



Introduction to SAT

Alexander Nadel

Technion, Data and Decision Sciences & Intel, Israel

Indian SAT+SMT School, 2024

August 18, 2024

Pune, Maharashtra, India

Introduction

$$\mathbf{F} = (\overset{\text{clause \#1}}{\mathbf{a} \vee \mathbf{b}}) \wedge (\overset{\text{clause \#2}}{\neg \mathbf{a} \vee \neg \mathbf{b} \vee \mathbf{c}})$$

Literals

SAT: determine if a Boolean formula in Conjunctive Normal Form (CNF) is satisfiable

The original NP-Complete problem

- The famous Cook-Levin theorem (early 70s)

SAT has exponential complexity unless $P = NP$

$P = NP$ (SAT): frequently called the most important outstanding question in CS

- If it is easy to check that a solution to a problem is correct, is it also easy to solve the problem?
- One of the 7 Clay Millennium Prize Problems – worth \$1,000,000

Introduction

SAT is an unresolved mystery!

Yet, SAT solvers are scalable widely used tools

Main goals for today:

- Explain how modern SAT solvers work
- Convey intuition why they work in practice
- Provide examples of applying SAT

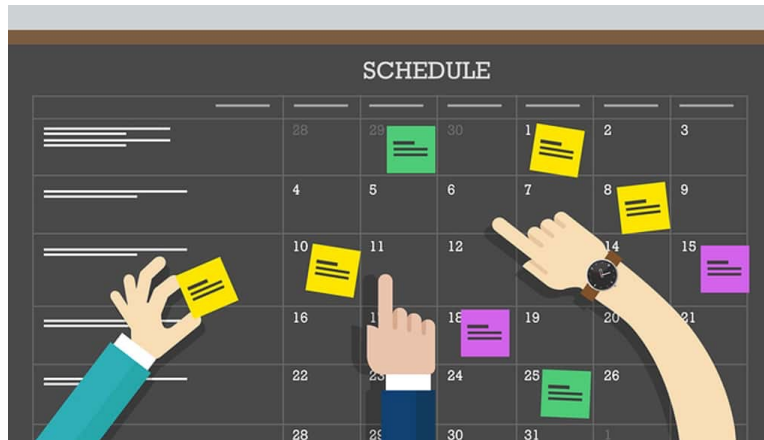


SAT Applications

Computer-Aided Design



Software Validation



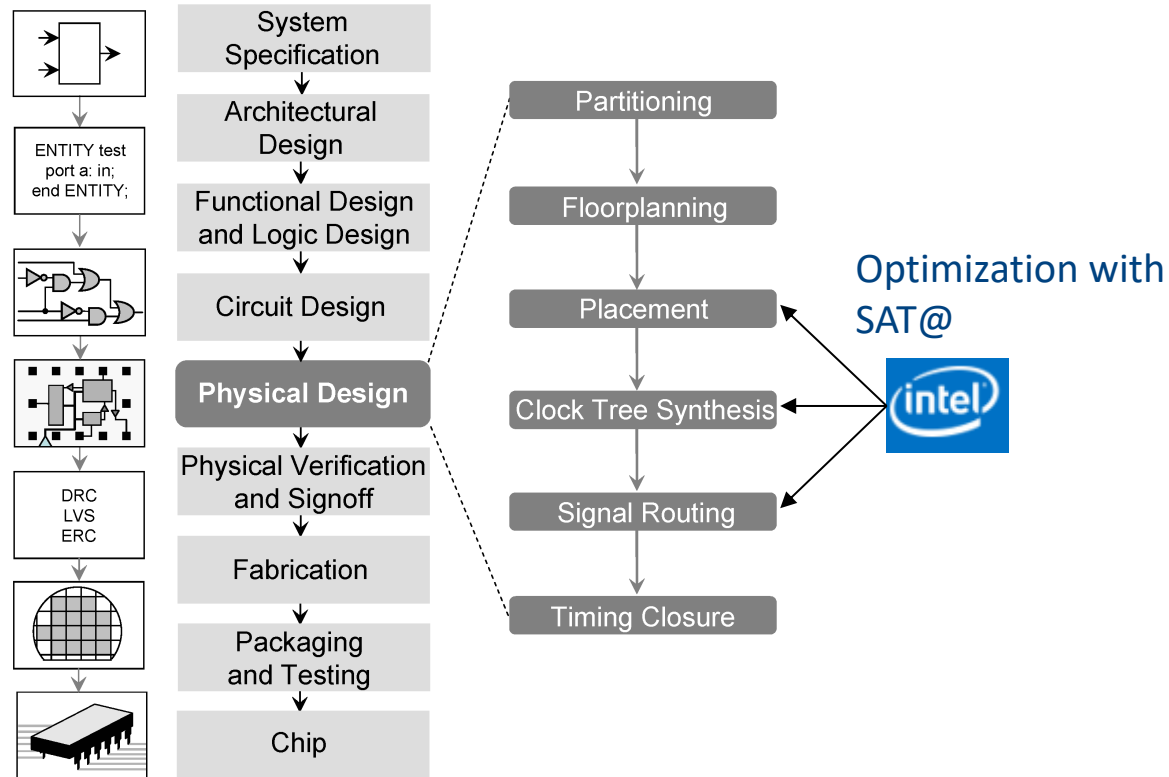
SAT Application Examples

CAV 2022 PROGRAM

Ziyad Hanna

Harnessing the Power of Formal Verification for the \$Trillion Chip Design Industry ([abstract](#))

Invited Talk: A Billion SMT Queries a Day (Neha Rungta)



MAY 30, 2016 [REPORT](#)

Computer generated math proof is largest ever at 200 terabytes

by Bob Yirka , Phys.org



Credit: Victorgrigas/Wikideia/ CC BY-SA 3.0

(Phys.org)—A trio of researchers has solved a single math problem by using a supercomputer to grind through over a trillion color combination possibilities, and in the process has generated the largest math proof ever—the text of it is 200 terabytes in size. In their paper uploaded to the preprint server *arXiv*, Marijn Heule with the University of Texas, Oliver Kullmann with Swansea University and Victor Marek with the University of Kentucky outline the math problem, the means by which a supercomputer was programmed to solve it, and the answer which the proof was asked to provide.

The math problem has been named the boolean Pythagorean Triples problem and was first proposed back in the 1980's by mathematician Ronald Graham. In looking

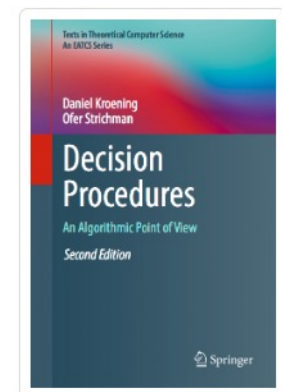
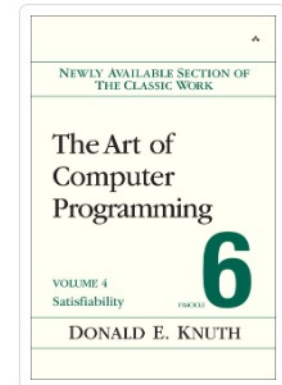
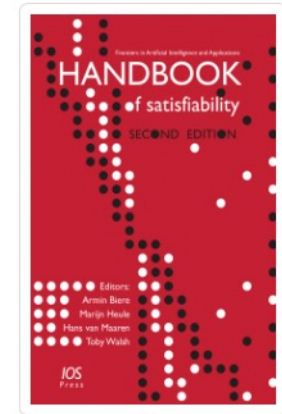
SAT Resources

SAT Association: <http://satassociation.org/>

SAT Conferences: <http://www.satisfiability.org/>

SAT Competitions: <http://www.satcompetition.org/>

SATLive: <http://www.satlive.org/>



Why am I Interested in SAT?

2002: stumbled upon SAT and completed my Master thesis about it (Hebrew University)

- Jerusat – won the Industrial, SAT category at SAT Competition 2004

2003: joined Intel

- Been developing & internally deploying SAT and SAT-based solvers (SMT, Model Checkers) ever since
- Till 2014: working on SAT-based validation as the rest of the semiconductor industry
- Since 2014: optimization (place & route, scheduling), test generation, physical design, lithography, ...

2009: PhD about SAT (Tel-Aviv University)

2023: joined the Technion's Data and Decision Sciences faculty as a part-time research fellow

- Looking for students!

Most SW is closed-sourced, but lately I was able to participate in some open-source projects:

- 2018: MapleLCMDistChronoBT SAT solver – won the SAT Competition 2018
- 2019–2024 : TT-Open-WBO-Inc MaxSAT solver – multiple medals in MaxSAT Evaluations
- 2022: Intel released my new SAT solver “Intel® SAT Solver” (IntelSAT), tuned for optimization flows

Agenda

How does a conflict-driven SAT solver work?

- The core: backtrack search, Boolean Constraint Propagation (BCP), conflict analysis
- Follow the first SAT Competition winners (from Chaff till Minisat)

Applying SAT by example

- Paradigms: incremental SAT solving, SAT-based local search, example encodings
- Applications: Bounded Model Checking (BMC), proof-based abstraction refinement, bug hunting, anytime MaxSAT

Advanced core SAT algorithms

- Follow the SAT Competition winners after Minisat & discussion about incremental SAT

Not in Today's Agenda

SAT Solving

- Parallel SAT solving (divide & conquer, portfolio, cloud)
- Non-CNF formulas
- Deserves much more attention: inprocessing, encodings

SAT-based paradigms and solvers

- Satisfiability Modulo Theories (SMT)
- Quantified Boolean Formula (QBF)
- Model counting
- AllSAT – enumerating all solutions
- Model sampling

...

SAT Fundamentals: Backtrack Search

The baseline algorithm in modern SAT solvers is backtrack search

Called DPLL or DLL

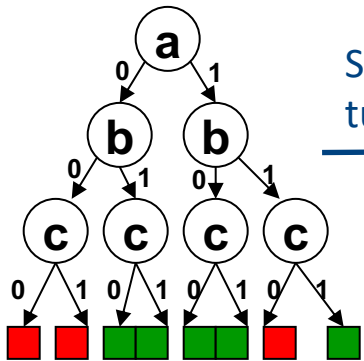
Davis, Martin; Logemann, George; Loveland, Donald: "A Machine Program for Theorem Proving". Communications of the ACM. 5 (7): 394–397. (1961).

Davis, Martin; Putnam, Hilary: A computing procedure for quantification theory. Journal of the ACM 7 (1960)

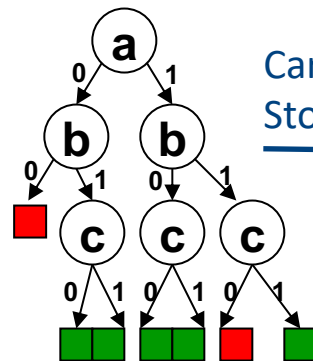
From Enumeration to DPLL

$$F = (\overset{\text{clause \#1}}{a \vee b}) \wedge (\overset{\text{clause \#2}}{\neg a \vee \neg b \vee c})$$

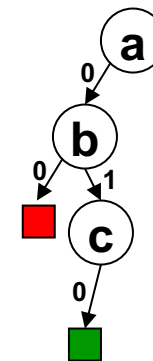
Literals



Stop when a clause turns UNSAT

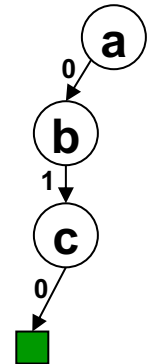


Carry out backtrack search. Stop when a model is found



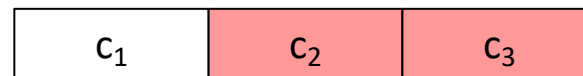
Boolean Constraint Propagation (BCP): after a decision, apply the unit clause rule till fixed-point

Implied in parent clause #1:

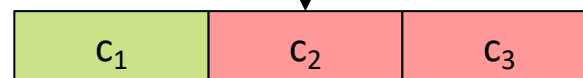


The **unit clause rule**: the unassigned literal in a unit clause must be 1

A **unit** clause -- one unassigned, rest falsified:



The unassigned literal c₁ must be **implied**



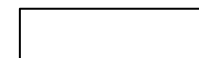
Falsified literal:



Satisfied literal:



Unassigned literal:



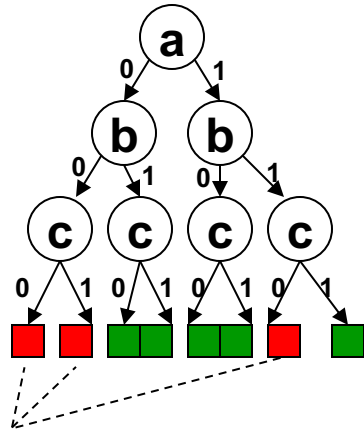
The Mystery of SAT Solver Scalability

DPLL: backtrack search with BCP until a model is found (SAT) or completion (UNSAT)

DPLL could handle formulas with <2,000 clauses

Modern SAT solvers cope with industrial instances of 100,000,000's clauses

The introduction of Conflict-Driven-Clause-Learning (CDCL) or, simply, Conflict-driven Solving was the birth of modern highly-scalable SAT solving



Learn from conflicts to drive & prune backtrack search

CDCL: the Intuitive Principles

Learning and pruning

- Block already explored sub-spaces

Locality

- Focus the search on the relevant data
- Learn strong clauses from the local context

Well-engineered data structures

- Extremely fast Boolean Constraint Propagation (BCP)

Beyond CDCL

- Inprocessing
- Local search integration

Basic CDCL Algorithm

Preprocess()

// Simplify the formula

While (true)

- Literal $l = \text{Decide}()$
- $\text{BCP}(l)$
- If (conflict)
 - $\text{ConflictAnalysisLoop}()$
- If (learned an empty clause)
 - Return UNSAT
- If (all the variables are assigned)
 - Return SAT
- Occasionally, restart
- Occasionally, delete conflict clauses

// Choose the next literal to assign

// Apply the unit clause rule till fixed point

// Learn a new *conflict clause(s)*, backtrack and flip a variable

Conflict-driven SAT Solving: Seminal Work

1996: GRASP by Joao P. Marques-Silva and Karem A. Sakallah

João P. Marques Silva, Karem A. Sakallah: GRASP - a new search algorithm for satisfiability. *ICCAD 1996*: 220-227

2001: Chaff by Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang and Sharad Malik

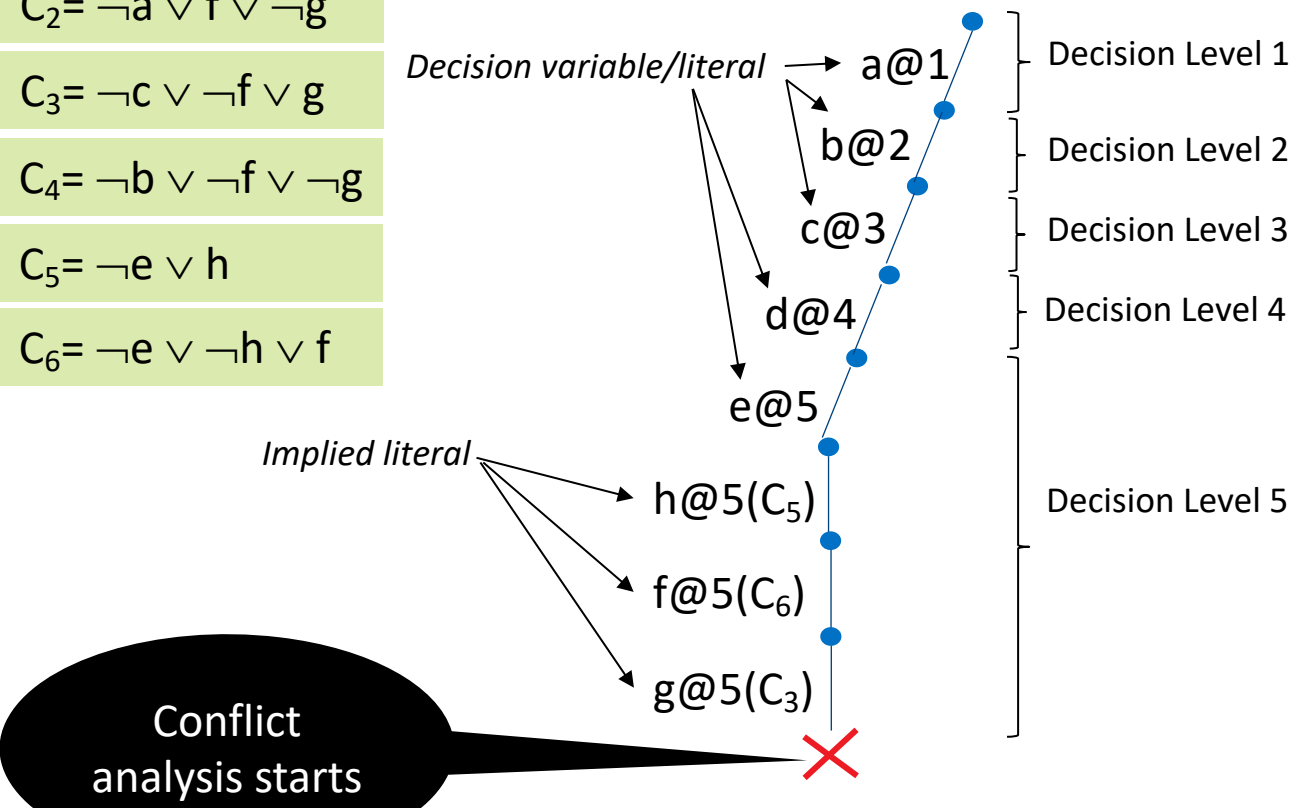
Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, Sharad Malik: Chaff: Engineering an Efficient SAT Solver. *DAC 2001*: 530-535



Abstract The 2009 CAV (Computer-Aided Verification) award was presented to seven individuals who made major advances in creating high-performance Boolean satisfiability solvers. This annual award recognizes a specific fundamental contribution or series of outstanding contributions to the CAV field.

Chaff's Conflict Analysis

- $C_1 = \neg a \vee f \vee g$
- $C_2 = \neg a \vee f \vee \neg g$
- $C_3 = \neg c \vee \neg f \vee g$
- $C_4 = \neg b \vee \neg f \vee \neg g$
- $C_5 = \neg e \vee h$
- $C_6 = \neg e \vee \neg h \vee f$



Implication Graphs and Conflict Analysis

$$C_1 = \neg a \vee f \vee g$$

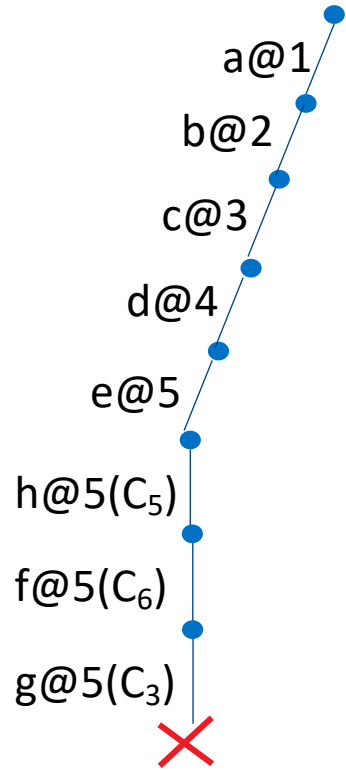
$$C_2 = \neg a \vee f \vee \neg g$$

$$C_3 = \neg c \vee \neg f \vee g$$

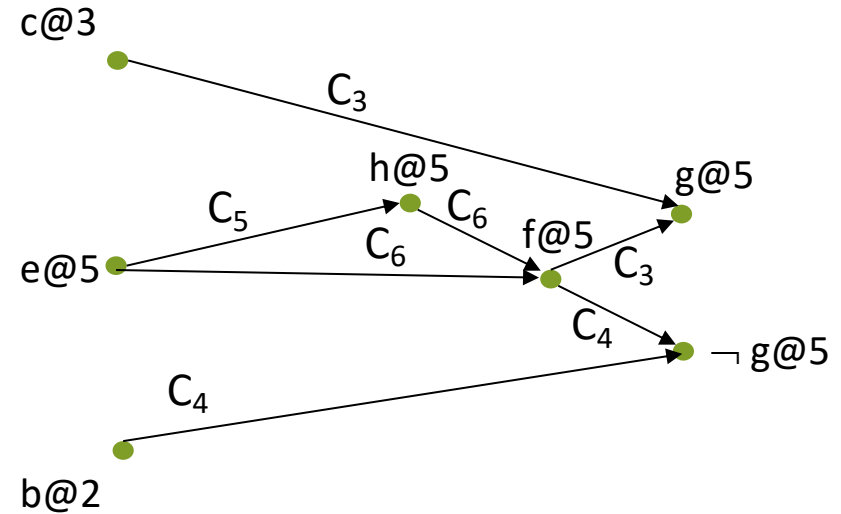
$$C_4 = \neg b \vee \neg f \vee \neg g$$

$$C_5 = \neg e \vee h$$

$$C_6 = \neg e \vee \neg h \vee f$$



Implication graph



Every vertex corresponds to an assigned literal

A decision literal has 0 incoming edges

A literal implied in clause C has $|C|-1$ incoming edges from every other literal in C

We only need the strongly connected component of the conflict

Implication Graphs and Conflict Analysis

$$C_1 = \neg a \vee f \vee g$$

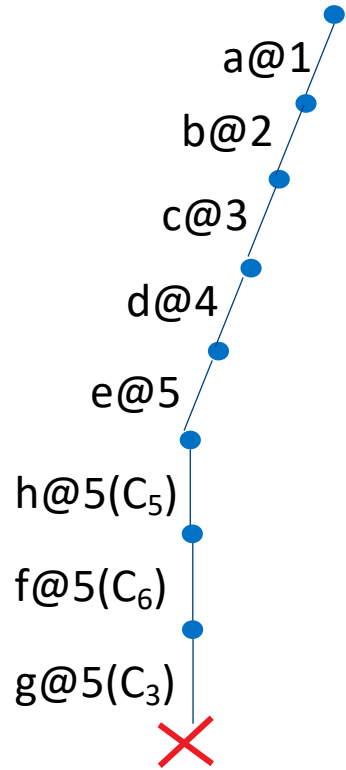
$$C_2 = \neg a \vee f \vee \neg g$$

$$C_3 = \neg c \vee \neg f \vee g$$

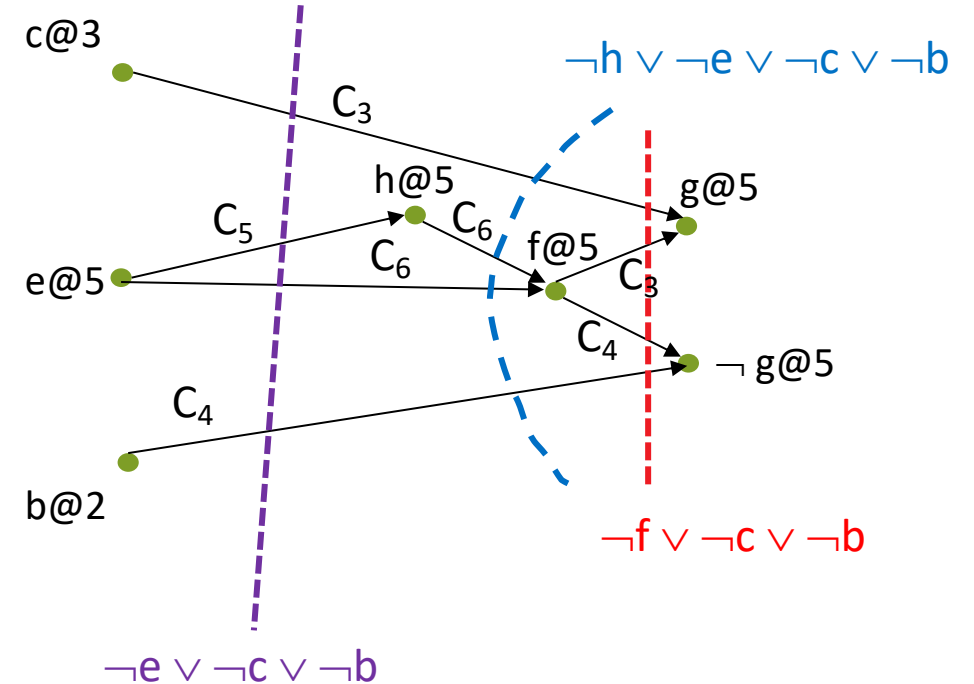
$$C_4 = \neg b \vee \neg f \vee \neg g$$

$$C_5 = \neg e \vee h$$

$$C_6 = \neg e \vee \neg h \vee f$$



Implication graph



Conflict cut

- Right (conflict): the two conflicting implications
- Left (reason): all the decision literals (roots)

