $C := \text{AnalyzeConflict}()$

6. \hspace{2em} $F := F \land C$

7. \hspace{2em} \textbf{if} $C$ is the empty clause $\bot$ \textbf{then return} UNSAT

8. \hspace{2em} $\alpha := \text{BackJump}(C, \alpha)$

9. \hspace{1em} \textbf{else if} the pruning predicate $P_{\alpha}(F)$ is satisfiable \textbf{then}

10. \hspace{2em} $C := \text{AnalyzeWitness}()$

11. \hspace{2em} $F := F \land C$

12. \hspace{2em} $\alpha := \text{BackJump}(C, \alpha)$

13. \hspace{1em} \textbf{else}

14. \hspace{2em} \textbf{if} all variables are assigned \textbf{then return} SAT

15. \hspace{2em} $l := \text{Decide}()$

16. \hspace{2em} $\alpha := \alpha \cup \{l\}$
Positive and Filtered Positive Reduct

[HeuleKieslSeidlBiere-HVC’17] [HeuleKieslBiere-TACAS’19]

In the positive reduct consider clauses satisfied by $\alpha$, unassigned literals and add $C$:

**Definition.** Let $F$ be a formula and $\alpha$ an assignment. Then, the positive reduct of $F$ and $\alpha$ is the formula $G \land C$ where $C$ is the clause that blocks $\alpha$ and

$$G = \{ \text{touched}_\alpha(D) \mid D \in F \text{ and } D|_\alpha = \top \}.$$

**Theorem.** Let $F$ be a formula, $\alpha$ an assignment, and $C$ the clause that blocks $\alpha$. Then, $C$ is SBC by an $L \subseteq C$ with respect to $F$ if and only if the assignment $\alpha_L$ satisfies the positive reduct.

We obtain the filtered positive reduct by not taking all satisfied clauses of $F$ but only those for which the untouched part is not implied by $F|_\alpha$ via unit propagation:

**Definition.** Let $F$ be a formula and $\alpha$ an assignment. Then, the filtered positive reduct of $F$ and $\alpha$ is the formula $G \land C$ where $G = \{ \text{touched}_\alpha(D) \mid D \in F \text{ and } F|_\alpha \not\models \bot \text{ untouched}_\alpha(D) \}$.

**Theorem.** Let $F$ be a formula, $\alpha$ an assignment, and $C$ the clause that blocks $\alpha$. Then, $C$ is SPR by an $L \subseteq C$ with respect to $F$ if and only if the assignment $\alpha_L$ satisfies the filtered positive reduct.