Conflict clause

- Corresponds to every cut: includes one appearance of $\neg l$ for every edge $l \rightarrow r$ in the cut
- Learning a conflict clause prevents the conflict from reappearing

Implication Graphs and Conflict Analysis

$$C_1 = \neg a \vee f \vee g$$

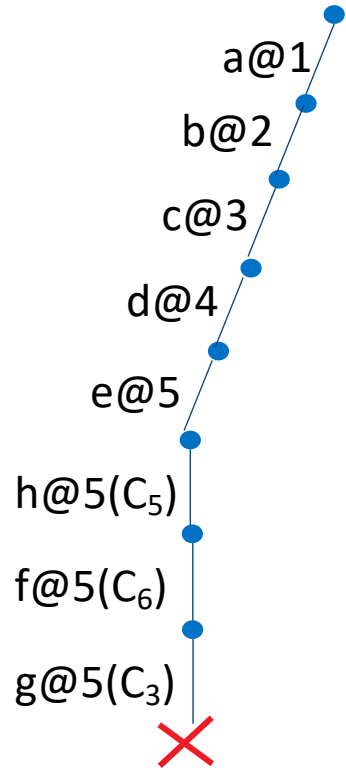
$$C_2 = \neg a \vee f \vee \neg g$$

$$C_3 = \neg c \vee \neg f \vee g$$

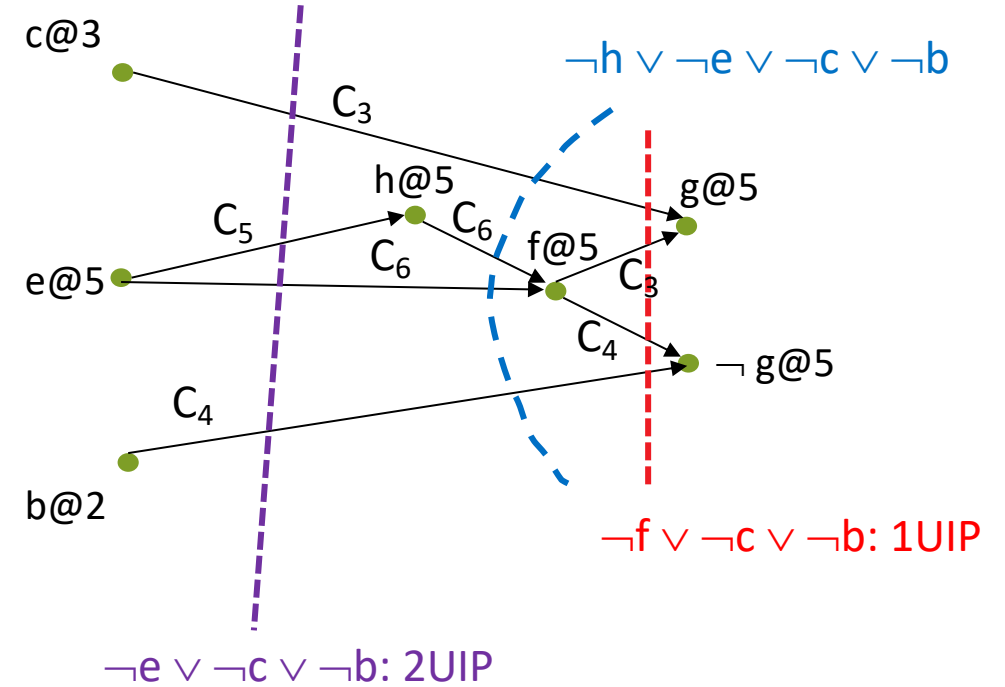
$$C_4 = \neg b \vee \neg f \vee \neg g$$

$$C_5 = \neg e \vee h$$

$$C_6 = \neg e \vee \neg h \vee f$$



Implication graph



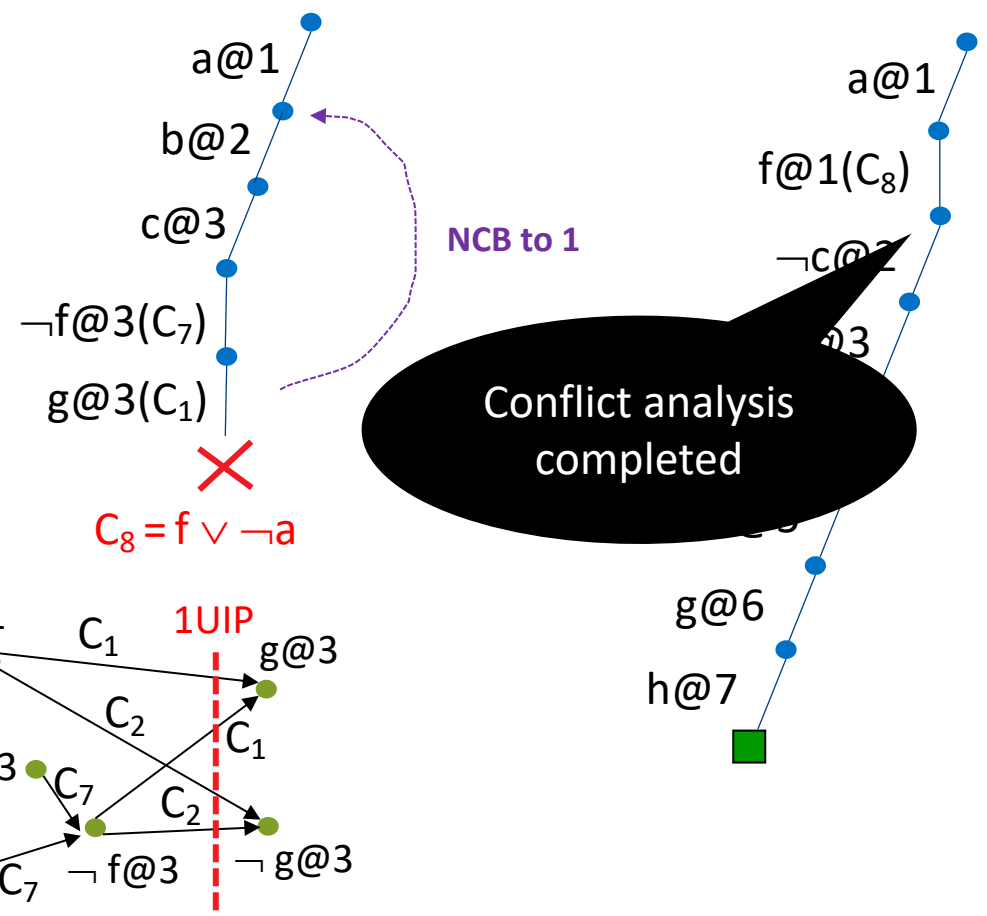
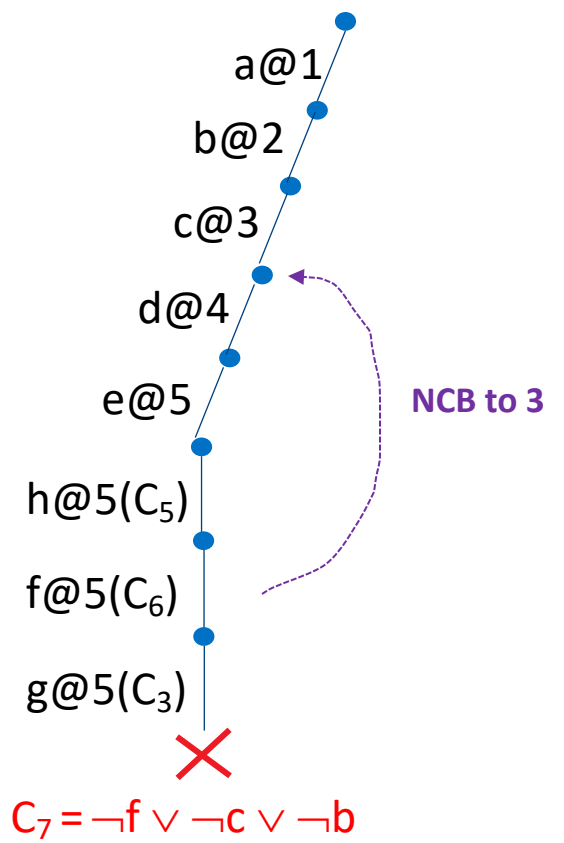
A **UIP cut** has exactly one literal l of the last level on the reason side of its edges

- l is a Unique Implication Point (UIP): a literal sufficient to imply the conflict at the last level
- A conflict clause is a **UIP clause** if it corresponds to a UIP cut

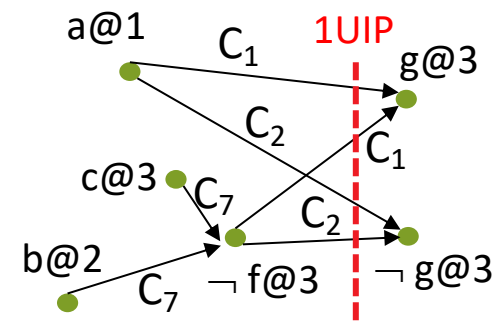
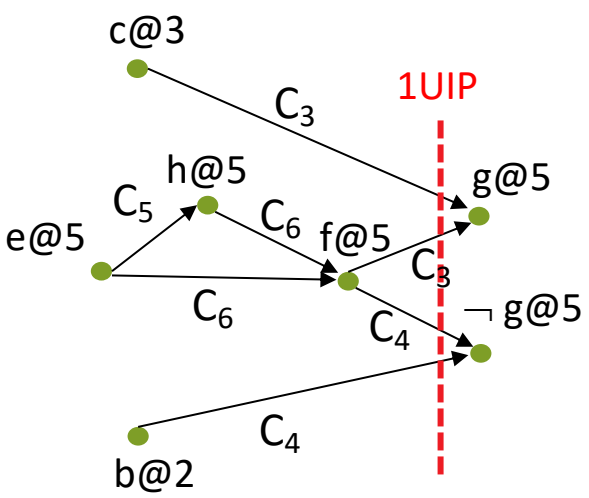
UIP's are ordered starting from the conflict

- $C_1 = \neg a \vee f \vee g$
- $C_2 = \neg a \vee f \vee \neg g$
- $C_3 = \neg c \vee \neg f \vee g$
- $C_4 = \neg b \vee \neg f \vee \neg g$
- $C_5 = \neg e \vee h$
- $C_6 = \neg e \vee \neg h \vee f$

Chaff's Conflict Analysis



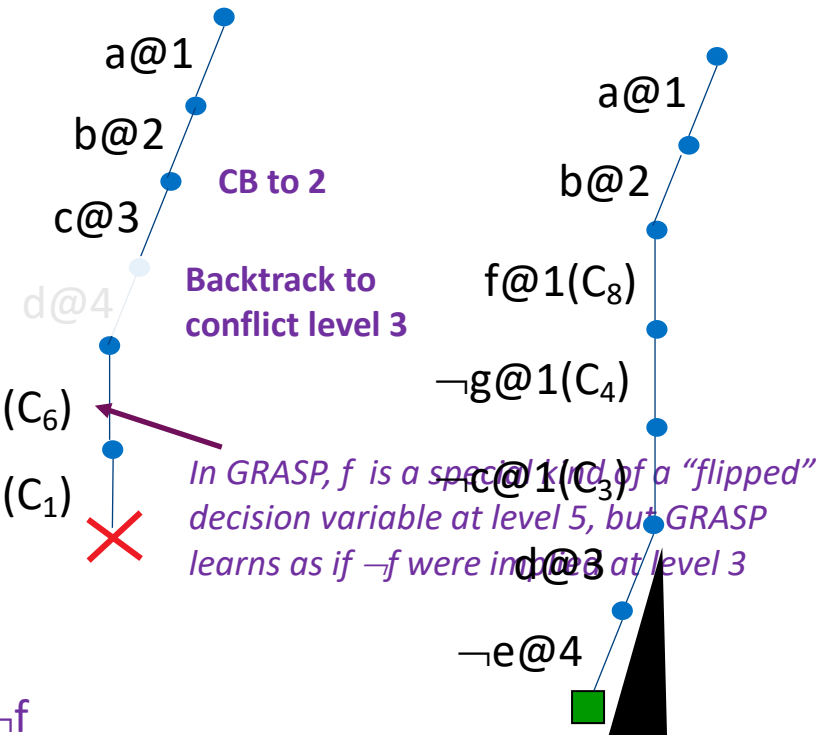
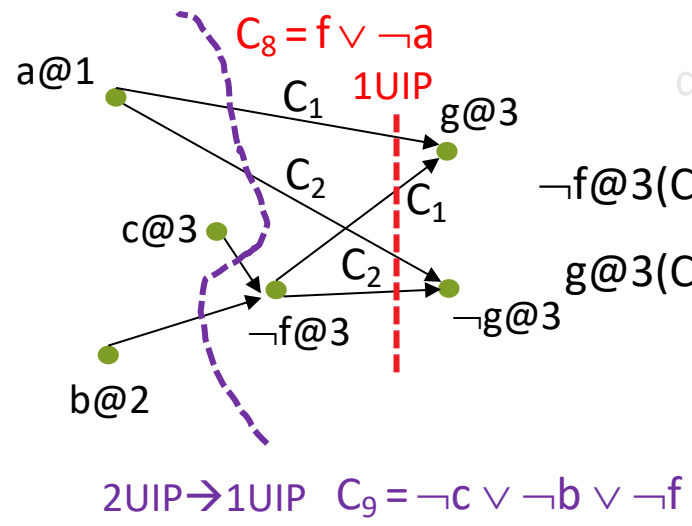
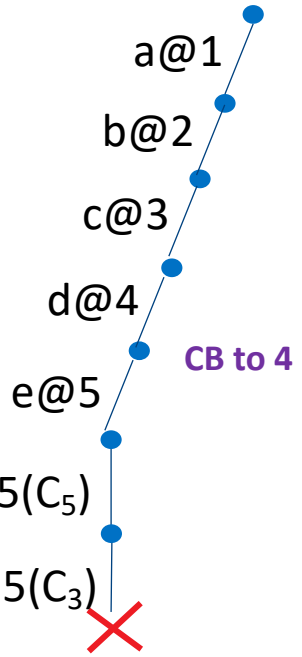
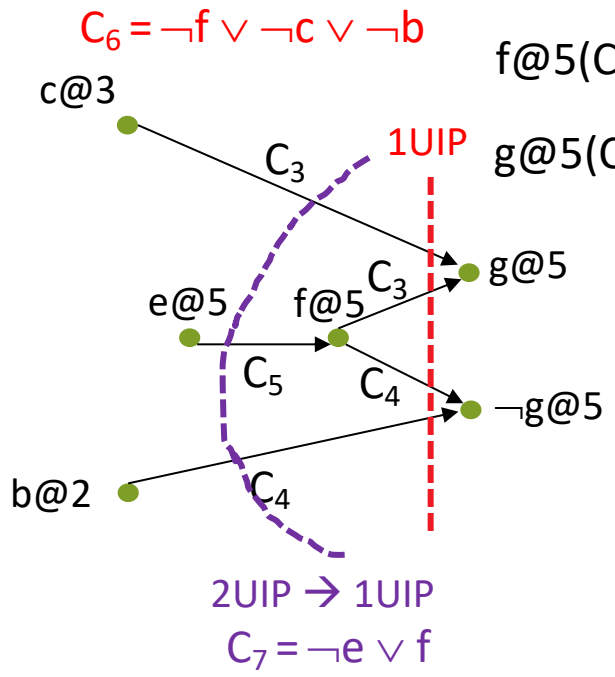
Implication graph



- Learn a falsified *asserting* clause $C = [c_1@{\delta}, c_2@{\beta < \delta}, c_3@{\leq \beta}, \dots, c_{|C|}@{\leq \beta}]$
 - 1UIP clause: fewest variables out of all UIP clauses (UIP clauses have one variable @ δ)
- Backtrack to level β : called Non-Chronological Backtracking (NCB) \rightarrow C becomes unit
- Flip & imply c_1 in its *parent* C and run BCP

GRASP's Conflict Analysis

- $C_1 = \neg a \vee f \vee g$
- $C_2 = \neg a \vee f \vee \neg g$
- $C_3 = \neg c \vee \neg f \vee g$
- $C_4 = \neg b \vee \neg f \vee \neg g$
- $C_5 = \neg e \vee f$



- Backtrack to the conflict level δ : called NCB in GRASP (unnamed to date)
- Learn a falsified asserting 1UIP clause $C = [c_1@{\delta}, c_2@{\beta < \delta}, c_3@{\leq \beta}]$
- Learn a clause per every other UIP of the last level
- Backtrack to level $\delta - 1$: called Chronological Backtracking
- Flip & imply c_1 in its parent C and run BCP

Conflict analysis completed

Up-to-date Conflict Analysis Algorithm

Covers GRASP & Chaff & Modern Solvers

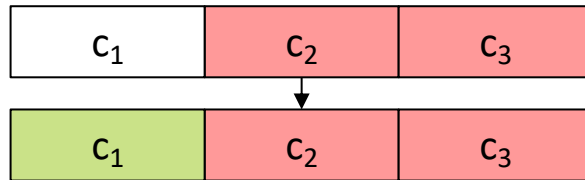
1. Backtrack before conflict analysis: backtrack to the conflict level δ , if required
 - Required in GRASP
 - Not required in Chaff: current decision level \equiv conflict level
2. Learn an asserting clause $C = [c_1^{@ \delta}, c_2^{@ \beta < \delta}, c_3^{@ @ \leq \beta}, \dots, c_i^{@ @ \leq \beta}, \dots, c_{|C|}^{@ \leq \beta}]$
 - 1UIP clause in both GRASP & Chaff
 - Return UNSAT, if the clause is empty
3. Optionally, learn other clauses
 - GRASP: a clause for every other UIP of the conflict decision level
4. Backtrack: backtrack to a level in $[\beta, \beta+1, \dots, \delta-1]$ -- renders the asserting clause unit
 - GRASP -- always $\delta-1$: Chronological Backtracking (CB) in today's terminology
 - Chaff -- always β : Non-Chronological Backtracking (NCB) in today's terminology
5. Flip c_1 by implying it in C and run BCP

Boolean Constraint Propagation (BCP) Essentials

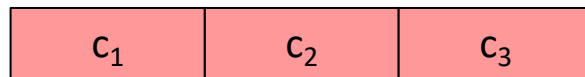
BCP is carried out after every decision and flip and consumes 80-90% run-time

What?

- Identify and propagate in unit clauses (*performance*)



- Identify and report any conflicts (*correctness*)

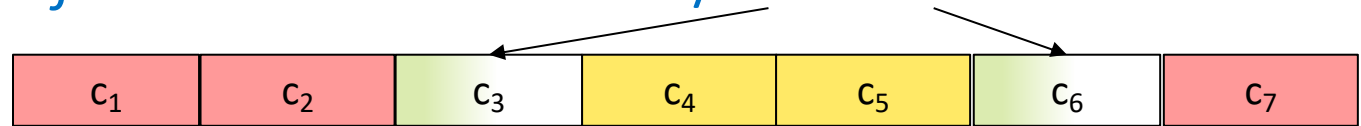


How?

- Visit a clause when one of its *watched* literals is falsified
 - Every literal l holds a *Watch List (WL)* with all the clauses where l is watched

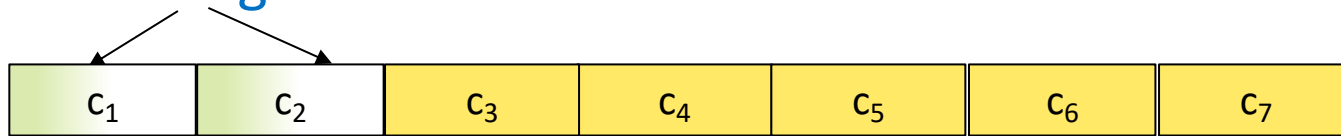
Efficient Data Structure for BCP

- GRASP watched all the literals in every clause
- It is sufficient to watch two *non-falsified* literals: SATO's Head/Tail!



Hantao Zhang: SATO: An Efficient Propositional Prover. *CADE 1997*: 272-275

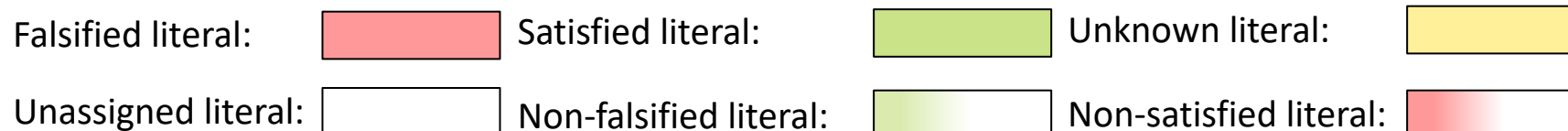
- Chaff's 2WL: watching the first two literals – no need to visit during backtracking!



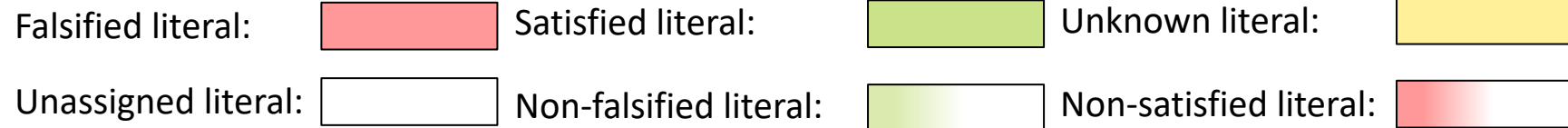
- as long as: $\text{decision-level}(\text{falsified watch}) \geq \text{decision-level}(\text{falsified non-watch})$
- Caching one literal inside the watches & inlining binary clauses

Sörensson, N., Eén, N.: *MiniSAT 2.1 and MiniSAT++ 1.0 - SAT race Editions*. SAT, Competitive Event Booklet (2008) (*caching one literal*)

Geoffrey Chu, Aaron Harwood, Peter J. Stuckey: *Cache Conscious Data Structures for Boolean Satisfiability Solvers*. *J. Satisf. Boolean Model. Comput.* 6(1-3): 99-120 (2009) (*caching one literal & inlining binary clauses*)



BCP assuming NCB



For every satisfied literal l in the literal stack Π (literals to be propagated)

- For $\langle h \neq \neg l \in C, C \rangle \in WL(\neg l)$

- If h is satisfied: continue

Clause visit: assume WLOG $c_2 \equiv \neg l$

- If c_1 is satisfied: continue

- If a non-falsified $k \neq c_1 \in C$ exists

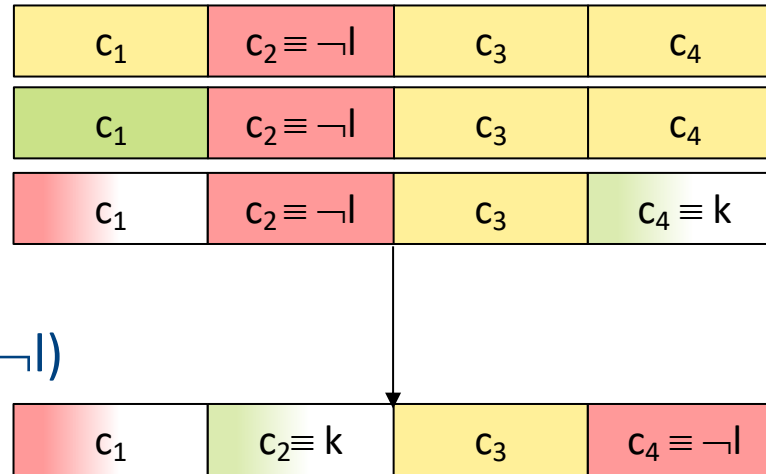
- Swap($C, k, \neg l$)
- Remove $\langle h \neq \neg l \in C, C \rangle$ from $WL(\neg l)$
- Add $\langle h' \neq k \in C, C \rangle$ to $WL(k)$

- Else (*unit or falsified*)

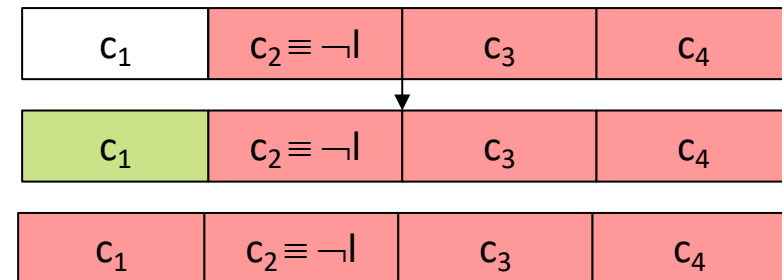
- If c_1 is unassigned, imply c_1 and add c_1 to Π (*unit*)
- If c_1 is falsified, report a conflict and return C (*falsified*)

h : cached literal; C : the visited clause, where $c_1 \equiv \neg l$ or $c_2 \equiv \neg l$

C is satisfied: no conflict, C isn't unit \rightarrow skip C



: heuristical



Agenda

How does a conflict-driven SAT solver work?

- The core: backtrack search, Boolean Constraint Propagation (BCP), conflict analysis
- Follow the first SAT Competition winners (from Chaff till Minisat)

Applying SAT by example

- Paradigms: incremental SAT solving, SAT-based local search, example encodings
- Applications: Bounded Model Checking (BMC), proof-based abstraction refinement, bug hunting, anytime MaxSAT

Advanced core SAT algorithms

- Follow the SAT Competition winners after Minisat & discussion about incremental SAT

SAT Competition & Race Winners (CNF & Appl. & Seq. & Non-incr. & All-inst.)

2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
zChaff	Forklift	zChaff	SatELite GTI	MiniSat	RSAT	MiniSat	Precosat	Crypto MiniSat	Glucose
Moskewicz Madigan Zhao Zhang Malik	Goldberg Novikov	Moskewicz Madigan Zhao Zhang Malik	Eén Sörensson	Eén Sörensson	Pipatsrisa wat Darwiche	Eén Sörensson	Biere	Soos	Audemard Simon
2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Glucose	Lingeling	Lingeling	abcdSAT	Maple COMSPS	Maple LCMDist	Maple LCMDist ChronoBT	Maple LCMDist ChronoBT DLv3	Kissat	KissatMAB
Audemard Simon	Biere	Biere	Chen	Liang Oh Ganesh Czarnecki Poupart	Xiao Luo Li Manya Lu	Nadel Ryvchin	Kochemazov Zaikin Kondratiev Semenov	Biere Fazekas Fleury Heisinger	Cherif Habet Terrioux
2022	2023								
KissatMAB-HyWalk	SBVA-CaDiCaL								
Zheng He Chen Zhou Li	Haberlandt Green								

MiniSat-based:


Armin Biere's & derived:


Others:


Chaff

Covered:

- Conflict analysis
- BCP

To cover:

- Variable State Independent Decaying Sum (VSIDS) decision heuristic
 - The first conflict-driven decision heuristic
- Conflict clause deletion
- Restarts

Variable State Independent Decaying Sum (VSIDS)

Each literal l has a counter $S(l)$, initialized to 0

For every new clause $C=[c_1, c_2, \dots, c_n]$, $S(c_i)$ is incremented for every $c_i \in C$

Including initial and conflict clauses

The (unassigned) variable and polarity with the highest counter is chosen

Ties are broken randomly

Periodically (once in 256 conflicts), all the counters are halved.

VSIDS Example

Heuristic-related data

Literal	Score
a	0
$\neg a$	0
b	0
$\neg b$	0
c	0
$\neg c$	0
d	0
$\neg d$	0
e	0
$\neg e$	0
...	...

Search tree



Conflicts till now: 0

VSIDS Example

Heuristic-related data

Literal	Score
a	4
$\neg a$	5
b	3
$\neg b$	3
c	2
$\neg c$	3
d	2
$\neg d$	4
e	2
$\neg e$	6
...	...

Count literal appearances in the initial formula



Search tree



Conflicts till now: 0

VSIDS Example

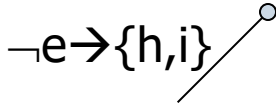
Heuristic-related data

Literal	Score
a	4
\neg a	5
b	3
\neg b	3
c	2
\neg c	3
d	2
\neg d	4
e	2
\neg e	6
...	...

Pick a literal with maximal score



Search tree



Conflicts till now: 0

VSIDS Example

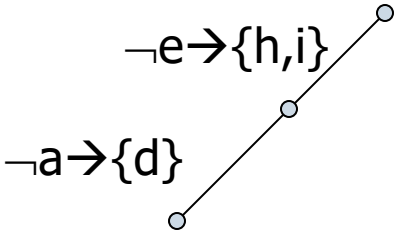
Heuristic-related data

Literal	Score
a	4
$\neg a$	5
b	3
$\neg b$	3
c	2
$\neg c$	3
d	2
$\neg d$	4
e	2
$\neg e$	6
...	...

Pick an unassigned literal with maximal score



Search tree



Conflicts till now: 0

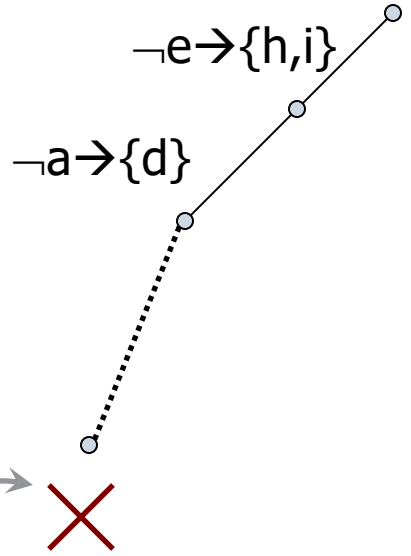
VSIDS Example

Conflict

Heuristic-related data

Literal	Score
a	4
$\neg a$	5
b	3
$\neg b$	3
c	2
$\neg c$	3
d	2
$\neg d$	4
e	2
$\neg e$	6
...	...

Search tree



Conflicts till now: 0

VSIDS Example

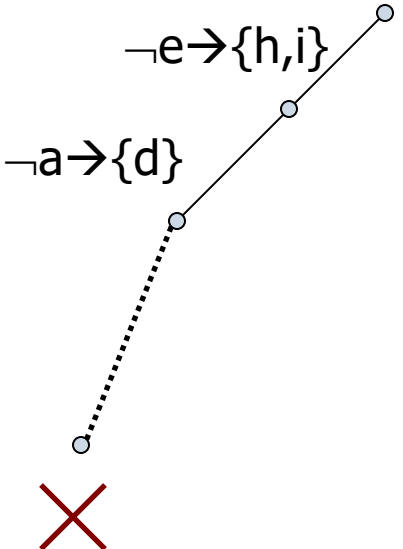
Heuristic-related data

Literal	Score
a	4→5
¬a	5
b	3
¬b	3→4
c	2→3
¬c	3
d	2
¬d	4
e	2
¬e	6
...	...

Increment scores for conflict clause literals



Search tree



¬h ∨ a ∨ c ∨ ¬b ∨ k

Conflicts till now: 1

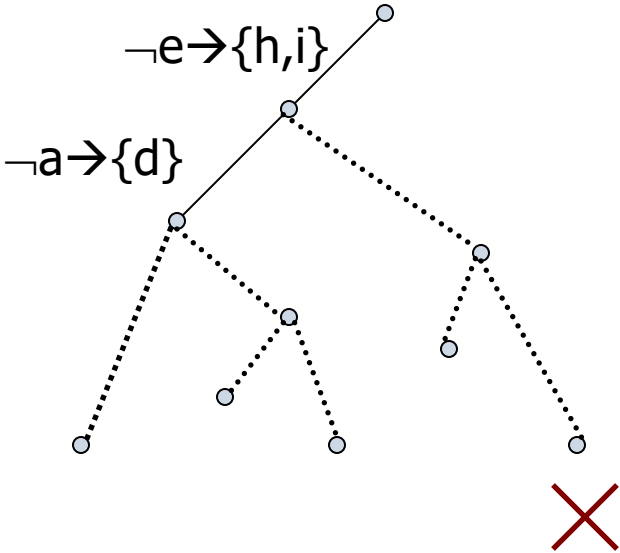
VSIDS Example

Heuristic-related data

Literal	Score
a	10
$\neg a$	12
b	18
$\neg b$	6
c	12
$\neg c$	6
d	2
$\neg d$	6
e	16
$\neg e$	6
...	...

Assume the threshold of 256 is reached

Search tree



Conflicts till now: 256

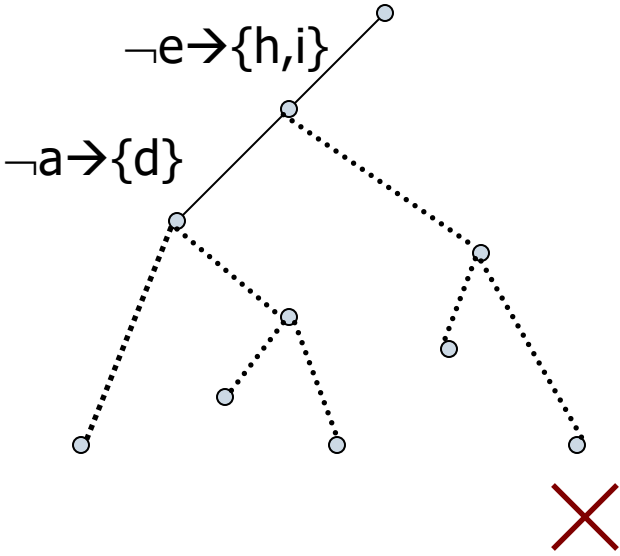
VSIDS Example

Heuristic-related data

Halve the scores

Literal	Score
a	10→5
¬a	12→6
b	18→9
¬b	6→3
c	12→6
¬c	6→3
d	2→1
¬d	6→3
e	16→8
¬e	6→3
...	...

Search tree



Conflicts till now: 256

VSIDS vs. Static Heuristics

Pre-Chaff static heuristics

- Go over all clauses that are not satisfied
- Compute some function $f(a)$ for each literal—based on frequency
- Choose literal with maximal $f(a)$

VSIDS was a breakthrough

- Extremely low overhead
- Conflict-driven → dynamic and local
 - Based on recent conflicts
 - Focuses the search to learn from the local context

Conflict Clause Deletion

Maintaining too many clauses slows down the solver

D. Gelperin: Deletion-directed search in resolution-based proof procedures, in Proc. of the 3rd Int. Joint Conf. on Artificial Intelligence (1973), pp. 47–50.

Chaff's strategy:

- Mark a clause for deletion, once 100-200 literals become unassigned

Restarts

C. P. Gomes, B. Selman and H. A. Kautz: Boosting combinatorial search through randomization, in Proc. of AAAI (1998), pp. 431–437

Refocus the search by starting from important variables

Chaff: restart every 700 conflicts

Chaff

Preprocess()

While (true)

- Literal l = Decide() // VSIDS
- BCP(l) // 2WL
- If (conflict)
 - ConflictAnalysisLoop() // 1UIP + non-chronological backtracking
- If (learned an empty clause)
 - Return UNSAT
- If (all the variables are assigned)
 - Return SAT
- Occasionally, restart // Every 700 conflicts
- Occasionally, delete conflict clauses // Mark for deletion, when 100-200 lit's are unassigned

SAT Competition & Race Winners (CNF & Appl. & Seq. & Non-incr. & All-inst.)

2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
zChaff	Forklift	zChaff	SatELite GTI	MiniSat	RSAT	MiniSat	Precosat	Crypto MiniSat	Glucose
Moskewicz Madigan Zhao Zhang Malik	Goldberg Novikov	Moskewicz Madigan Zhao Zhang Malik	Eén Sörensson	Eén Sörensson	Pipatsrisa wat Darwiche	Eén Sörensson	Biere	Soos	Audemard Simon
2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Glucose	Lingeling	Lingeling	abcdSAT	Maple COMSPS	Maple LCMDist	Maple LCMDist ChronoBT	Maple LCMDist ChronoBT DLv3	Kissat	KissatMAB
Audemard Simon	Biere	Biere	Chen	Liang Oh Ganesh Czarnecki Poupart	Xiao Luo Li Manya Lu	Nadel Ryvchin	Kochemazov Zaikin Kondratiev Semenov	Biere Fazekas Fleury Heisinger	Cherif Habet Terrioux
2022	2023								
KissatMAB-HyWalk	SBVA-CaDiCaL								
Zheng He Chen Zhou Li	Haberlandt Green								

MiniSat-based:


Armin Biere's & derived:


Others:


BerkMin & Forklift by Goldberg & Novikov

Forklift: industrial closed-source solver (Cadence)

We discuss Forklift's direct ancestor BerkMin (won Handmade, SAT category at SC'02)

Goldberg, Novikov: BerkMin: A fast and robust SAT-solver, DATE, 2002.

Clause deletion is based on “age” and size. The strategy, simplified:

- Age: remove clauses which did not participate in recent conflict clause derivation
- Size: keep short clauses forever ($|C| < 8$)

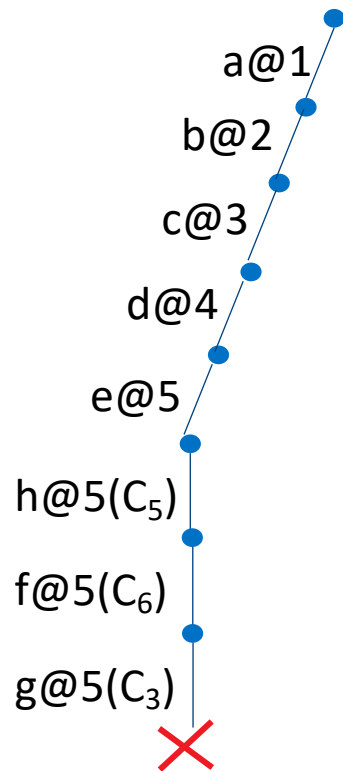
Restarts every 550 conflicts

Innovation in decision heuristics

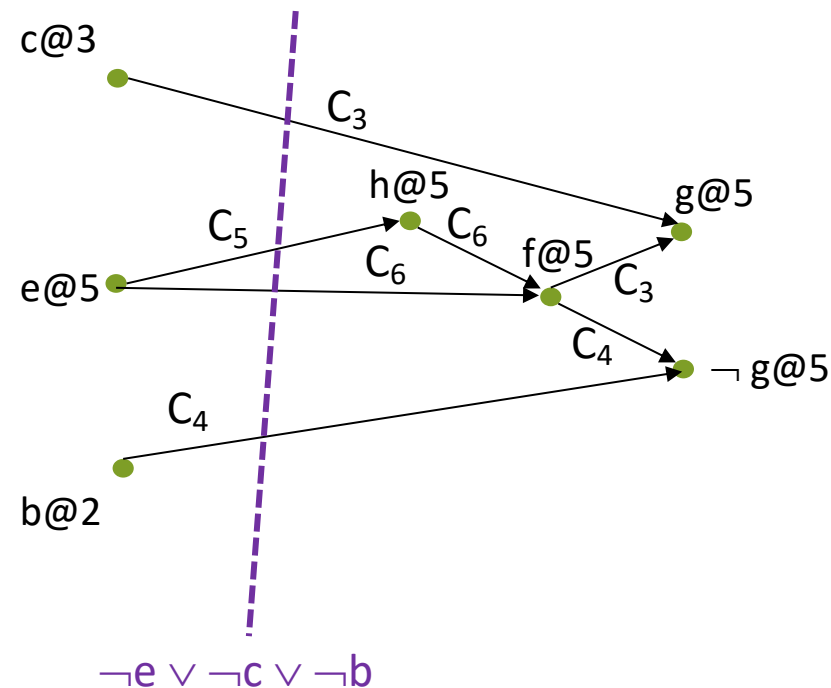
- Boost the score for all the literals visited during conflict analysis (rather than only in the conflict clause)

Extended Score Boost Example

- $C_1 = \neg a \vee f \vee g$
- $C_2 = \neg a \vee f \vee \neg g$
- $C_3 = \neg c \vee \neg f \vee g$
- $C_4 = \neg b \vee \neg f \vee \neg g$
- $C_5 = \neg e \vee h$
- $C_6 = \neg e \vee \neg h \vee f$



Implication graph



- Chaff's: boost the scores of $\neg e$, $\neg c$ and $\neg b$
- BerkMin: additionally to $\neg e$, $\neg c$ and $\neg b$, boost the scores of h , f , g , $\neg g$

BerkMin & Forklift by Goldberg & Novikov

Forklift: industrial closed-source solver (Cadence)

We discuss Forklift's direct ancestor BerkMin (won Handmade, SAT category at SC'02)

Goldberg, Novikov: BerkMin: A fast and robust SAT-solver, DATE, 2002.

Clause deletion is based on “age” and size. The strategy, simplified:

- Age: remove clauses which did not participate in recent conflict clause derivation
- Size: keep short clauses forever ($|C| < 8$)

Restarts every 550 conflicts

Innovation in decision heuristics

- Boost the score for all the literals visited during conflict analysis (rather than only in the conflict clause)
 - Used in modern solvers
- Clause-based heuristic

Clause-based Heuristics

Berkmin

- The conflict clauses are placed on a stack
- The next variable is picked from the topmost unsatisfied clause
- If no such clause exists, use VSIDS

HaifaSAT (3^d in three Industrial categories at SC'05)

Roman Gershman, Ofer Strichman: HaifaSat: A New Robust SAT Solver. Haifa Verification Conference 2005: 76-89

- Move clauses visited during conflict analysis to the top

CBH -- Eureka SAT solver (2nd at SR'06)

Nachum Dershowitz, Ziyad Hanna, Alexander Nadel: A Clause-Based Heuristic for SAT Solvers. SAT 2005: 46-60

- Either all the clauses (including the initial clauses) or only the initial clauses are on the stack
- Move clauses visited during conflict analysis to the top

Added value w.r.t variable-based heuristics: picks interrelated variables


Didn't make it to mainstream modern solvers, though CBH is occasionally very useful in my experience


- 2023: modified CBH works great for constraint-based product configuration

Matthias Gorenflo, Tomás Balyo, Markus Iser, Tobias Ostertag: **Decision Heuristics in a Constraint-based Product Configurator**. ConfWS 2023: 51-59

SAT Competition & Race Winners (CNF & Appl. & Seq. & Non-incr. & All-inst.)

2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
zChaff	Forklift	zChaff	SatELite GTI	MiniSat	RSAT	MiniSat	Precosat	Crypto MiniSat	Glucose
Moskewicz Madigan Zhao Zhang Malik	Goldberg Novikov	Moskewicz Madigan Zhao Zhang Malik	Eén Sörensson	Eén Sörensson	Pipatsrisa wat Darwiche	Eén Sörensson	Biere	Soos	Audemard Simon
2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Glucose	Lingeling	Lingeling	abcdSAT	Maple COMSPS	Maple LCMDist	Maple LCMDist ChronoBT	Maple LCMDist ChronoBT DLv3	Kissat	KissatMAB
Audemard Simon	Biere	Biere	Chen	Liang Oh Ganesh Czarnecki Poupart	Xiao Luo Li Manya Lu	Nadel Ryvchin	Kochemazov Zaikin Kondratiev Semenov	Biere Fazekas Fleury Heisinger	Cherif Habet Terrioux
2022	2023								
KissatMAB-HyWalk	SBVA-CaDiCaL								
Zheng He Chen Zhou Li	Haberlandt Green								

MiniSat-based:


Armin Biere's & derived:


Others:


MiniSat & SatELite: Seminal Works

Niklas Eén, Niklas Sörensson: An Extensible SAT-solver. SAT 2003: 502-518

- Minisat solver
- “Test of time” award at SAT’22
- Simple & elegant engineering: the ancestor of a long line of solvers!
- Impactful heuristics & algorithms:
 - Exponential VSIDS (EVSIDS)
 - sometimes still called VSIDS today
 - Conflict clause minimization
- Incremental-under-assumptions API: enabler of major real-world flows

Niklas Eén, Armin Biere: Effective Preprocessing in SAT Through Variable and Clause Elimination. SAT 2005: 61-75

- SatELite Preprocessing

Minisat's Decision Heuristic

Separate variable and polarity heuristics

- Keep score per variable, rather than per literal
- Choose 0 as the first polarity

EVSIDS

- Increment activity by an exponentially increasing increment $(g=1/f)^{\#conflict}$
 - Minisat: $f=0.95 \rightarrow g \approx 1.05$
- Rescale when activity (for any variable) becomes higher than 10^{100}
 - $g *= 10^{-100}$
 - `#conflict = 1`
- Even more dynamic than VSIDS

Both features (further updated) made it to today's state-of-the-art solvers

Restart & Clause Deletion in Minisat

Restarts

- Geometric series
 - starting from 100, with the factor of 1.5
- Too slow: didn't make it to modern solvers

Clause deletion

- Activity-based: smoothing BerkMin's scheme
 - Each clause is associated with a float activity
 - Each time a clause is used in conflict analysis, its activity is increased
 - Periodically, the less active clauses are deleted (half of the clauses)
 - Still in use (along with other ideas)

MiniSat & SatELite: Seminal Works

Niklas Eén, Niklas Sörensson: An Extensible SAT-solver. SAT 2003: 502-518

- Minisat solver
- Simple & elegant engineering: the ancestor of a long line of solvers!
- Impactful heuristics & algorithms:
 - Exponential VSIDS (EVSIDS)
 - sometimes still called VSIDS today
 - Conflict clause minimization **After presenting SatELite**
- Incremental-under-assumptions API: enabler of major real-world flows
- “Test of time” award at SAT’22

Niklas Eén, Armin Biere: Effective Preprocessing in SAT Through Variable and Clause Elimination. SAT 2005: 61-75

- SatELite Preprocessing **Next: after some preliminary material**

Resolution and Variable Elimination

I is the pivot

$$\left. \begin{array}{l} C \vee I \\ D \vee \neg I \end{array} \right\} \rightarrow C \vee D$$

Resolution *(Davis & Putnam'60)*

$$\left. \begin{array}{l} a \vee b \vee I \\ c \vee b \vee \neg I \end{array} \right\} \rightarrow a \vee b \vee c$$