where SPR extends SBC in the same way by propagation as RAT extends BC
<table>
<thead>
<tr>
<th>formula</th>
<th>MAPLE</th>
<th>CHRONO</th>
<th>[HVC'17]</th>
<th>CDCL</th>
<th>positive</th>
<th>filtered</th>
<th>ACL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Urquhart-s3-b1</td>
<td>2.95</td>
<td>5.86</td>
<td></td>
<td>16.31</td>
<td>&gt; 3600</td>
<td>0.02</td>
<td>0.09</td>
</tr>
<tr>
<td>Urquhart-s3-b2</td>
<td>1.36</td>
<td>2.4</td>
<td></td>
<td>2.82</td>
<td>&gt; 3600</td>
<td>0.03</td>
<td>0.13</td>
</tr>
<tr>
<td>Urquhart-s3-b3</td>
<td>2.28</td>
<td>19.94</td>
<td></td>
<td>2.08</td>
<td>&gt; 3600</td>
<td>0.03</td>
<td>0.16</td>
</tr>
<tr>
<td>Urquhart-s3-b4</td>
<td>10.74</td>
<td>32.42</td>
<td></td>
<td>7.65</td>
<td>&gt; 3600</td>
<td>0.03</td>
<td>0.17</td>
</tr>
<tr>
<td>Urquhart-s4-b1</td>
<td>86.11</td>
<td>583.96</td>
<td></td>
<td>&gt; 3600</td>
<td>&gt; 3600</td>
<td>0.32</td>
<td>2.37</td>
</tr>
<tr>
<td>Urquhart-s4-b2</td>
<td>154.35</td>
<td>1824.95</td>
<td></td>
<td>183.77</td>
<td>&gt; 3600</td>
<td>0.11</td>
<td>0.78</td>
</tr>
<tr>
<td>Urquhart-s4-b3</td>
<td>258.46</td>
<td>&gt; 3600</td>
<td></td>
<td>129.27</td>
<td>&gt; 3600</td>
<td>0.16</td>
<td>1.12</td>
</tr>
<tr>
<td>Urquhart-s4-b4</td>
<td>&gt; 3600</td>
<td>&gt; 3600</td>
<td></td>
<td>&gt; 3600</td>
<td>&gt; 3600</td>
<td>0.14</td>
<td>1.17</td>
</tr>
<tr>
<td>Urquhart-s5-b1</td>
<td>&gt; 3600</td>
<td>&gt; 3600</td>
<td></td>
<td>&gt; 3600</td>
<td>&gt; 3600</td>
<td>1.27</td>
<td>9.86</td>
</tr>
<tr>
<td>Urquhart-s5-b2</td>
<td>&gt; 3600</td>
<td>&gt; 3600</td>
<td></td>
<td>&gt; 3600</td>
<td>&gt; 3600</td>
<td>0.58</td>
<td>4.38</td>
</tr>
<tr>
<td>Urquhart-s5-b3</td>
<td>&gt; 3600</td>
<td>&gt; 3600</td>
<td></td>
<td>&gt; 3600</td>
<td>&gt; 3600</td>
<td>1.67</td>
<td>17.99</td>
</tr>
<tr>
<td>Urquhart-s5-b4</td>
<td>&gt; 3600</td>
<td>&gt; 3600</td>
<td></td>
<td>&gt; 3600</td>
<td>&gt; 3600</td>
<td>2.91</td>
<td>24.24</td>
</tr>
<tr>
<td>hole20</td>
<td>&gt; 3600</td>
<td>1.13</td>
<td>&gt; 3600</td>
<td></td>
<td></td>
<td>0.22</td>
<td>0.55</td>
</tr>
<tr>
<td>hole30</td>
<td>&gt; 3600</td>
<td>8.81</td>
<td>&gt; 3600</td>
<td></td>
<td></td>
<td>1.71</td>
<td>4.30</td>
</tr>
<tr>
<td>hole40</td>
<td>&gt; 3600</td>
<td>43.10</td>
<td>&gt; 3600</td>
<td></td>
<td></td>
<td>7.94</td>
<td>20.38</td>
</tr>
<tr>
<td>hole50</td>
<td>&gt; 3600</td>
<td>149.67</td>
<td>&gt; 3600</td>
<td></td>
<td></td>
<td>25.60</td>
<td>68.46</td>
</tr>
<tr>
<td>mchess_15</td>
<td>51.53</td>
<td>1473.11</td>
<td>2480.67</td>
<td>&gt; 3600</td>
<td></td>
<td>13.14</td>
<td>29.12</td>
</tr>
<tr>
<td>mchess_16</td>
<td>380.45</td>
<td>&gt; 3600</td>
<td>2115.75</td>
<td>&gt; 3600</td>
<td></td>
<td>15.52</td>
<td>36.86</td>
</tr>
<tr>
<td>mchess_17</td>
<td>2418.35</td>
<td>&gt; 3600</td>
<td>&gt; 3600</td>
<td>&gt; 3600</td>
<td></td>
<td>25.54</td>
<td>57.83</td>
</tr>
<tr>
<td>mchess_18</td>
<td>&gt; 3600</td>
<td>&gt; 3600</td>
<td>&gt; 3600</td>
<td>&gt; 3600</td>
<td></td>
<td>43.88</td>
<td>100.71</td>
</tr>
</tbody>
</table>
Further things we could discuss …

- relation to proof complexity  Banff, Fields, Dagstuhl seminars
- extensions formalisms: QBF, Pseudo-Boolean, #SAT, …
- local search  this year’s best solvers have all local search in it
- challenges: arithmetic reasoning (and proofs)
  - best paper [KaufmannBiereKauers-FMCAD’17]  [PhD thesis Daniela Kaufmann 2020]
- chronological backtracking  [RyvchinNadel-SAT’18] [MöhleBiere-SAT’19]
- incremental SAT solving  [PhD thesis of Katalin Fazekas in 2020]
  - best student paper [FazekasBiereScholl-SAT’19]
- parallel and distributed SAT solving  Handbook of Parallel Constraint Reasoning, …
- and probably many more …