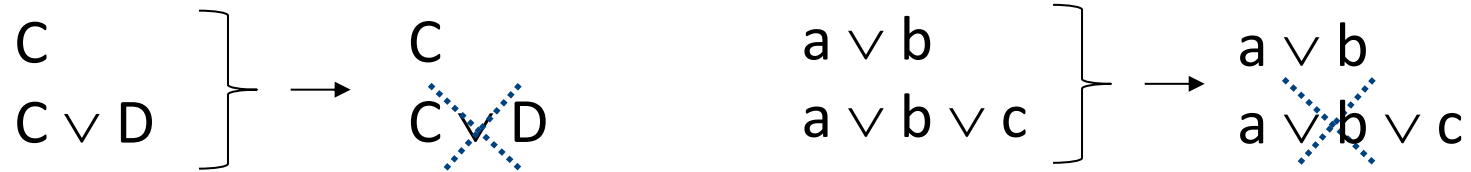
Variable Elimination by Resolution *(Davis & Putnam'60)*

$$\left. \begin{array}{l} C_1 \vee I \\ D_1 \vee \neg I \\ C_2 \vee I \\ D_2 \vee \neg I \\ \dots \\ C_n \vee I \\ D_m \vee \neg I \end{array} \right\} \xrightarrow{\text{Equisatisfiable}} \begin{array}{l} C_1 \vee D_1 \\ C_1 \vee D_2 \\ \dots \\ C_1 \vee D_m \\ \dots \\ C_2 \vee D_1 \\ C_2 \vee D_2 \\ \dots \\ C_2 \vee D_m \end{array} \quad \begin{array}{l} \dots \\ C_n \vee D_1 \\ C_n \vee D_2 \\ \dots \\ C_n \vee D_m \end{array}$$

$$\left. \begin{array}{l} a \vee b \vee I \\ d \vee I \\ c \vee b \vee \neg I \\ \neg c \vee \neg b \vee \neg I \end{array} \right\} \rightarrow \begin{array}{l} a \vee b \vee c \\ c \vee b \vee d \\ \neg c \vee \neg b \vee d \end{array}$$

Subsumption and Self-Subsuming Resolution

Subsumption



Self-subsuming Resolution



SatELite Preprocessor

Run the following till fixed-point

- Variable elimination
 - Bounded: the number of clauses doesn't grow or grows by a constant factor
 - Gate identification: next slide
- Subsumption for removing subsumed clauses:
- Self-subsuming resolution for removing literals:

Crucial on many difficult instances ever since 2005!

$$\left. \begin{array}{l} a \vee b \vee l \\ d \vee l \\ c \vee b \vee \neg l \\ \neg c \vee \neg b \vee \neg l \end{array} \right\} \rightarrow \begin{array}{l} a \vee b \vee c \\ c \vee b \vee d \\ \neg c \vee \neg b \vee d \end{array}$$

$$\left. \begin{array}{l} a \vee b \\ a \vee b \vee c \end{array} \right\} \rightarrow \begin{array}{l} a \vee b \\ \cancel{a \vee b \vee c} \end{array}$$

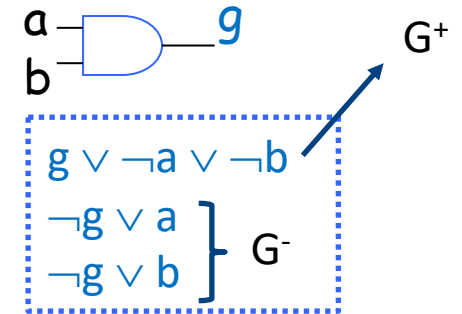
$$\left. \begin{array}{l} a \vee b \vee l \\ b \vee \neg l \end{array} \right\} \rightarrow \begin{array}{l} a \vee b \\ b \vee \neg l \end{array}$$

SatELite with Gate Identification

Idea: reduce the number of generated resolvents when eliminating a variable


How (on an AND-gate example; applicable to other gates too):

- Assume the algorithm considers eliminating the variable g
- Was g created by translating an AND-gate to clauses?
 - Are the following clauses (or their simplified variants) present:
- Negative \rightarrow optimization can't be applied. Positive:
- Let the gate definition clauses be $G = G^+ \cup G^-$ and the rest be $R = R^+ \cup R^-$
 - + clauses contain g ; - clauses contain $\neg g$
- Create only the resolvents between G and R !
 - Resolving between G^+ and G^- yield tautologies
 - Resolving between R^+ and R^- is unnecessary \rightarrow next slide

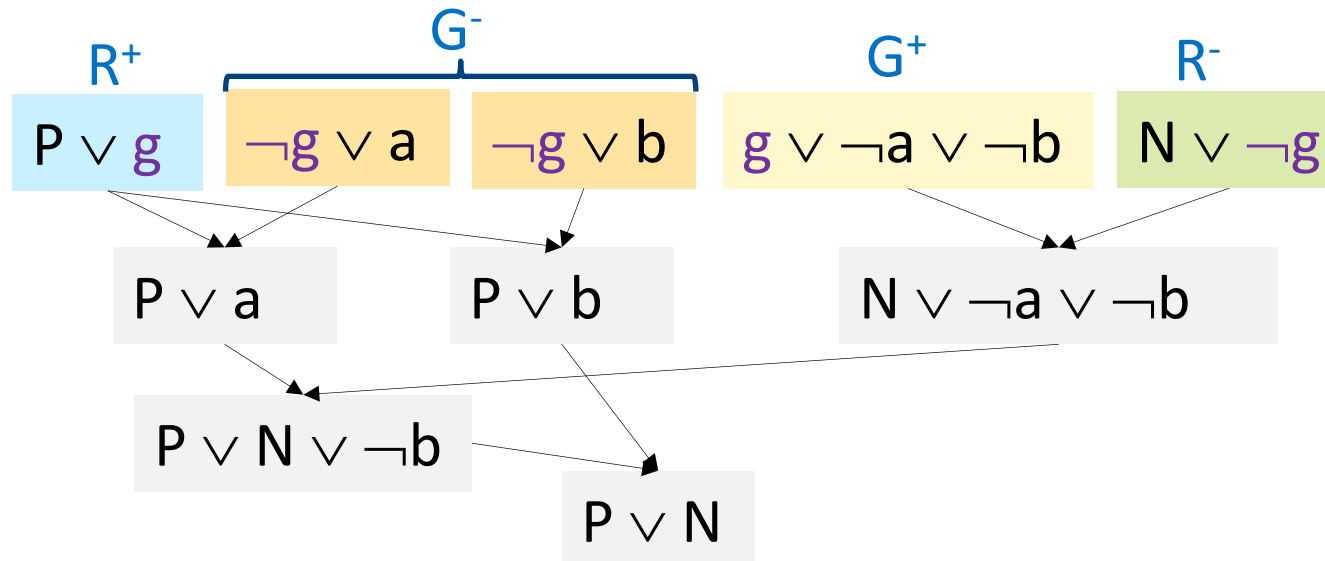
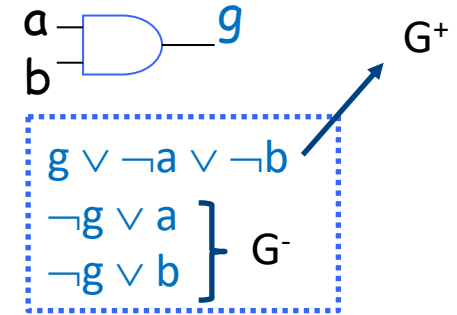


SatELite with Gate Identification: Cont.

Resolving between R^+ and R^- is unnecessary: the resolvents are obsolete

$$\left. \begin{array}{l} P \vee g \in R^+ \\ N \vee \neg g \in R^- \end{array} \right\} \rightarrow P \vee N$$


It can be yielded solely by resolutions between G and R :



(Learned Clause) Minimization

The idea:

- During conflict analysis,
- given a learnt clause C,
- remove unnecessary literals from C by resolution with parent clauses

Local minimization

Beame, P., Kautz, H., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. J. Artif. Intell. Res. (JAIR) 22 (2004)

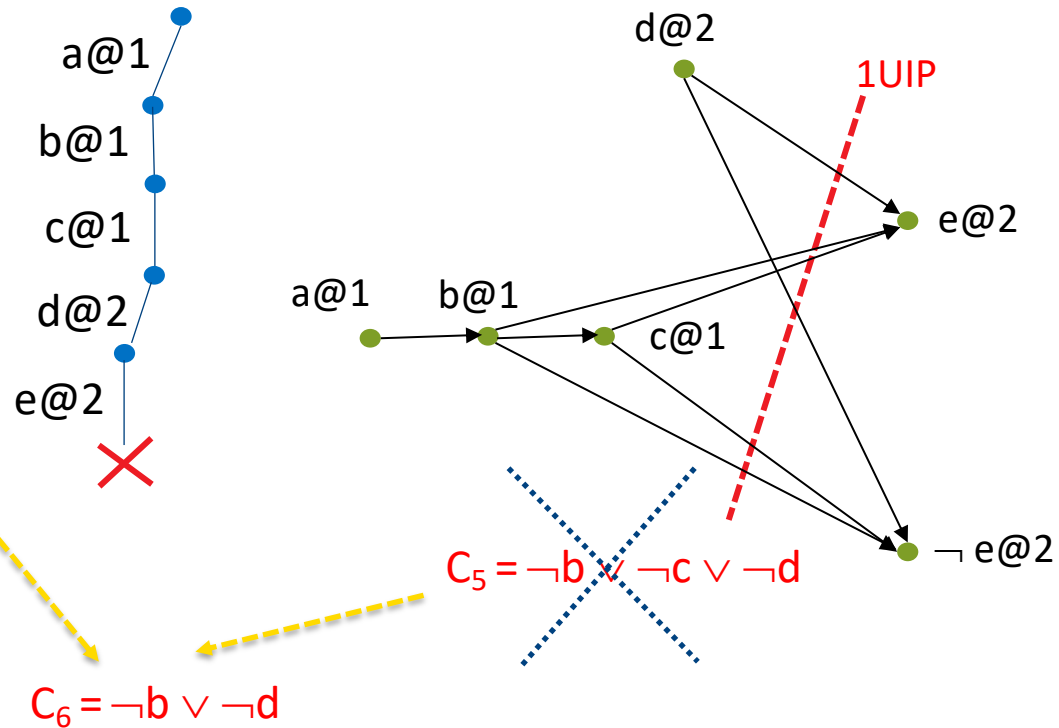
Recursive minimization

Niklas Sörensson, Armin Biere: Minimizing Learned Clauses. SAT 2009: 237-243

- In MiniSat since 2005
- Standard nowadays, applied for every learnt clause

Local Minimization

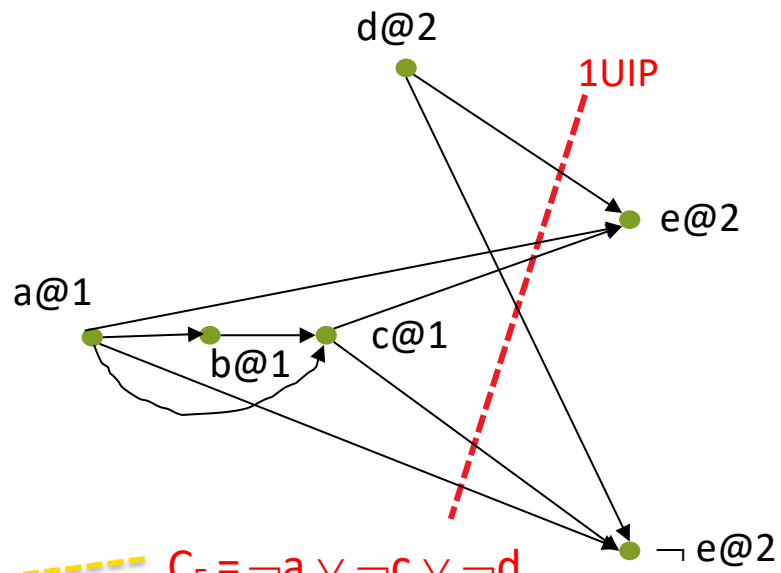
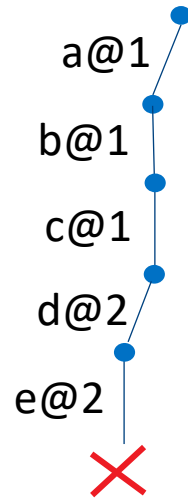
- $C_1 = \neg a \vee b$
- $C_2 = \neg b \vee c$
- $C_3 = \neg b \vee \neg c \vee \neg d \vee e$
- $C_4 = \neg b \vee \neg c \vee \neg d \vee \neg e$



- Given a newly learnt clause C , remove literals, whose antecedents (in the implication graph) are already in C
- By applying self-subsuming resolution with the parent clause

Local Minimization Shortcoming

- $C_1 = \neg a \vee b$
- $C_2 = \neg a \vee \neg b \vee c$
- $C_3 = \neg a \vee \neg c \vee \neg d \vee e$
- $C_4 = \neg a \vee \neg c \vee \neg d \vee \neg e$



$C_6 = \neg a \vee \neg b \vee \neg d$ $C_5 = \neg a \vee \neg c \vee \neg d$

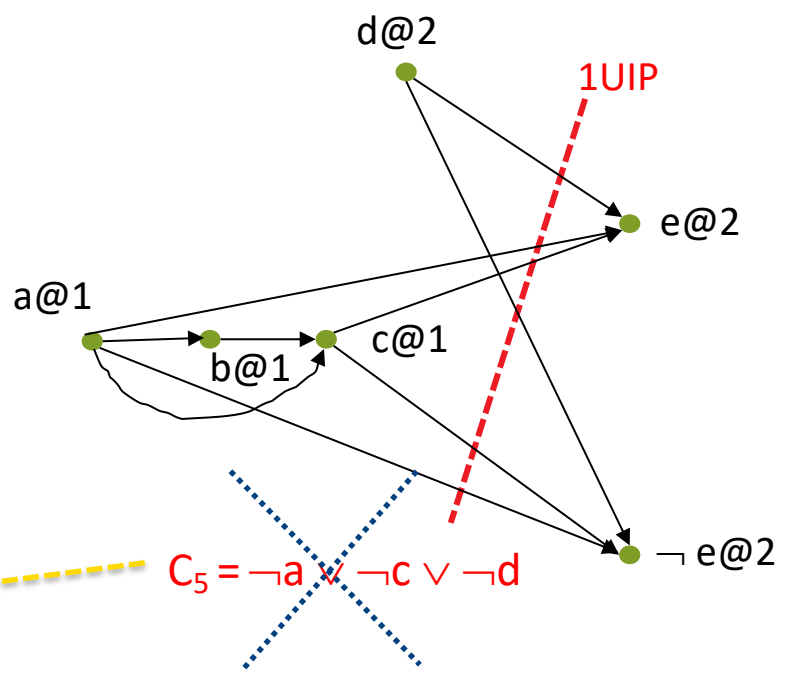
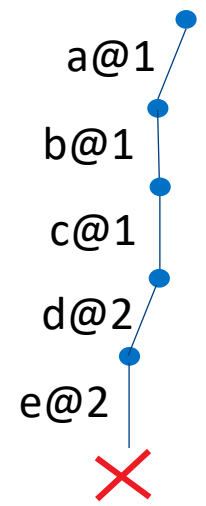
A new literal → local minimization fails

However, a further resolution step with C_1 would have yielded $C_7 = \neg a \vee \neg d$, which subsumes C_5

- Given a newly learnt clause C, remove literals, whose antecedents (in the implication graph) are already in C
- By applying self-subsuming resolution with the parent clause

Recursive Minimization

- $C_1 = \neg a \vee b$
- $C_2 = \neg a \vee \neg b \vee c$
- $C_3 = \neg a \vee \neg c \vee \neg d \vee e$
- $C_4 = \neg a \vee \neg c \vee \neg d \vee \neg e$



- $C_6 = \neg a \vee \neg b \vee \neg d$
- $C_7 = \neg a \vee \neg d$

- Given newly a learnt clause C , try to remove literals one-by-one in decreasing assignment order by continuous resolution with the parents till either:
 - A new level or a new decision variable is reached \rightarrow removal not possible
 - Literals already in the clause reached \rightarrow remove the literal from the clause

MiniSat in Non-Incremental Mode

`Preprocess()` // Variable elimination with gate identification & subsumption & self-subsuming resolution

`While (true)`

- `Literal l = Decide()` // Variable-based EVSIDS + polarity 0
- `BCP(l)`
- `If (conflict)`
 - `ConflictAnalysisLoop()` // minimized 1UIP
- `If (learned an empty clause)`
 - `Return UNSAT`
- `If (all the variables are assigned)`
 - `Return SAT`
- `Occasionally, restart` // geometric series
- `Occasionally, delete conflict clauses` // activity-based

Agenda

How does a conflict-driven SAT solver work?

- The core: backtrack search, Boolean Constraint Propagation (BCP), conflict analysis
- Follow the first SAT Competition winners (from Chaff till Minisat)

Applying SAT by example

- Paradigms: incremental SAT solving, SAT-based local search, example encodings
- Applications: Bounded Model Checking (BMC), proof-based abstraction refinement, bug hunting, anytime MaxSAT

Advanced core SAT algorithms

- Follow the SAT Competition winners after Minisat & discussion about incremental SAT

Hardware Circuit Example

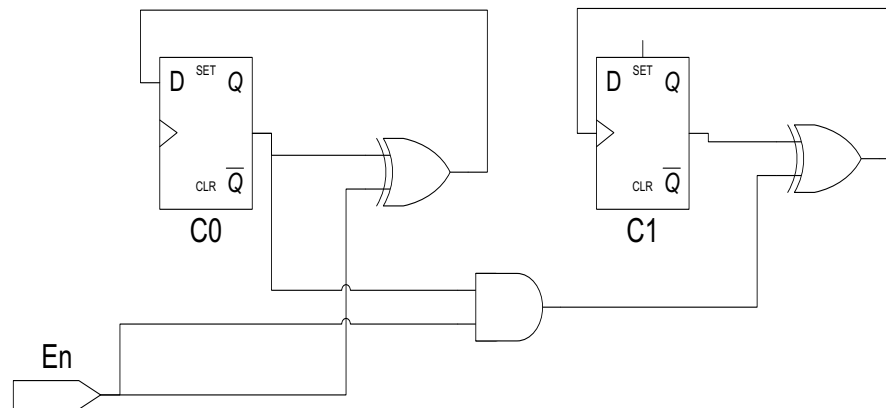
definitions:

$I(C_0) := \text{FALSE};$
 $I(C_1) := \text{FALSE};$
 $X(C_0) := C_0 \oplus \text{en};$
 $X(C_1) := C_1 \oplus (C_0 \wedge \text{en});$

constraints:

$\text{en} \rightarrow \neg X(\text{en})$

2-Bit Counter, counting when en=1



Cycle	en	C ₁	C ₀
0	1	0	0
1	0	0	1
2	1	0	1
3	0	1	0
4	1	1	0
5	0	1	1

Hardware Model Checking

Model Checking: given a circuit and a property, does the property always hold?

- Safety property: something bad will never happen
 - Example: the counter never reaches the value 11

definitions:

$I(C_0) := \text{FALSE};$
 $I(C_1) := \text{FALSE};$
 $X(C_0) := C_0 \oplus \text{en};$
 $X(C_1) := C_1 \oplus (C_0 \wedge \text{en});$

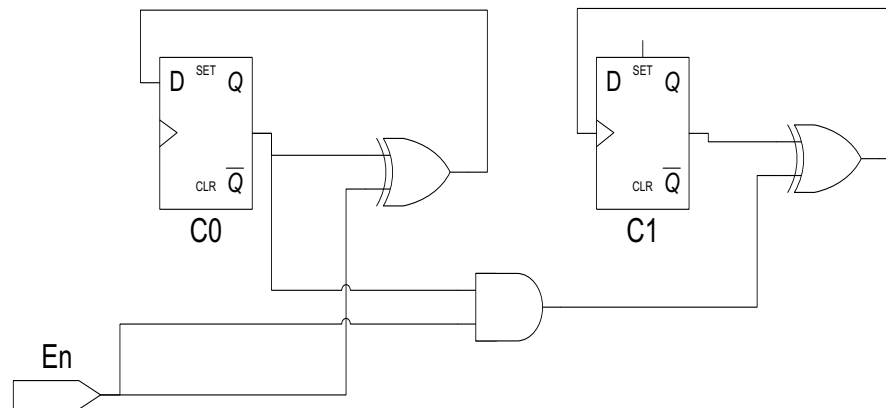
constraints:

$\text{en} \rightarrow \neg X(\text{en})$

proof obligations:

$\neg(C_0 \wedge C_1)$

2-Bit Counter, counting when en=1



Cycle	en	C ₁	C ₀
0	1	0	0
1	0	0	1
2	1	0	1
3	0	1	0
4	1	1	0
5	0	1	1

Hardware Model Checking

Model Checking: given a circuit and a property, does the property always hold?

- Safety property: something bad will never happen
 - Example: the counter never reaches the value 11

FALSE

2-Bit Counter, counting when en=1

definitions:

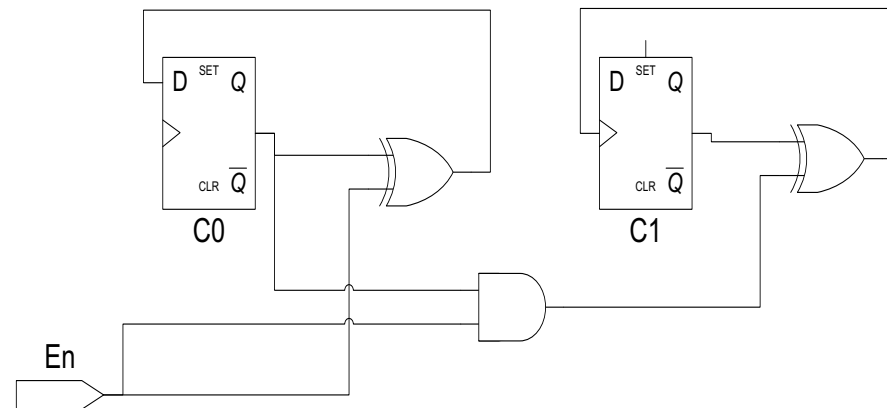
$I(C_0) := \text{FALSE};$
 $I(C_1) := \text{FALSE};$
 $X(C_0) := C_0 \oplus \text{en};$
 $X(C_1) := C_1 \oplus (C_0 \wedge \text{en});$

constraints:

$\text{en} \rightarrow \neg X(\text{en})$

proof obligations:

$\neg(C_0 \wedge C_1)$



Cycle	en	C ₁	C ₀
0	1	0	0
1	0	0	1
2	1	0	1
3	0	1	0
4	1	1	0
5	0	1	1

Bounded Model Checking (BMC)

BMC: given a circuit φ and a property P , verify P until a user-given bound n

CAV Award

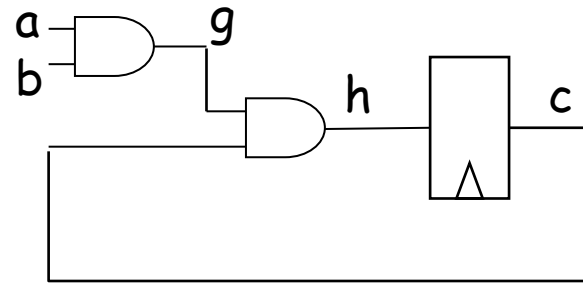
The recipients of the 2018 CAV Award are:

- Armin Biere
- Alessandro Cimatti
- Edmund M. Clarke
- Daniel Kroening
- Flavio Lerda
- Yunshan Zhu

for their **Outstanding contribution to the enhancement and scalability of model checking by introducing Bounded Model Checking based on Boolean Satisfiability (SAT) for hardware (BMC) and software (CBMC)**

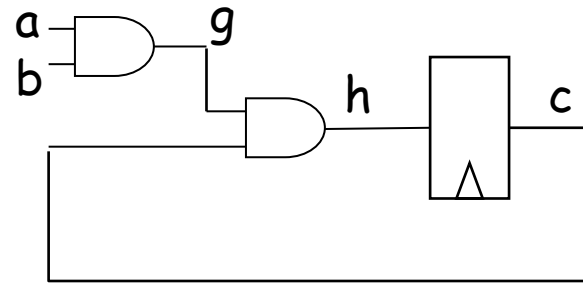
- For every bound $b \in \{1, \dots, n\}$
 - The property holds at bound b iff $(\varphi \text{ unrolled to bound } b) \wedge \neg P$ is UNSAT
 - If it's SAT, the model comprises the trace of a bug

BMC Example

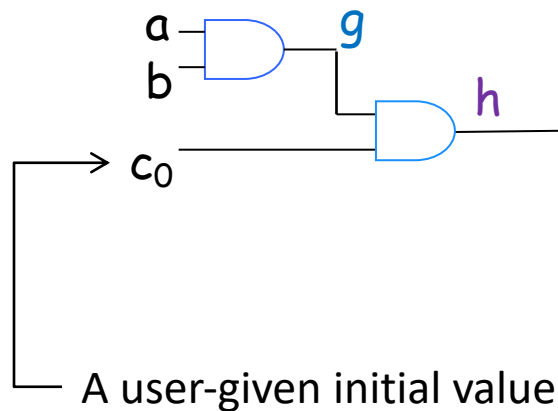


The property: $h \rightarrow b$

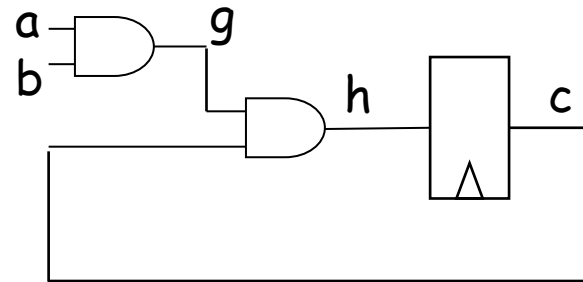
BMC Example: Cycle 0



The property: $h \rightarrow b$

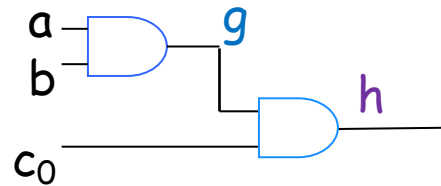


BMC Example: Cycle 0



The property: $h \rightarrow b$

$g \vee \neg a \vee \neg b$
 $\neg g \vee a$
 $\neg g \vee b$



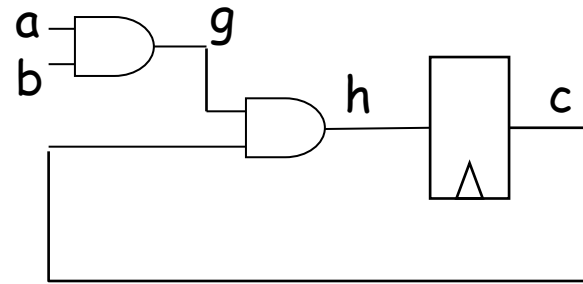
$h \vee \neg g \vee \neg c_0$
 $\neg h \vee g$
 $\neg h \vee c_0$

The negation of the property $h \rightarrow b$:

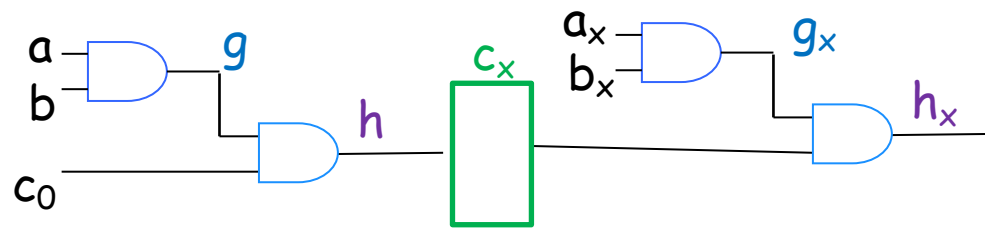
$\neg b$
 h

UNSAT: the property holds!

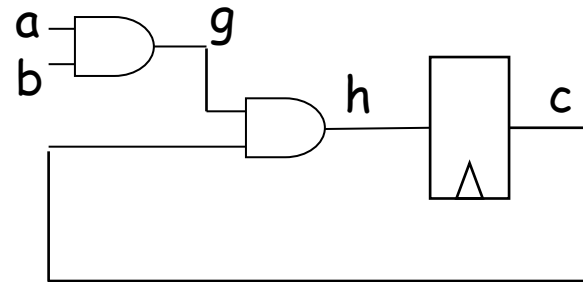
BMC Example: Cycle 1



The property: $h \rightarrow b$

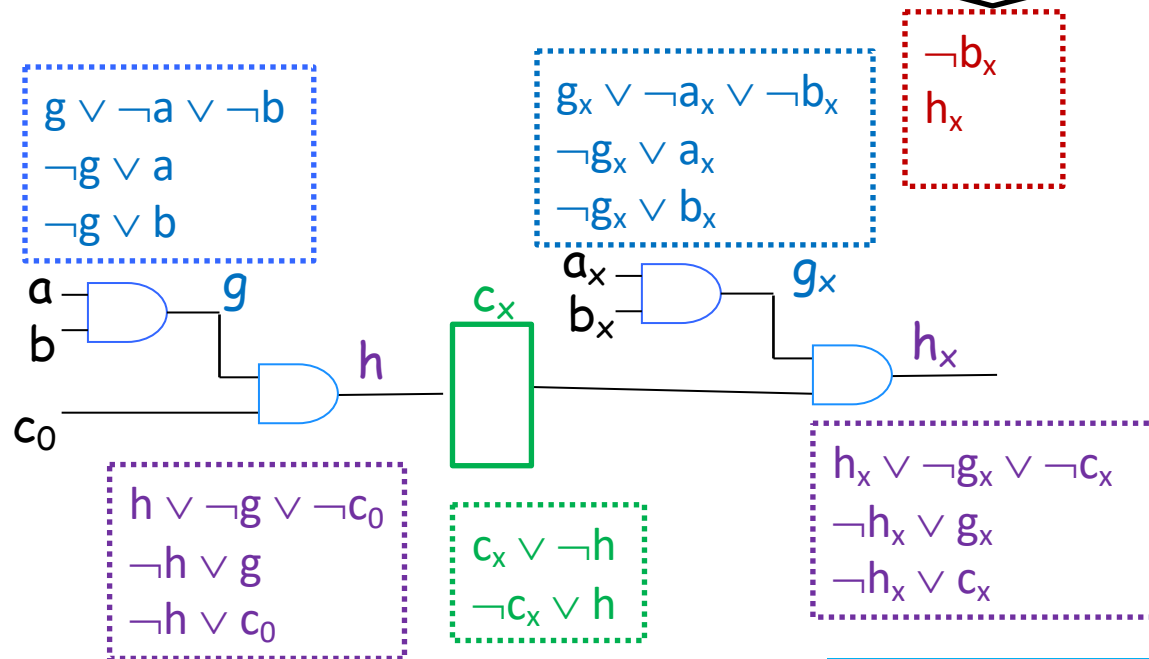


BMC Example: Cycle 1



The property: $h \rightarrow b$

The negation of the property $h_x \rightarrow b_x$:



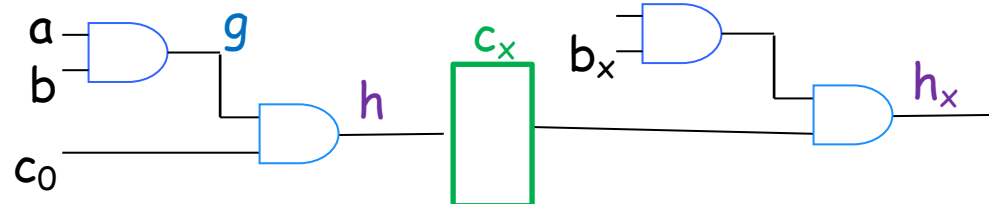
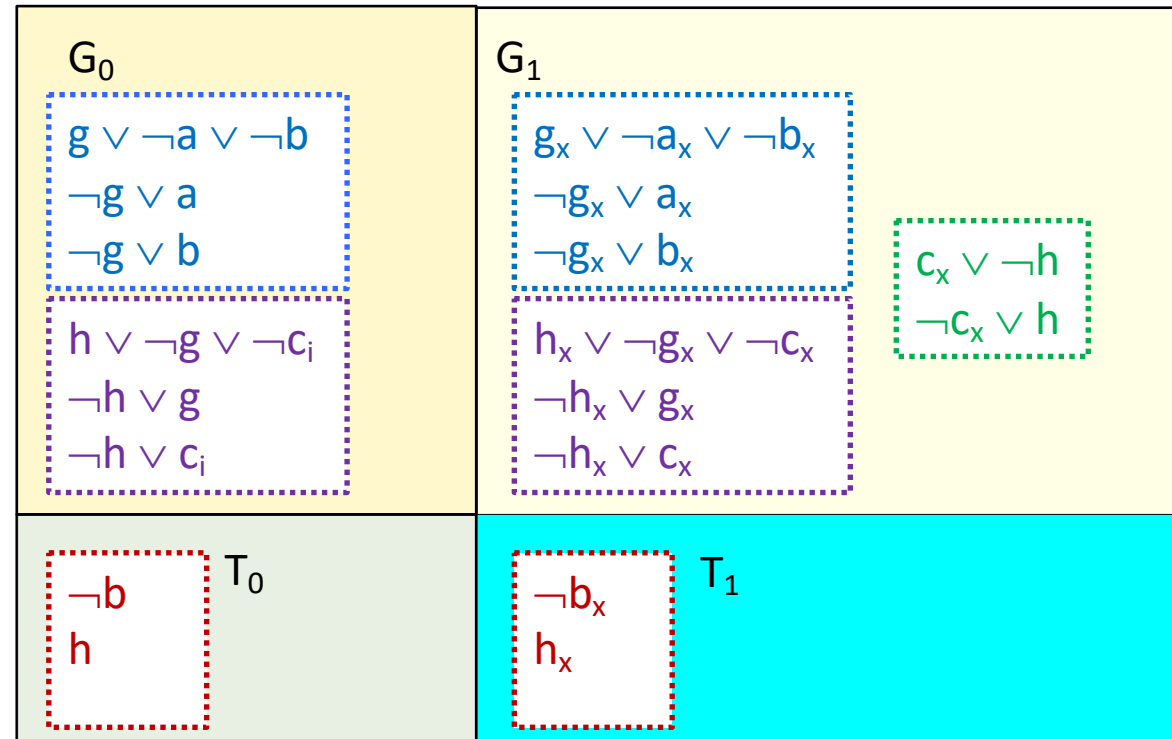
UNSAT!

Re-Using Relevant Information from Previous Cycles

G_0 and G_1 : hold globally

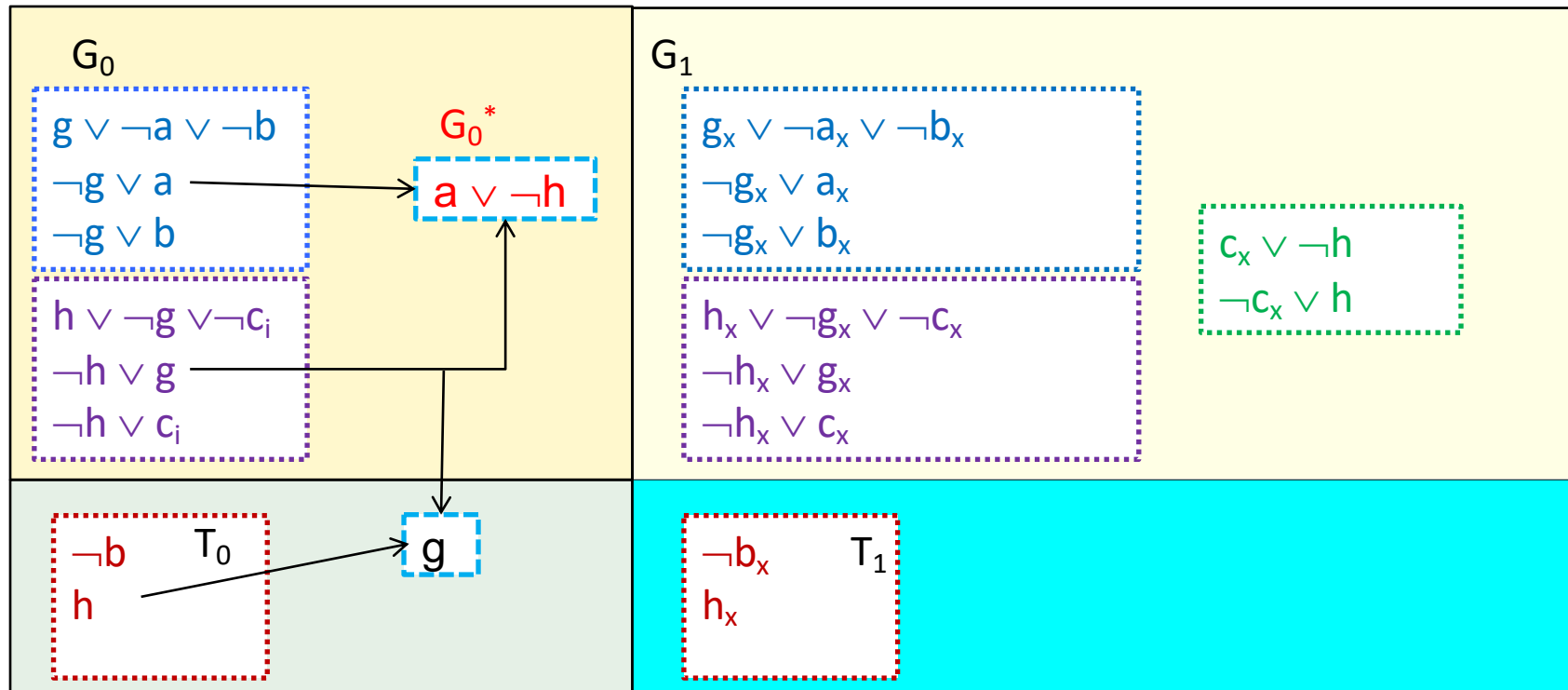
T_0 and T_1 : hold temporary

- solely for a particular cycle



The property: $h \rightarrow b$

Pervasive Clause Learning (*GRASP*)

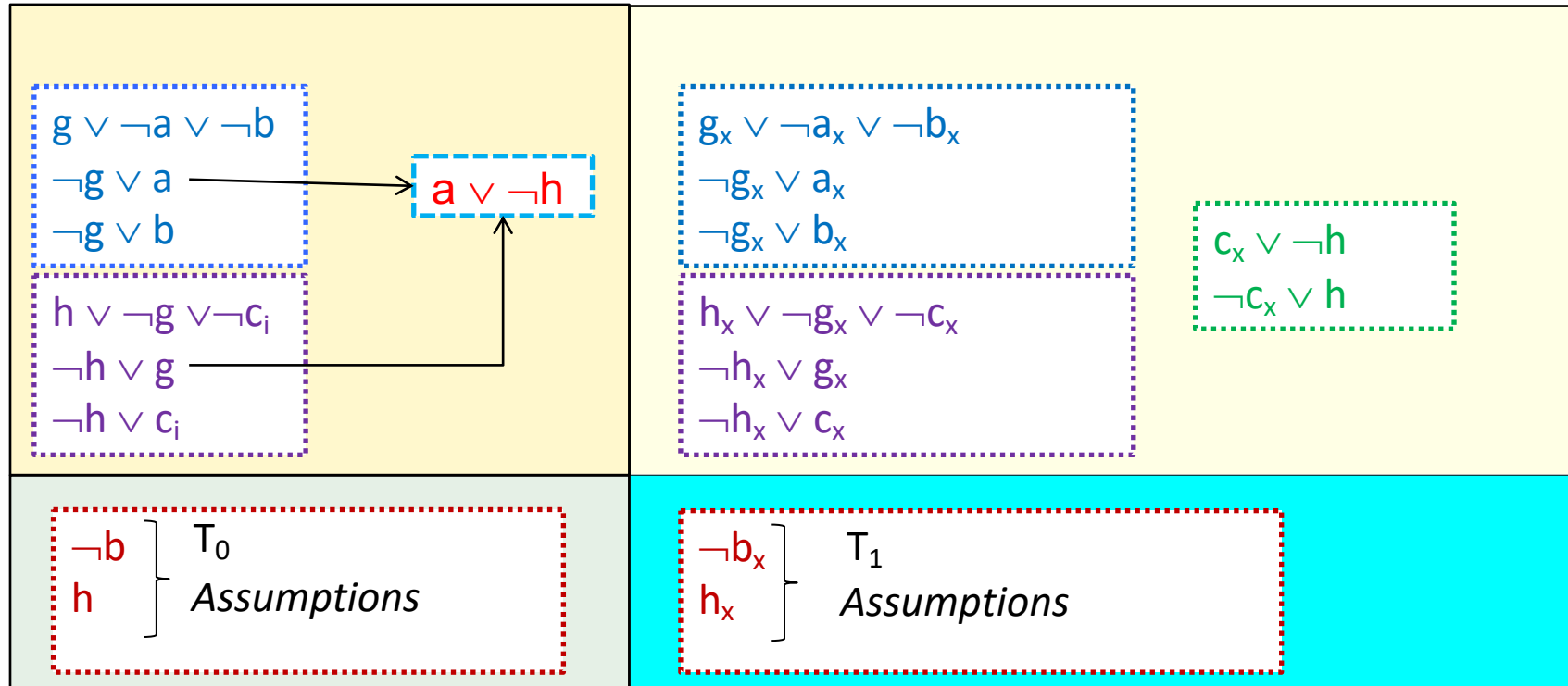


Cycle 0: create a SAT instance $G_0 \wedge T_0$ and solve it

- Let G_0^* be the set of **pervasive conflict clauses**, that is conflict clauses that depend only on G_0

Cycle 1: create a SAT instance $G_0 \wedge G_0^* \wedge G_1 \wedge S_1$ and solve it

Incremental SAT Solving under Assumptions (*Minisat*)



Cycle 0: create a SAT instance and solve it under the temporary assumptions T_0

- T_0 clauses are not part of the instance, instead:
- The literals of T_0 are used as the first decision, or assumptions
- Any learnt clause which depends on an assumption a , will contain a itself or derived literals \rightarrow all the learnts are pervasive!
- If one of the assumptions must be flipped, the solver returns UNSAT

Cycle 1: add the clauses C_1 to the same SAT instance and solve under the assumptions T_1

Incremental SAT Solving under Assumptions

Basic API:

- AddClause(Clause C)
- Solve(Literals A)

Output:

- SAT iff $F \wedge A$ is SAT
 - F: all the clauses added so far
 - A: current assumptions

Allows the user to add groups of clauses temporarily (for invocation #i)

- Using a new **selector (activation)** variable s
- To add a temporary clause T_i : AddClause($T_i \vee s$)
- Solve($A \wedge \neg s$)
- To delete all the temporary clauses afterwards: do nothing or add a unit clause s

Incremental SAT Solving under Assumptions

A breakthrough

- Extremely easy to use
- Very efficient, since it retains the following info across all the queries:
 - the conflict clauses
 - heuristical data: variable scores, clause activities, ...

Incremental solving under assumptions is widely used, including:

- Hardware & Software Validation
- SAT-based Optimization, including MaxSAT

My personal industrial experience: can't recall any non-incremental SAT application

Minisat: SatELite preprocessing is incompatible with incremental solving

- Later works make it compatible:

*Alexander Nadel, Vadim Ryvchin, Ofer Strichman: **Preprocessing in Incremental SAT.** SAT 2012: 256-269*
*Katalin Fazekas, Armin Biere, Christoph Scholl: **Incremental Inprocessing in SAT Solving.** SAT 2019: 136-154*

Unsatisfiable Core in Terms of Assumptions

Extended API:

- AddClause(Clause C)
- Solve(Literals A)
- If UNSAT:
 - Is assumption $l \in A$ required for unsatisfiability proof?

Algorithm outline

- Conflict at decision level 0: unrelated to the assumptions
 - None of the assumptions is required
- Otherwise, an assumption must have been flipped
 - Go over the trail backwards
 - Mark all the decision variables (must be assumptions!), connected to the flipped assumption
 - Return the marked assumptions

```

/*
analyzeFinal : (p : Lit) -> [void]
Description:
Specialized analysis procedure to express the final conflict in terms of assumptions.
Calculates the (possibly empty) set of assumptions that led to the assignment of 'p', and
stores the result in 'out_conflict'.
@*/

```

```

void Solver::analyzeFinal(Lit p, vec <Lit> &out_conflict) {
out_conflict.clear();
out_conflict.push(p);
if(decisionLevel() == 0)
return;

```

p: the flipped assumption
out_conflict: the returned set of assumptions in the core

```

seen[var(p)] = 1;

```

seen: marks variables connected to the flipped assumption

```

for(int i = trail.size() - 1; i >= trail_lim[0]; i--) {
Var x = var(trail[i]);
if(seen[x]) {
if(reason(x) == CRef_Undef) {
out_conflict.push(~trail[i]);
} else {
Clause &c = ca[reason(x)];
for(int j = ((c.size() == 2) ? 0 : 1); j < c.size(); j++)
if(level(var(c[j])) > 0)
seen[var(c[j])] = 1;
}
}
seen[x] = 0;
}
}

```

Go over the trail backwards

A decision variable: must be an assumption, since the flip must have occurred at an assumption level

An implied variable: mark all the variables in its parent

```

seen[x] = 0;
}
}

```

```

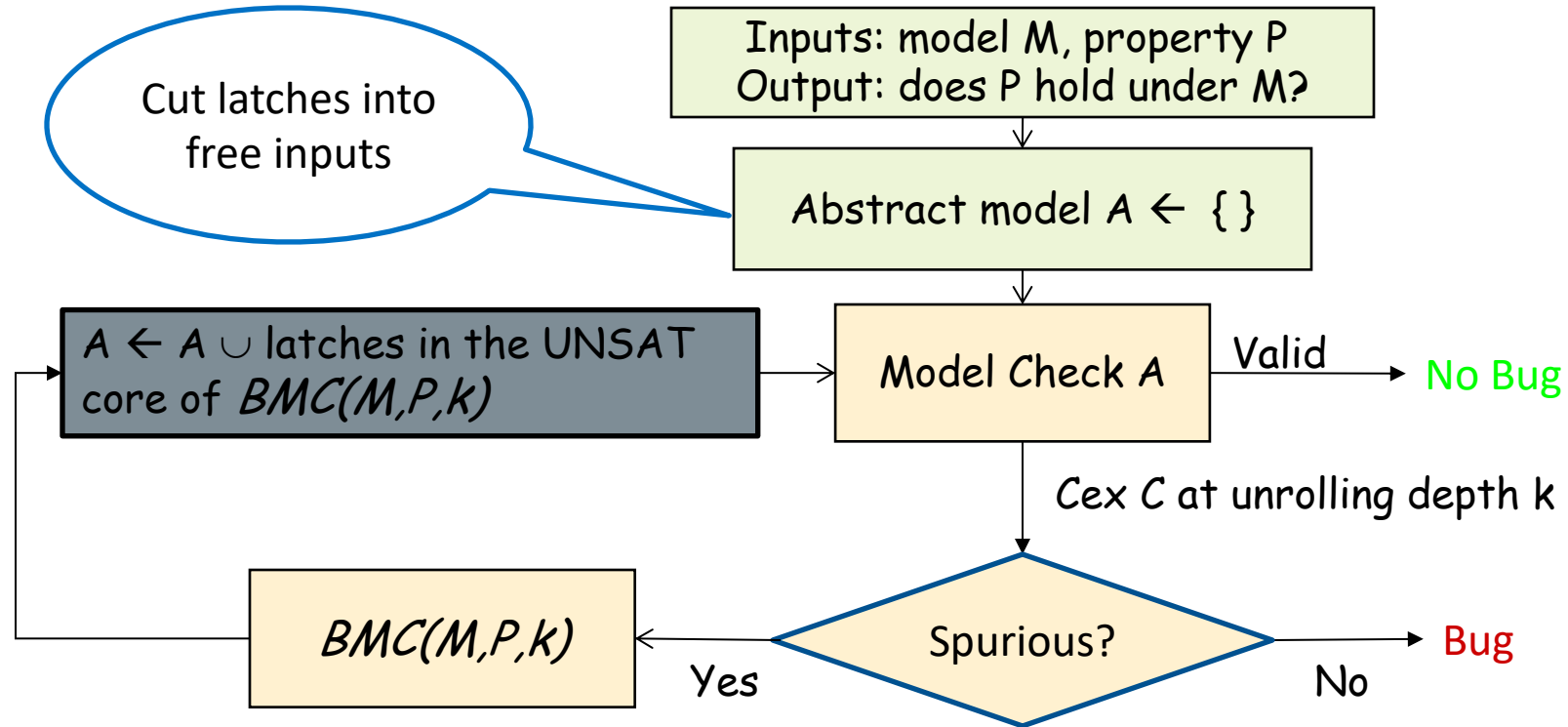
seen[var(p)] = 0;
}

```

Proof-based Abstraction Refinement Example

Kenneth L. McMillan, Nina Amla: *Automatic Abstraction without Counterexamples*. *TACAS 2003*: 2-17

Aarti Gupta, Malay K. Ganai, Zijiang Yang, Pranav Ashar: *Iterative Abstraction using SAT-based BMC with Proof Analysis*. *ICCAD 2003*: 416-423



The unsatisfiable core in terms of the latches is required

SAT-based Local Search: Finding a Solution Near an Assignment

Find a solution *near* an assignment $M = \{v_1 = \sigma_1, v_2 = \sigma_2, \dots, v_n = \sigma_n\}$

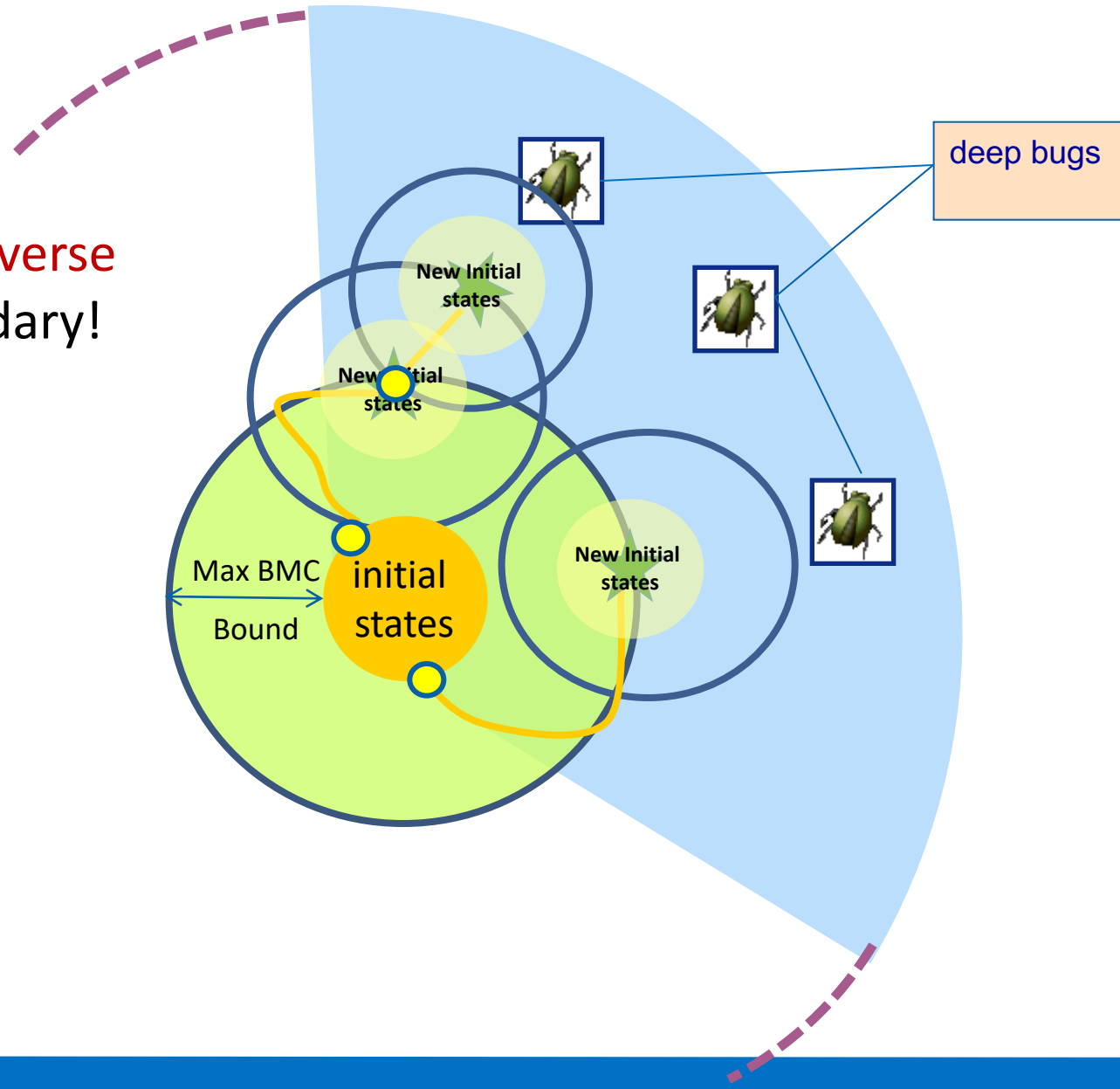
Polarity-based

Sabih Agbaria, Dan Carmi, Orly Cohen, Dmitry Korchemny, Michael Lifshits, Alexander Nadel:
SAT-based semiformal verification of hardware. FMCAD 2010: 25-32

- Change only the polarity selection heuristic
- Whenever a decision variable v_i is chosen, choose σ_i as its first polarity

Original Application: Bug Hunting

Needed to generate **diverse** solutions on the boundary!



Original Application: DiverseKSet for Bug Hunting

DiverseKSet in SAT: generate a user-given number of **diverse** solutions

- Maximize the average Hamming distance between the solutions

Diverse solutions, given an empty CNF

- 000000
 - Any model
- 111111
 - Flip every variable
- 001110
 - Pick a random value
- 110001
 - Balance the values for every variable

Given a CNF formula:

- **SAT-based local search:** fix the polarities to match the current target solution (which would maximize the Hamming distance)
- Run SAT incrementally
- Adjust the target solution

Optimization in SAT

OptSAT(F, ψ): given a propositional formula F in CNF and a **Pseudo-Boolean objective function** ψ , return a model to F which **minimizes** ψ

- A Pseudo-Boolean (PB) function: a mapping from every full assignment to a real number

Example: $F = (a + b) (a + \neg c) (\neg a + c)$

H has 3 models:

- $M_1 = \{a=0, b=1, c=0\}$
- $M_2 = \{a=1, b=0, c=1\}$
- $M_3 = \{a=1, b=1, c=1\}$

a	b	c	ψ
0	0	0	2.3
0	0	1	3.5
0	1	0	8
0	1	1	100.1
1	0	0	96.3
1	0	1	75
1	1	0	1.35
1	1	1	20.4

Optimization in SAT

OptSAT(F, ψ): given a propositional formula F in CNF and a **Pseudo-Boolean objective function** ψ , return a model to F which **minimizes** ψ

- A Pseudo-Boolean (PB) function: a mapping from every full assignment to a real number

Example: $F = (a + b) (a + \neg c) (\neg a + c)$

H has 3 models:

- $M_1 = \{a=0, b=1, c=0\}$
- $M_2 = \{a=1, b=0, c=1\}$
- $M_3 = \{a=1, b=1, c=1\}$

Best
model

a	b	c	ψ
0	0	0	2.3
0	0	1	3.5
0	1	0	8
0	1	1	100.1
1	0	0	96.3
1	0	1	75
1	1	0	1.35
1	1	1	20.4

Solving OptSAT(F, ψ) Instances in Real-life

Is ψ is a **linear** PB function: $\psi = w_{n-1} * t_{n-1} + \dots + w_1 * t_1 + \dots + w_0 * t_0$?

- t_i 's are Boolean variables
- w_i 's are strictly positive integer coefficients
- Example: $\psi = 2 * t_2 + 5 * t_1 + 7 * t_0$

Yes

No

- MaxSAT
 - A well-established field
 - Myriads of applications
 - MaxSAT Evaluations since 2006

- Few works
- Industrial usage @ Intel

MaxSAT: Optimizing a Linear PB Function in SAT

Input: Hard Clauses H
- Satisfiable

Optimization **Target** $T = \{t_{n-1}, t_{n-2}, \dots, t_0\}$
- Each **target bit** t_i is a literal (unit clause), associated with an integer weight $w(t_i) > 0$

Output: A model M to H which **minimizes** the **weight** of the **satisfied** target bits $\psi = w_{n-1} * t_{n-1} + \dots + w_1 * t_1 + \dots + w_0 * t_0$

Unweighted MaxSAT: All the weights are 1

MaxSAT: Optimizing a Linear PB Function in SAT

Input: Hard Clauses H
- Satisfiable

Optimization **Target** $T = \{t_{n-1}, t_{n-2}, \dots, t_0\}$

- Each **target bit** t_i is a literal (unit clause), associated with an integer weight $w(t_i) > 0$

Output: A model M to H which **minimizes** the **weight** of the **satisfied** target bits $\psi = w_{n-1} * t_{n-1} + \dots + w_1 * t_1 + \dots + w_0 * t_0$

Unweighted MaxSAT: All the weights are 1

Example: $H = (a + b) (a + \neg c) (\neg a + c)$; $T = \{a, b\}$

H has 3 models:

- $M_1 = \{a=0, b=1, c=0\}$
- $M_2 = \{a=1, b=0, c=1\}$
- $M_3 = \{a=1, b=1, c=1\}$

MaxSAT: Optimizing a Linear PB Function in SAT

Input: Hard Clauses H
- Satisfiable

Optimization **Target** $T = \{t_{n-1}, t_{n-2}, \dots, t_0\}$
- Each **target bit** t_i is a literal (unit clause), associated with an integer weight $w(t_i) > 0$

Output: A model M to H which **minimizes** the **weight** of the **satisfied** target bits $\psi = w_{n-1} * t_{n-1} + \dots + w_1 * t_1 + \dots + w_0 * t_0$

Unweighted MaxSAT: All the weights are 1

Example: $H = (a + b) (a + \neg c) (\neg a + c)$; $T = \{a, b\}$

H has 3 models:

- $M_1 = \{a=0, b=1, c=0\}$
- $M_2 = \{a=1, b=0, c=1\}$
- $M_3 = \{a=1, b=1, c=1\}$

For unweighted MaxSAT, M_1 and M_2 are optimal, since:

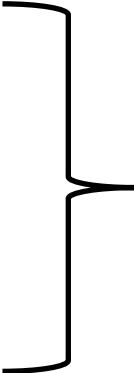
$M_1(\psi) = M_2(\psi) = 1$, while $M_3(\psi) = 2$

Exact vs. Anytime MaxSAT

Anytime algorithm: expected to find better and better solutions, the longer it keeps running

MaxSAT research can be roughly classified into two categories

- Exact
 - Guaranteed to return an optimal solution
 - No intermediate solutions are required
 - Evaluated at MaxSAT Evaluations since 2006
- Anytime
 - Not guaranteed to return an optimal solution
 - Some algorithms do
 - Improving solutions are output frequently
 - Evaluated at MaxSAT Evaluations since 2011



Handy for industrial usage
Modern solvers normally do:
Local search preprocessing
SAT-based algorithm
Next we review the core SAT-based algorithm

Linear Search SAT-UNSAT (LSU)

*Daniel Le Berre and Anne Parrain: **The sat4j library, release 2.2.** JSAT, 7(2-3):59–64, 2010.*

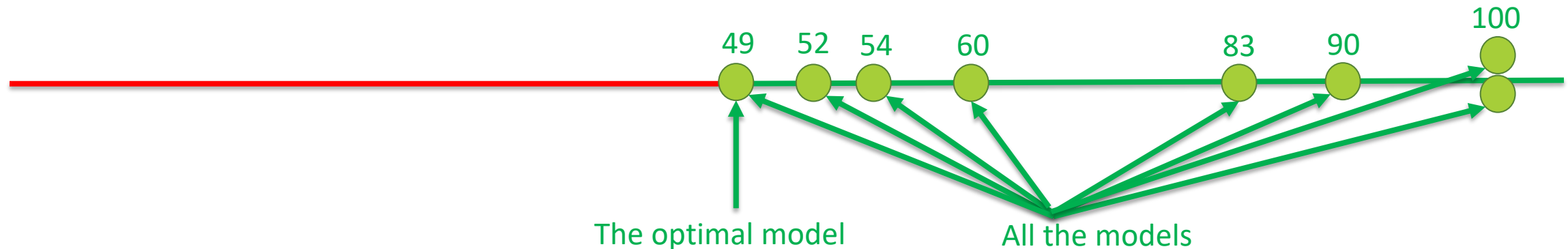
1. $F \leftarrow H$ (a CNF F is initialized with the hard clauses)
2. Run an incremental SAT solver over F
3. If SAT with a model M
 - Block all the models of weight $\geq M(\psi)$ in F (using a cardinality or a Pseudo-Boolean constraint)
4. Else (UNSAT)
 - Return the last model M (M is guaranteed to be an optimal model)

LSU is applied in leading anytime MaxSAT solvers

- During various stages, often as the last fallback option

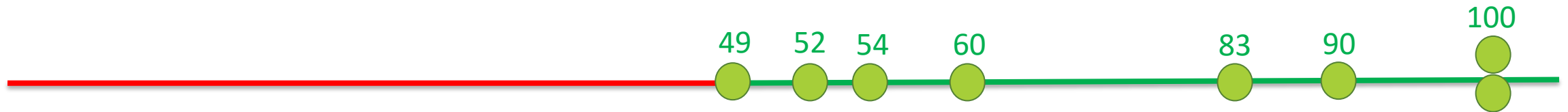
Linear Search SAT-UNSAT (LSU) Concept

1. $F \leftarrow H$ (a CNF F is initialized with the hard clauses)
2. Run an incremental SAT solver over F
3. If SAT with a model M
 - Block all the models of weight $\geq M(\psi)$ in F (using a cardinality or a Pseudo-Boolean constraint)
4. Else (UNSAT)
 - Return the last model M (M is guaranteed to be an optimal model)



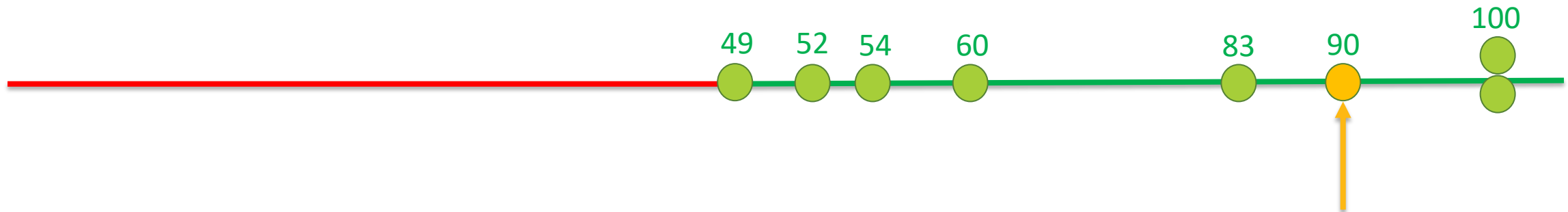
Linear Search SAT-UNSAT (LSU) Concept

1. $F \leftarrow H$ (a CNF F is initialized with the hard clauses)
2. Run an incremental SAT solver over F
3. If SAT with a model M
 - Block all the models of weight $\geq M(\psi)$ in F (using a cardinality or a Pseudo-Boolean constraint)
4. Else (UNSAT)
 - Return the last model M (M is guaranteed to be an optimal model)



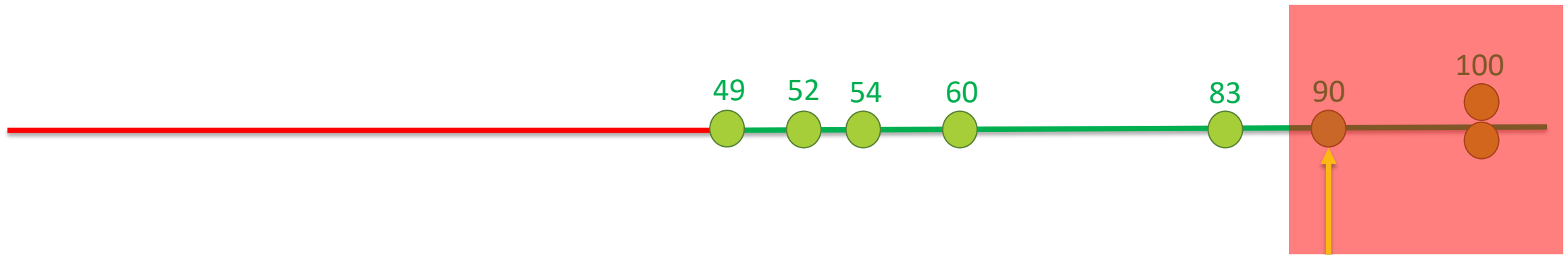
Linear Search SAT-UNSAT (LSU) Concept

1. $F \leftarrow H$ (a CNF F is initialized with the hard clauses)
2. Run an incremental SAT solver over F
3. If SAT with a model M
 - Block all the models of weight $\geq M(\psi)$ in F (using a cardinality or a Pseudo-Boolean constraint)
4. Else (UNSAT)
 - Return the last model M (M is guaranteed to be an optimal model)



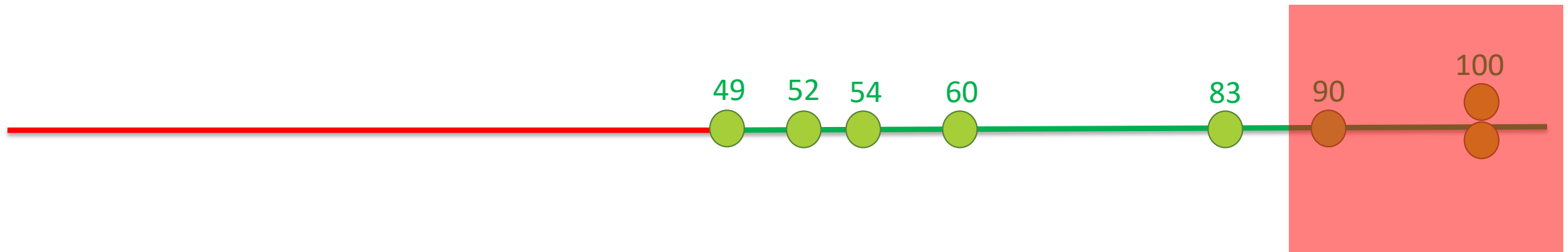
Linear Search SAT-UNSAT (LSU) Concept

1. $F \leftarrow H$ (a CNF F is initialized with the hard clauses)
2. Run an incremental SAT solver over F
3. If SAT with a model M
 - Block all the models of weight $\geq M(\psi)$ in F (using a cardinality or a Pseudo-Boolean constraint)
4. Else (UNSAT)
 - Return the last model M (M is guaranteed to be an optimal model)



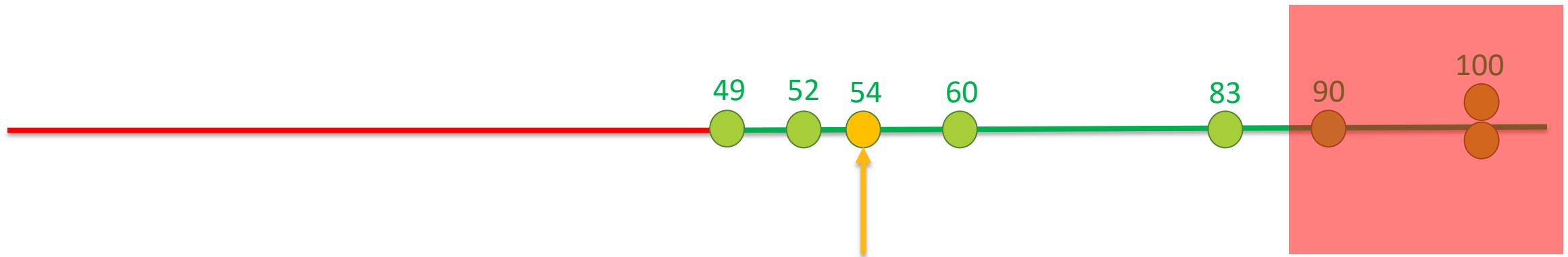
Linear Search SAT-UNSAT (LSU) Concept

1. $F \leftarrow H$ (a CNF F is initialized with the hard clauses)
2. Run an incremental SAT solver over F
3. If SAT with a model M
 - Block all the models of weight $\geq M(\psi)$ in F (using a cardinality or a Pseudo-Boolean constraint)
4. Else (UNSAT)
 - Return the last model M (M is guaranteed to be an optimal model)



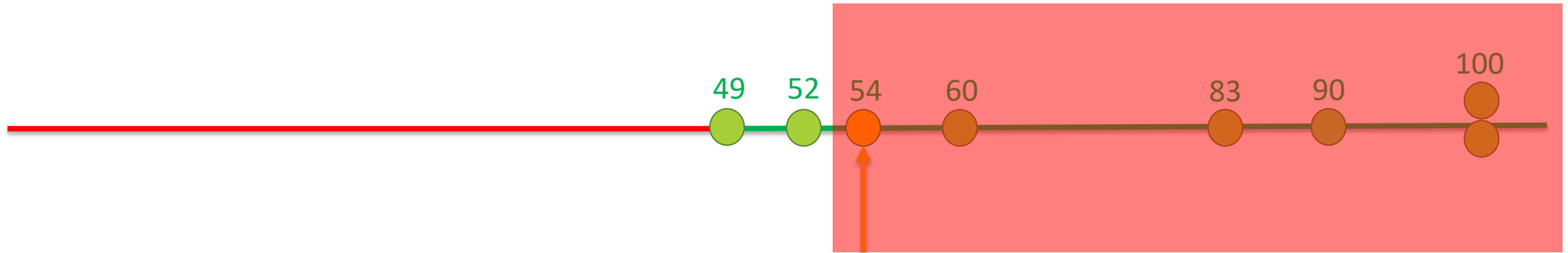
Linear Search SAT-UNSAT (LSU) Concept

1. $F \leftarrow H$ (a CNF F is initialized with the hard clauses)
2. Run an incremental SAT solver over F
3. If SAT with a model M
 - Block all the models of weight $\geq M(\psi)$ in F (using a cardinality or a Pseudo-Boolean constraint)
4. Else (UNSAT)
 - Return the last model M (M is guaranteed to be an optimal model)



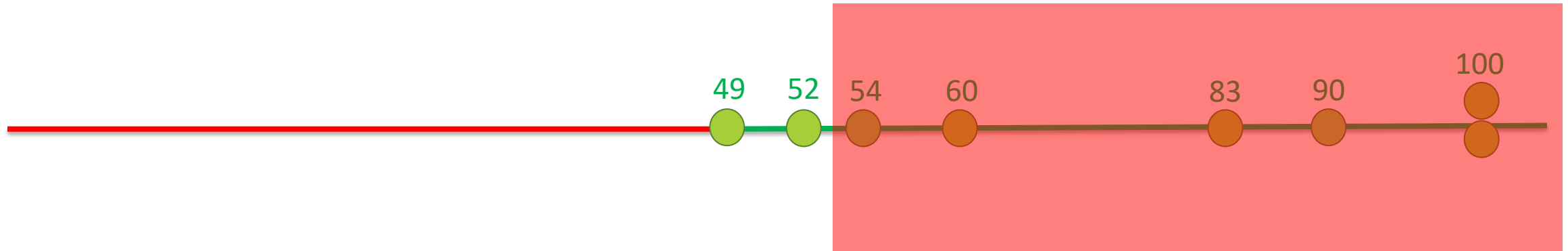
Linear Search SAT-UNSAT (LSU) Concept

1. $F \leftarrow H$ (a CNF F is initialized with the hard clauses)
2. Run an incremental SAT solver over F
3. If SAT with a model M
 - Block all the models of weight $\geq M(\psi)$ in F (using a cardinality or a Pseudo-Boolean constraint)
4. Else (UNSAT)
 - Return the last model M (M is guaranteed to be an optimal model)



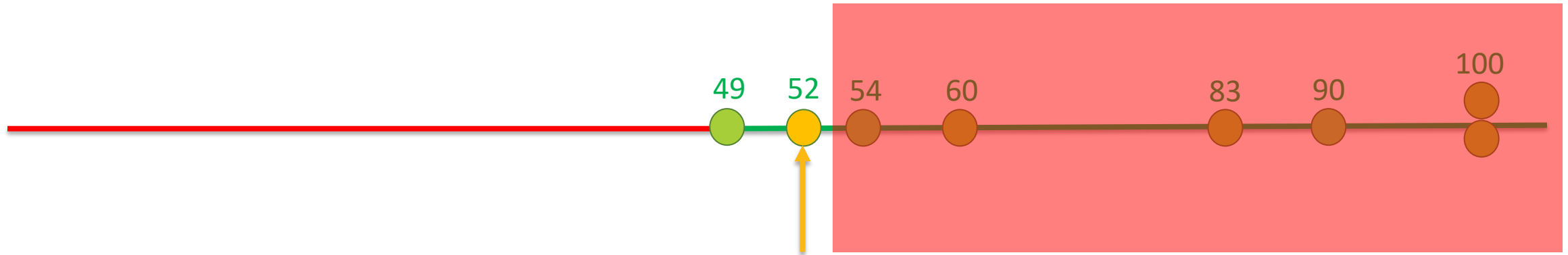
Linear Search SAT-UNSAT (LSU) Concept

1. $F \leftarrow H$ (a CNF F is initialized with the hard clauses)
2. Run an incremental SAT solver over F
3. If SAT with a model M
 - Block all the models of weight $\geq M(\psi)$ in F (using a cardinality or a Pseudo-Boolean constraint)
4. Else (UNSAT)
 - Return the last model M (M is guaranteed to be an optimal model)



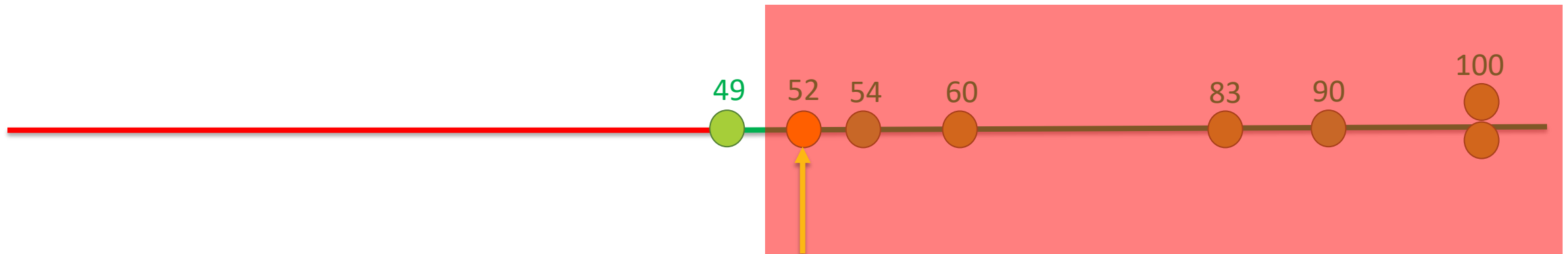
Linear Search SAT-UNSAT (LSU) Concept

1. $F \leftarrow H$ (a CNF F is initialized with the hard clauses)
2. Run an incremental SAT solver over F
3. If SAT with a model M
 - Block all the models of weight $\geq M(\psi)$ in F (using a cardinality or a Pseudo-Boolean constraint)
4. Else (UNSAT)
 - Return the last model M (M is guaranteed to be an optimal model)



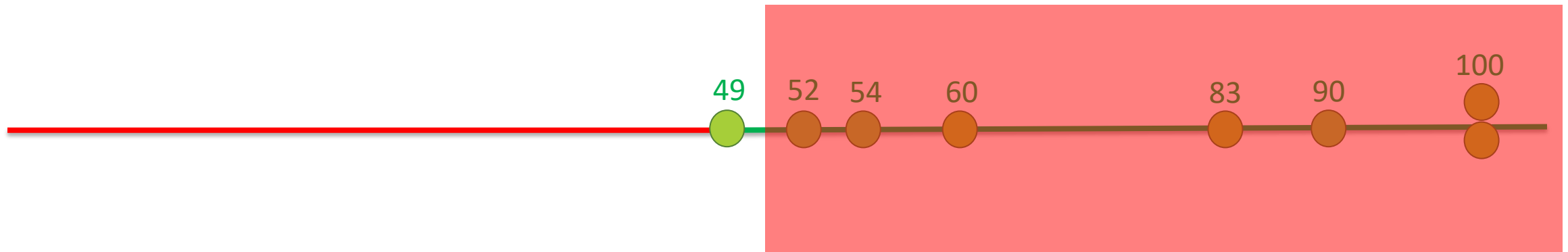
Linear Search SAT-UNSAT (LSU) Concept

1. $F \leftarrow H$ (a CNF F is initialized with the hard clauses)
2. Run an incremental SAT solver over F
3. If SAT with a model M
 - Block all the models of weight $\geq M(\psi)$ in F (using a cardinality or a Pseudo-Boolean constraint)
4. Else (UNSAT)
 - Return the last model M (M is guaranteed to be an optimal model)



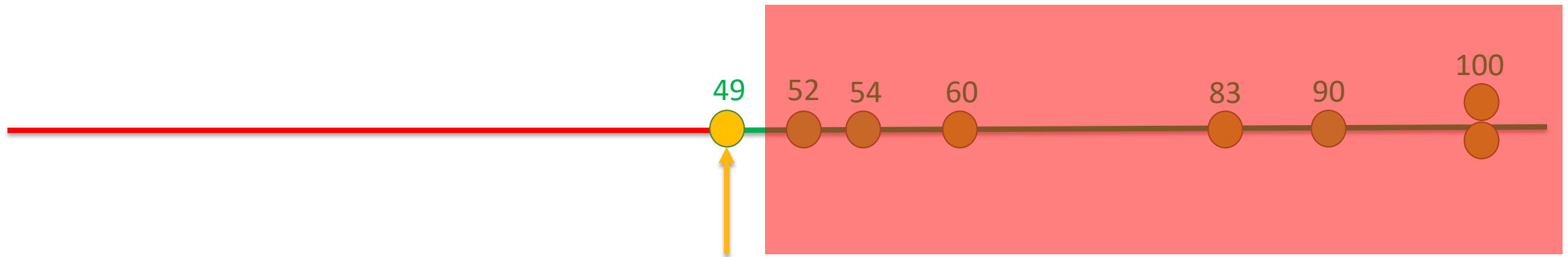
Linear Search SAT-UNSAT (LSU) Concept

1. $F \leftarrow H$ (a CNF F is initialized with the hard clauses)
2. Run an incremental SAT solver over F
3. If SAT with a model M
 - Block all the models of weight $\geq M(\psi)$ in F (using a cardinality or a Pseudo-Boolean constraint)
4. Else (UNSAT)
 - Return the last model M (M is guaranteed to be an optimal model)



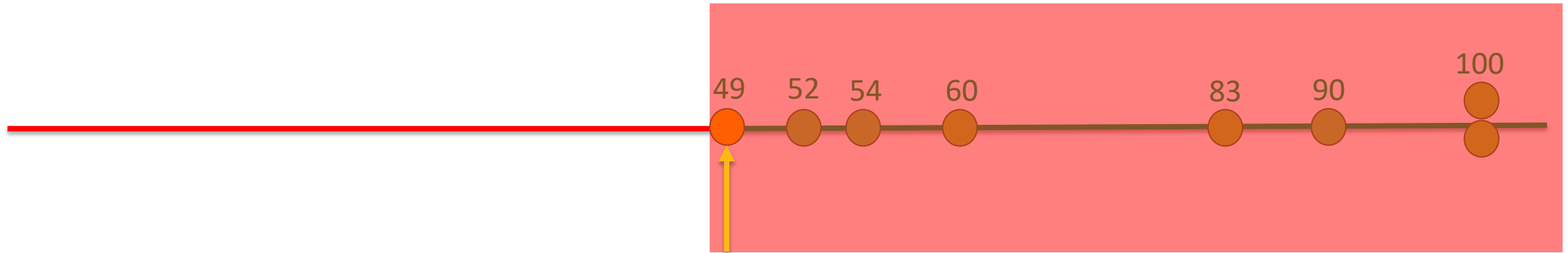
Linear Search SAT-UNSAT (LSU) Concept

1. $F \leftarrow H$ (a CNF F is initialized with the hard clauses)
2. Run an incremental SAT solver over F
3. If SAT with a model M
 - Block all the models of weight $\geq M(\psi)$ in F (using a cardinality or a Pseudo-Boolean constraint)
4. Else (UNSAT)
 - Return the last model M (M is guaranteed to be an optimal model)



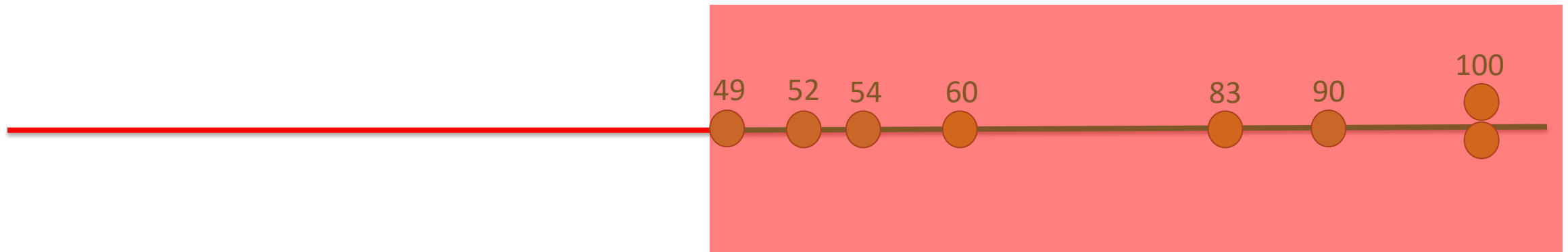
Linear Search SAT-UNSAT (LSU) Concept

1. $F \leftarrow H$ (a CNF F is initialized with the hard clauses)
2. Run an incremental SAT solver over F
3. If SAT with a model M
 - Block all the models of weight $\geq M(\psi)$ in F (using a cardinality or a Pseudo-Boolean constraint)
4. Else (UNSAT)
 - Return the last model M (M is guaranteed to be an optimal model)



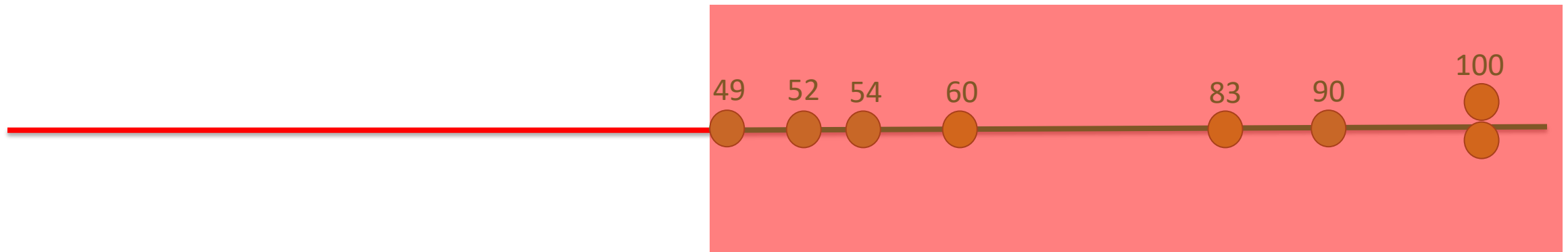
Linear Search SAT-UNSAT (LSU) Concept

1. $F \leftarrow H$ (a CNF F is initialized with the hard clauses)
2. Run an incremental SAT solver over F
3. If SAT with a model M
 - Block all the models of weight $\geq M(\psi)$ in F (using a cardinality or a Pseudo-Boolean constraint)
4. Else (UNSAT)
 - Return the last model M (M is guaranteed to be an optimal model)



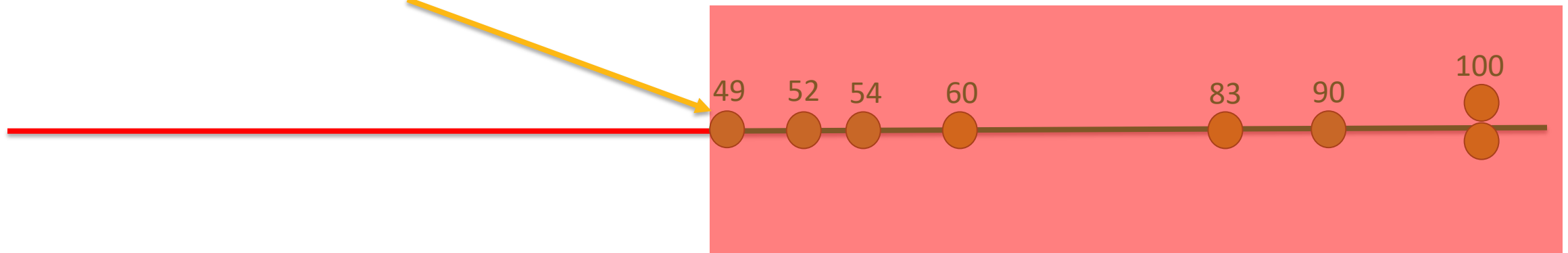
Linear Search SAT-UNSAT (LSU) Concept

1. $F \leftarrow H$ (a CNF F is initialized with the hard clauses)
2. Run an incremental SAT solver over F
3. If SAT with a model M
 - Block all the models of weight $\geq M(\psi)$ in F (using a cardinality or a Pseudo-Boolean constraint)
4. Else (UNSAT)
 - Return the last model M (M is guaranteed to be an optimal model)



Linear Search SAT-UNSAT (LSU) Concept

1. $F \leftarrow H$ (a CNF F is initialized with the hard clauses)
2. Run an incremental SAT solver over F
3. If SAT with a model M
 - Block all the models of weight $\geq M(\psi)$ in F (using a cardinality or a Pseudo-Boolean constraint)
4. Else (UNSAT)
 - Return the last model M (M is guaranteed to be an optimal model)



Cardinality and Pseudo-Boolean (PB) Constraints

In LSU: how to block the models with an upper bound on the weight?

Unweighted MaxSAT:

- Cardinality constraint: $t_1 + \dots + t_n \leq b$
 - Example: $t_1 + t_2 + t_3 + t_4 + t_5 \leq 3$

Weighted MaxSAT:

- PB constraint: $w_1 * t_1 + \dots + w_n * t_n \leq b$
 - Example: $2t_1 + 4t_2 + t_3 + 5t_4 + 7t_5 \leq 10$

Next: the totalizer encoding for unweighted MaxSAT

- Used by state-of-the-art anytime MaxSAT solvers

Encoding Cardinality Constraints: The Totalizer

Olivier Bailleux and Yacine Boufkhad: *Efficient CNF encoding of boolean cardinality constraints*. CP 2003: 108–122.

Totalizer: unary encoding-based addition tree to represent $(t_1 + \dots + t_n)$

- Unary encoding: 1 = 01; 2 = 011; 3 = 0111; 4 = 01111; 5 = 011111; ...

Complexity: $O(n^2)$ clauses and $O(n * \log(n))$ variables

Given an upper-bound b on the sum value: $O(n * b)$ clauses

Markus Büttner, Jussi Rintanen: *Satisfiability Planning with Constraints on the Number of Actions*. ICAPS 2005: 292-299

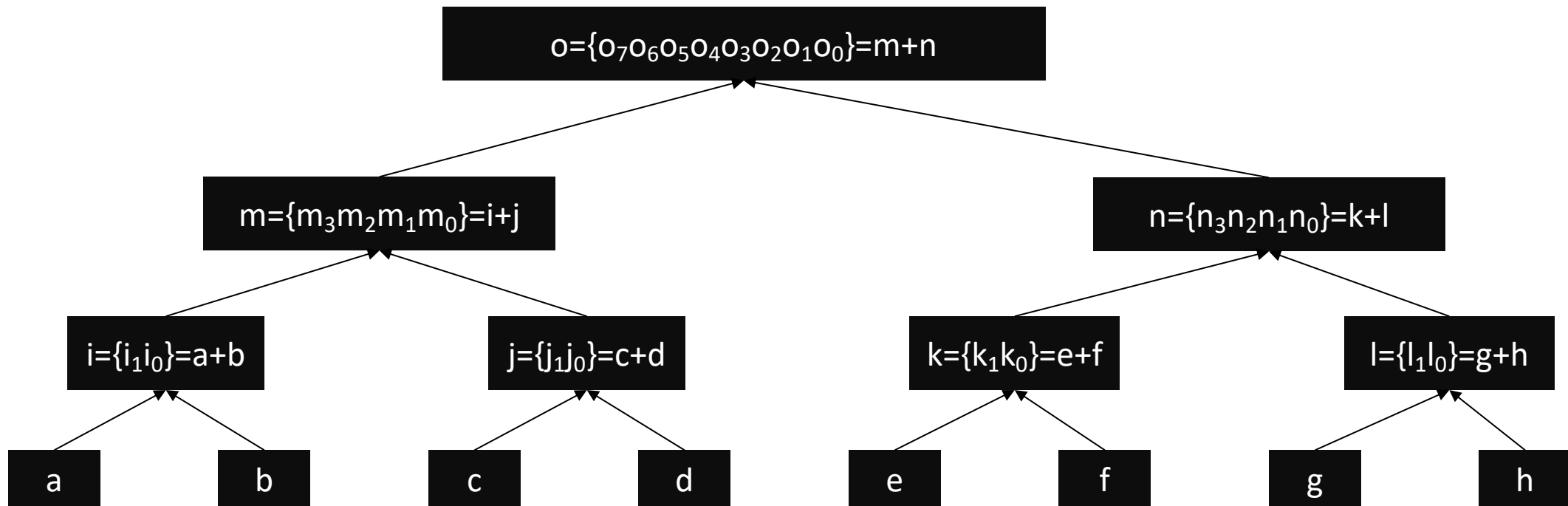
- Tight upper bound significantly reduces complexity

Useful feature (shared with many other encodings): arc consistency

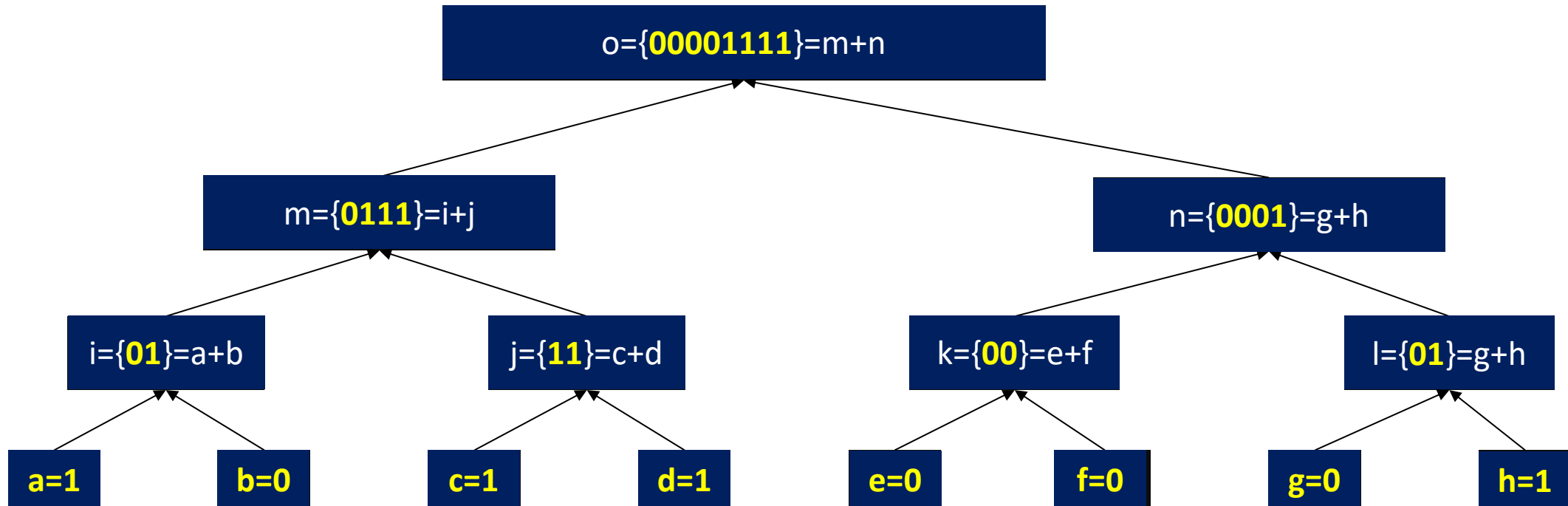
- Consider the following cardinality constraint $t_1 + \dots + t_n \leq b$.
- If b variables are assigned 1, then unit propagation enforces 0 on the remaining $n - b$ variables.
- If $b+1$ variables are assigned 1, then unit propagation triggers a conflict

Tightening the upper bound is easy and efficient → handy for LSU & anytime MaxSAT

The Totalizer: $o = a + b + c + d + e + f + g + h$



The Totalizer Example



The Totalizer Clauses

$$N_3 = N_1 + N_2$$

for (p = 0; p <= |N₁|; ++p)

for (q = 0; q <= |N₂|; ++q)

Add clause $\neg N_1[p] \vee \neg N_2[q] \vee N_3[p+q]$ ($N_1[p]=1$ and $N_2[q]=1 \rightarrow N_3[p+q]=1$)

Add clause $N_1[p+1] \vee N_2[q+1] \vee \neg N_3[p+q+1]$ ($N_1[p+1]=0$ and $N_2[q+1]=0 \rightarrow N_3[p+q+1]=0$)

p=2

q=3

p+q=5

(sum of the inputs is ≥ 5)

p+1=2

q+1=3

p+q+1=5

(sum of the inputs < 5)

Complexity: for n inputs, $O(n^2)$ clauses and $O(n * \log(n))$ variables

$N_3 = N_1 + N_2$: bits 1-indexed

N_1 : bits 1-indexed

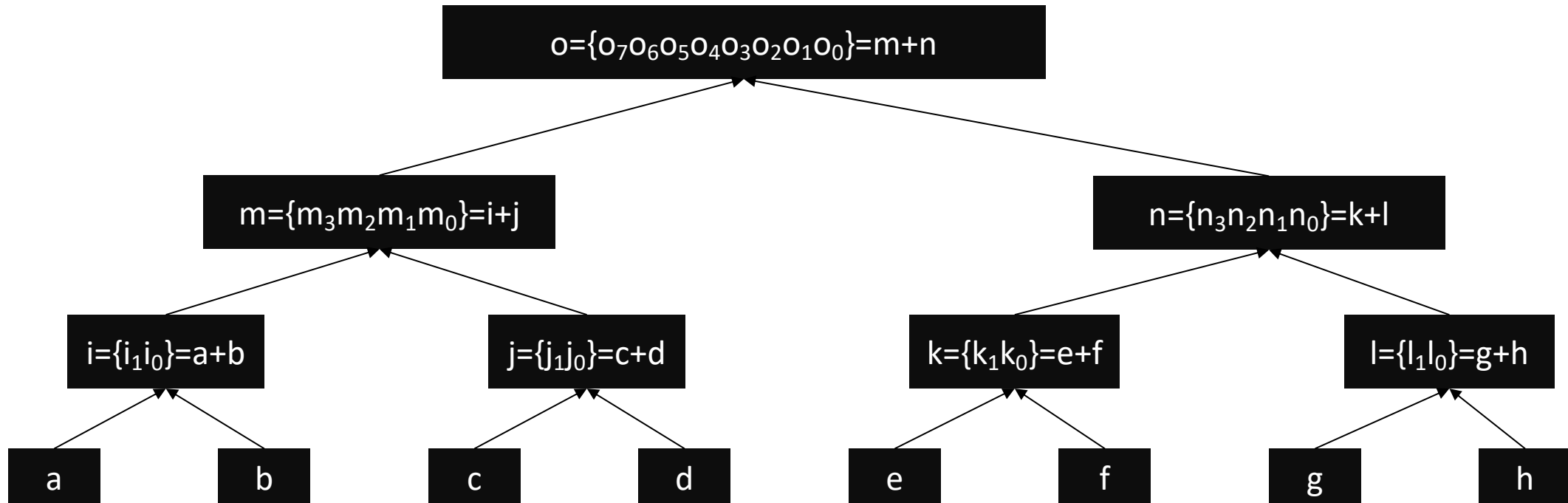
N_2 : bits 1-indexed

$N_1[0] = N_2[0] = N_3[0] = 1;$

$N_1[|N_1+1|] = N_2[|N_2+1|] = N_3[|N_3+1|] = 1$

Totalizer with Upper Bound

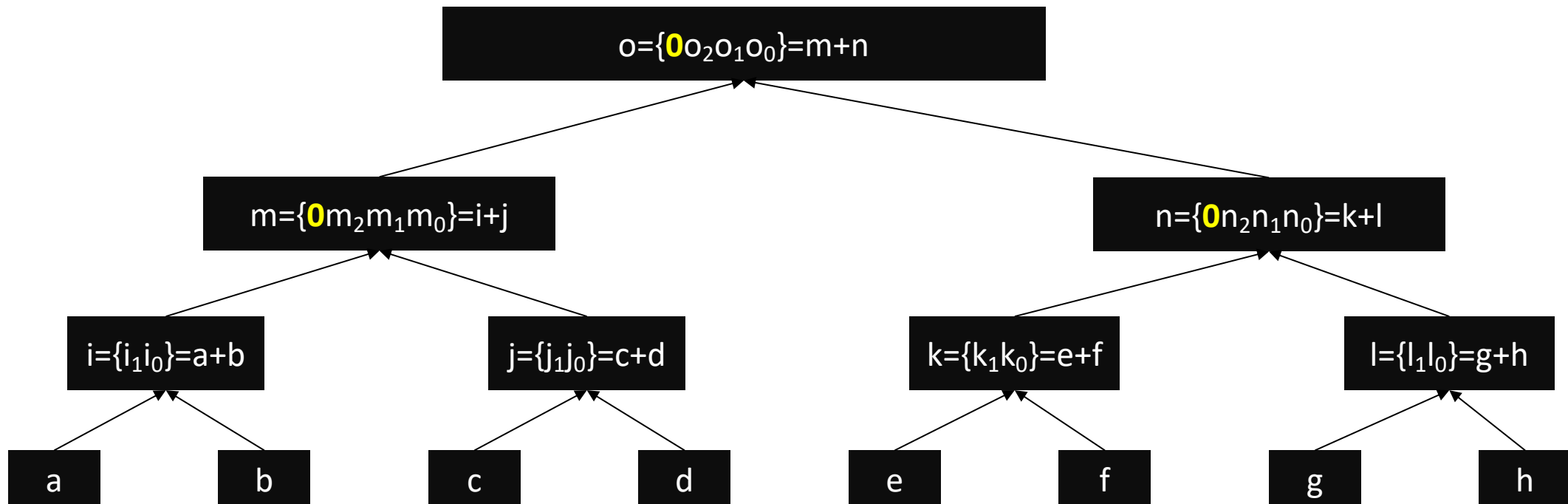
$$o = a + b + c + d + e + f + g + h \leq 3$$



Totalizer with Upper Bound:

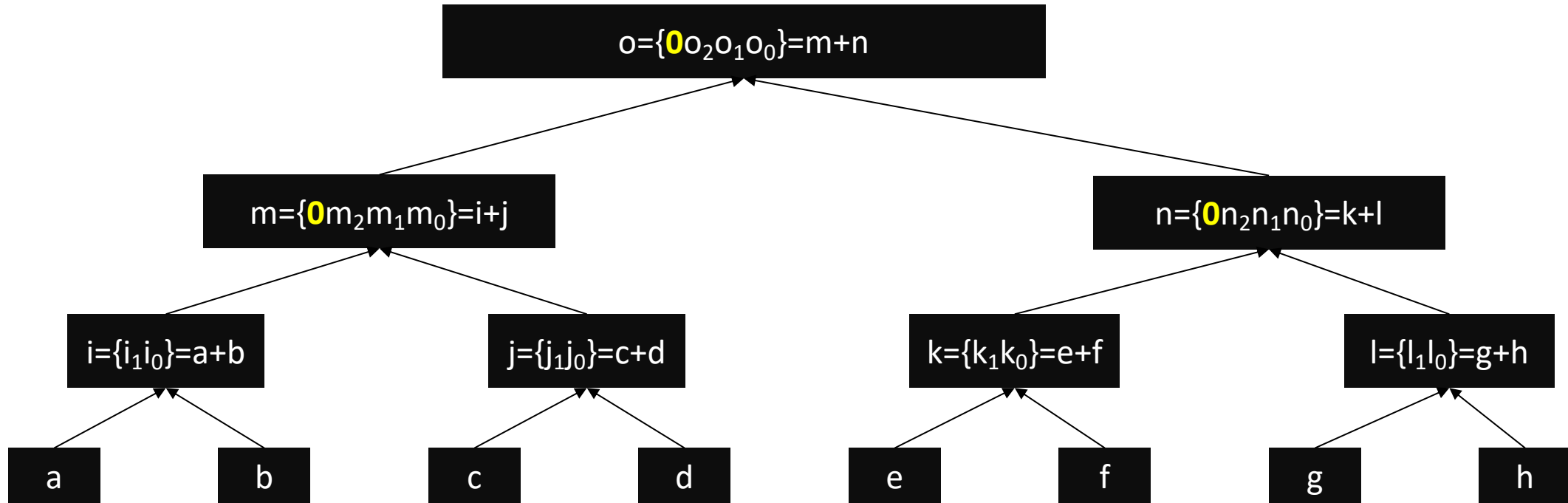
$$o = a + b + c + d + e + f + g + h \leq b=3$$

Complexity: for n inputs, $O(n * b)$ clauses and $O(n * \log(n))$ variables



Totalizer: Tightening the Upper Bound

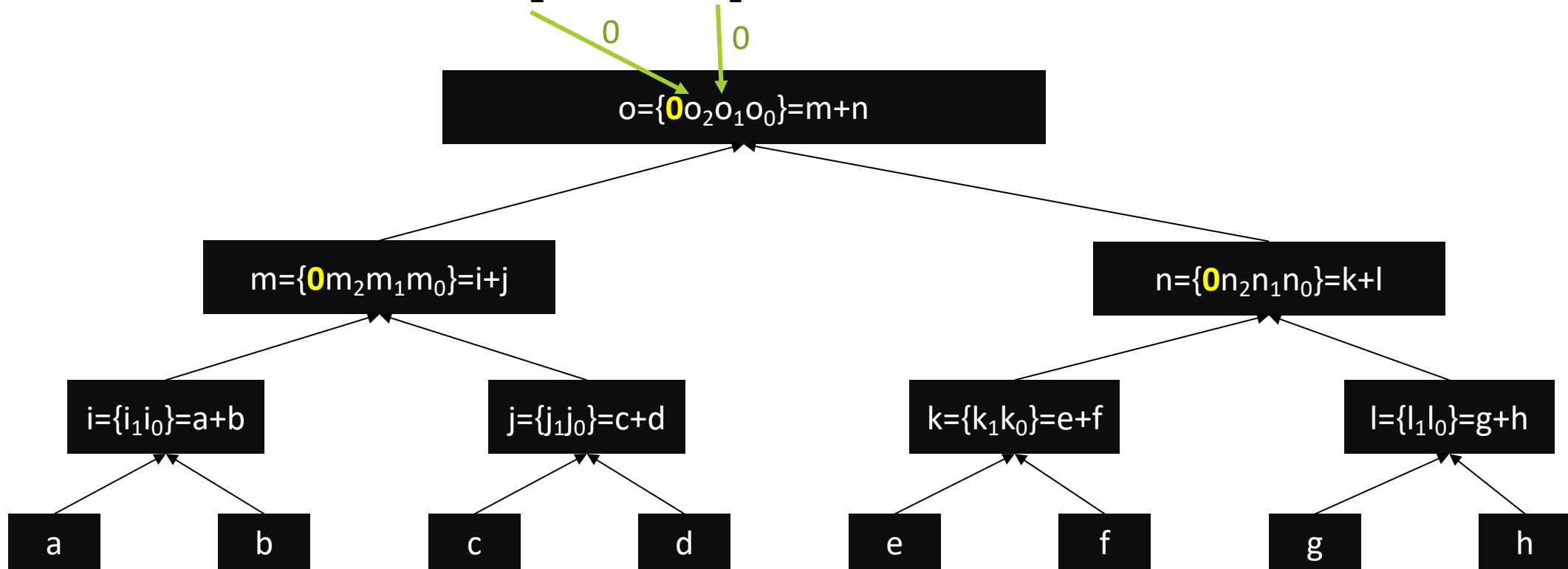
$$o = a + b + c + d + e + f + g + h \leq b=3$$



Totalizer: Tightening the Upper Bound

$$o = a + b + c + d + e + f + g + h \leq b=3 \rightarrow b=1$$

Assert the unit clauses $\{\neg o_2\}$ and $\{\neg o_1\}$; no need to create a new totalizer!



LSU's Main Problem

The convergence is often too slow

- The first model might be too far away
- Finding the next best model might be too slow

SAT-based Local Search to the Rescue

TORC (Target-Optimal-Rest-Conservative) polarity selection

Alexander Nadel: **Anytime Weighted MaxSAT with Improved Polarity Selection and Bit-Vector Optimization**. [FMCAD 2019](#): 193-202

- A **non-target** variable: TORC sets its polarity to its value in the best model so far
 - Only after the initial SAT invocation is completed
 - In practice, finds the next best model much faster!
- A **target** variable: TORC sets its polarity to 0
 - The first model and any subsequent model is closer to the ideal
- Applied by the state-of-the-art anytime MaxSAT solvers

Polosat: a dedicated SAT-based local search algorithm, invoked *instead* of SAT (next)

Alexander Nadel: **On Optimizing a Generic Function in SAT**. [FMCAD 2020](#): 205-213

Polosat for MaxSAT

Hard Clauses H

- Satisfiable

Optimization Target $T = \{t_{n-1}, t_{n-2}, \dots, t_0\}$

- Each target bit t_i is a literal (unit clause), associated with an integer weight $w(t_i) > 0$

Output: A model M to H which minimizes the weight of the satisfied target bits $\psi = w_{n-1} * t_{n-1} + \dots + w_1 * t_1 + \dots + w_0 * t_0$

Polosat: further simulate local search with CDCL SAT

- $M := \text{SAT}(H)$
- Run the following loop until M is not improved anymore
 - Go over all the “bad” target bits (not assigned 0 in any model so far)
 - Flip the current bad target bit t:
 - » $M' := \text{SAT}(H, \{\neg t\})$, but stop after 1000 conflicts! ($\neg t$ is an assumption)
 - If (satisfiable and M' improves M) $M := M'$
- Always apply TORC: fix targets to 0 and non-targets to the best model so far

Polosat Cont.

Default in state-of-the-art anytime MaxSAT solvers

- Used by the top 3 solvers in all the 4 anytime categories @ MaxSAT Evaluation 2023

Can be applied to optimize any PB function

Enabler for solving industrial optimization problems at Intel

Recently shown to boost Pseudo-Boolean (PB) Optimization

- Markus Iser , Jeremias Berg, Matti Järvisalo:
Oracle-Based Local Search for Pseudo-Boolean Optimization. [ECAI 2023: 1124-1131](#)

Agenda

How does a conflict-driven SAT solver work?

- The core: backtrack search, Boolean Constraint Propagation (BCP), conflict analysis
- Follow the first SAT Competition winners (from Chaff till Minisat)

Applying SAT by example


- Paradigms: incremental SAT solving, SAT-based local search, example encodings
- Applications: Bounded Model Checking (BMC), proof-based abstraction refinement, bug hunting, anytime MaxSAT


Advanced core SAT algorithms


- Follow the SAT Competition winners after Minisat & discussion about incremental SAT

SAT Competition & Race Winners (CNF & Appl. & Seq. & Non-incr. & All-inst.)

2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
zChaff	Forklift	zChaff	SatELite GTI	MiniSat	RSAT	MiniSat	Precosat	Crypto MiniSat	Glucose
Moskewicz Madigan Zhao Zhang Malik	Goldberg Novikov	Moskewicz Madigan Zhao Zhang Malik	Eén Sörensson	Eén Sörensson	Pipatsrisa wat Darwiche	Eén Sörensson	Biere	Soos	Audemard Simon
2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Glucose	Lingeling	Lingeling	abcdSAT	Maple COMSPS	Maple LCMDist	Maple LCMDist ChronoBT	Maple LCMDist ChronoBT DLv3	Kissat	KissatMAB
Audemard Simon	Biere	Biere	Chen	Liang Oh Ganesh Czarnecki Poupart	Xiao Luo Li Manya Lu	Nadel Ryvchin	Kochemazov Zaikin Kondratiev Semenov	Biere Fazekas Fleury Heisinger	Cherif Habet Terrioux
2022	2023								
KissatMAB-HyWalk	SBVA-CaDiCaL								
Zheng He Chen Zhou Li	Haberlandt Green								

MiniSat-based:


Armin Biere's & derived:


Others:


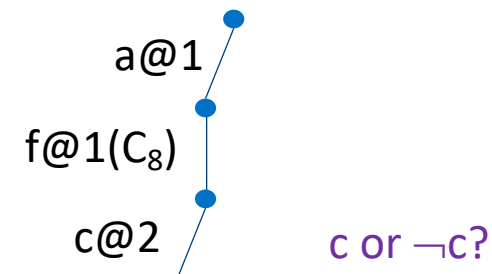
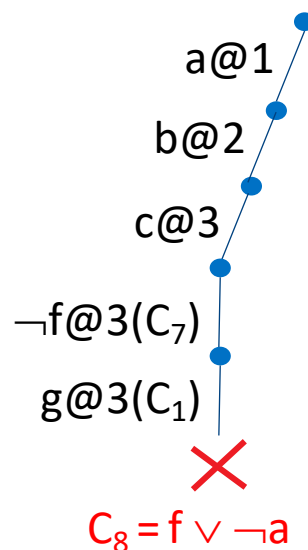
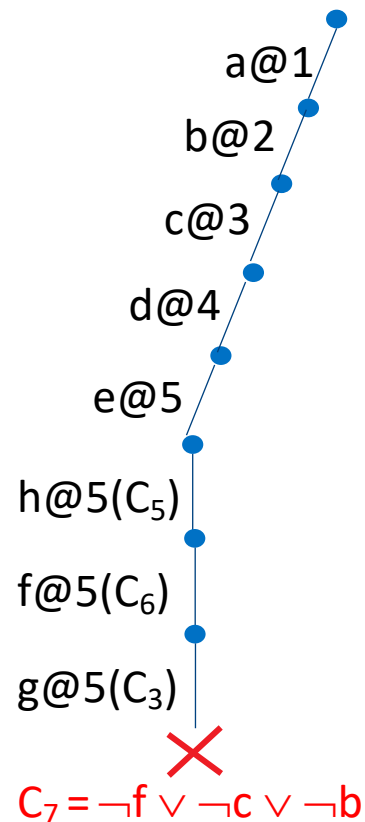
RSAT's Phase Saving

Polarity selection: always choose the latest polarity – aka *phase saving*

Knot Pipatsrisawat, Adnan Darwiche:

A Lightweight Component Caching Scheme for Satisfiability Solvers. SAT 2007: 294-299

- $C_1 = \neg a \vee f \vee g$
- $C_2 = \neg a \vee f \vee \neg g$
- $C_3 = \neg c \vee \neg f \vee g$
- $C_4 = \neg b \vee \neg f \vee \neg g$
- $C_5 = \neg e \vee h$
- $C_6 = \neg e \vee \neg h \vee f$



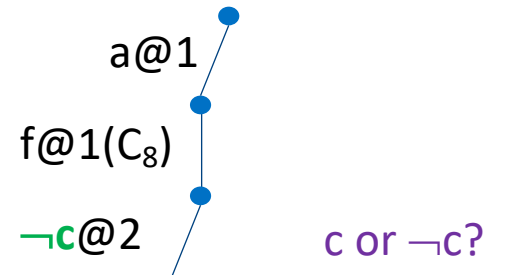
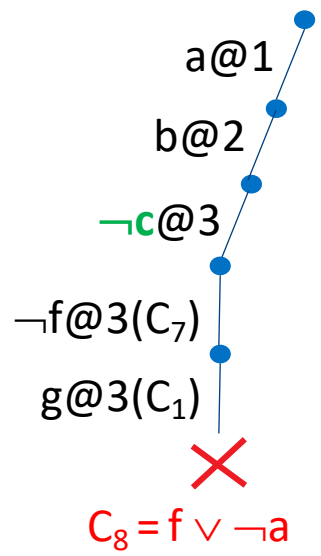
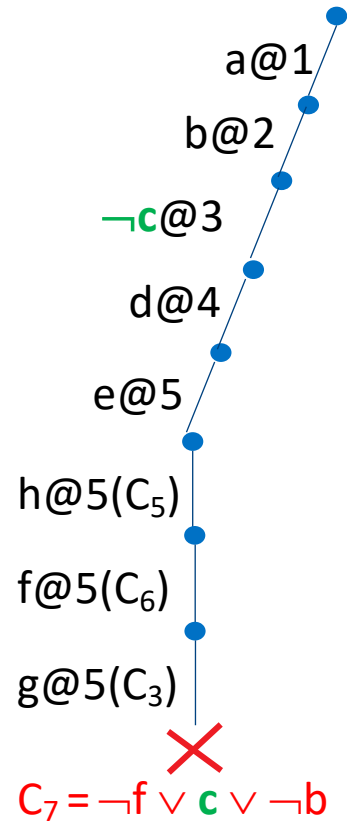
RSAT's Phase Saving

Polarity selection: always choose the latest polarity – aka *phase saving*

Knot Pipatsrisawat, Adnan Darwiche:

A Lightweight Component Caching Scheme for Satisfiability Solvers. SAT 2007: 294-299

- $C_1 = \neg a \vee f \vee g$
- $C_2 = \neg a \vee f \vee \neg g$
- $C_3 = c \vee \neg f \vee g$
- $C_4 = \neg b \vee \neg f \vee \neg g$
- $C_5 = \neg e \vee h$
- $C_6 = \neg e \vee \neg h \vee f$




Phase Saving


Locality principle: refocus on the currently explored subspace


State-of-the-art polarity selection heuristic till 2020!

SAT Competition & Race Winners (CNF & Appl. & Seq. & Non-incr. & All-inst.)

2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
zChaff	Forklift	zChaff	SatELite GTI	MiniSat	RSAT	MiniSat	Precosat	Crypto MiniSat	Glucose
Moskewicz Madigan Zhao Zhang Malik	Goldberg Novikov	Moskewicz Madigan Zhao Zhang Malik	Eén Sörensson	Eén Sörensson	Pipatsrisa wat Darwiche	Eén Sörensson	Biere	Soos	Audemard Simon
2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Glucose	Lingeling	Lingeling	abcdSAT	Maple COMSPS	Maple LCMDist	Maple LCMDist ChronoBT	Maple LCMDist ChronoBT DLv3	Kissat	KissatMAB
Audemard Simon	Biere	Biere	Chen	Liang Oh Ganesh Czarnecki Poupart	Xiao Luo Li Manya Lu	Nadel Ryvchin	Kochemazov Zaikin Kondratiev Semenov	Biere Fazekas Fleury Heisinger	Cherif Habet Terrioux
2022	2023								
KissatMAB-HyWalk	SBVA-CaDiCaL								
Zheng He Chen Zhou Li	Haberlandt Green								

MiniSat-based:


Armin Biere's & derived:


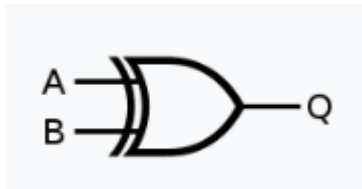
Others:


CryptoMiniSat

Mate Soos, [Karsten Nohl](#), [Claude Castelluccia](#): **Extending SAT Solvers to Cryptographic Problems.** [SAT 2009](#): 244-257

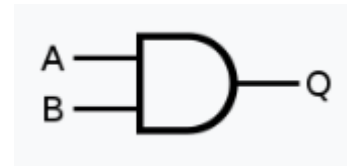
Motivation: XOR's are notoriously difficult

- Appear frequently in cryptographic problems
- Pruning doesn't work for XOR gates



INPUT		OUTPUT
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

$$\begin{aligned} &a \vee b \vee \neg q \\ &a \vee \neg b \vee q \\ &\neg a \vee b \vee q \\ &\neg a \vee \neg b \vee \neg q \end{aligned}$$



INPUT		OUTPUT
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

$$\begin{aligned} &a \vee \neg q \\ &b \vee \neg q \\ &\neg a \vee \neg b \vee q \end{aligned}$$

CryptoMiniSat is still under active development!

- [CryptoMiniSat 5 | Wonderings of a SAT geek \(msoos.org\)](#)

CryptoMiniSat (as of 2010): Techniques

XOR clauses

- Identification, separate propagation & watches, binary XORs, Gaussian elimination

Polarity selection:

- Phase saving + occasional random flipping

Heuristics tuned separately for cryptographic vs. industrial instances

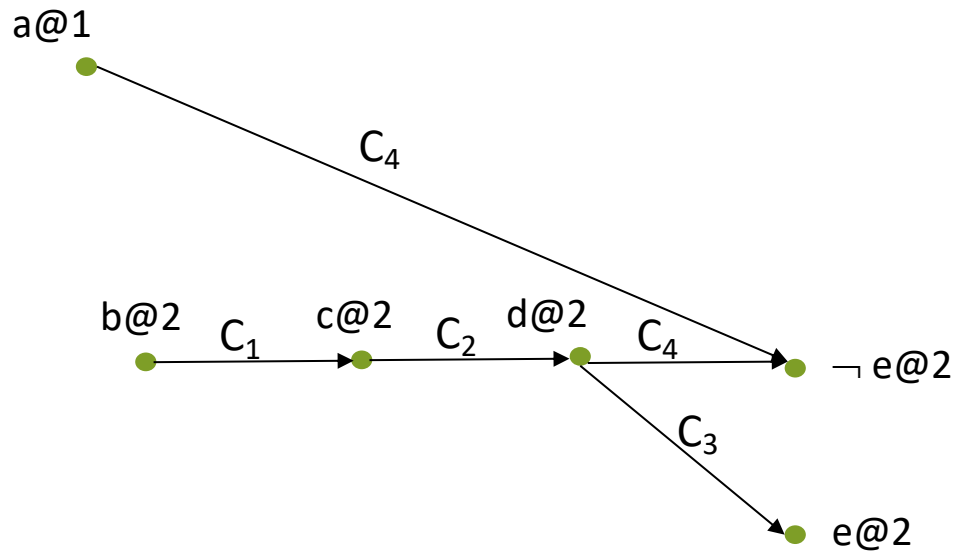
- based on the percentage of XOR clauses and stability of variable activity

3 techniques presented next (after some preliminary material)

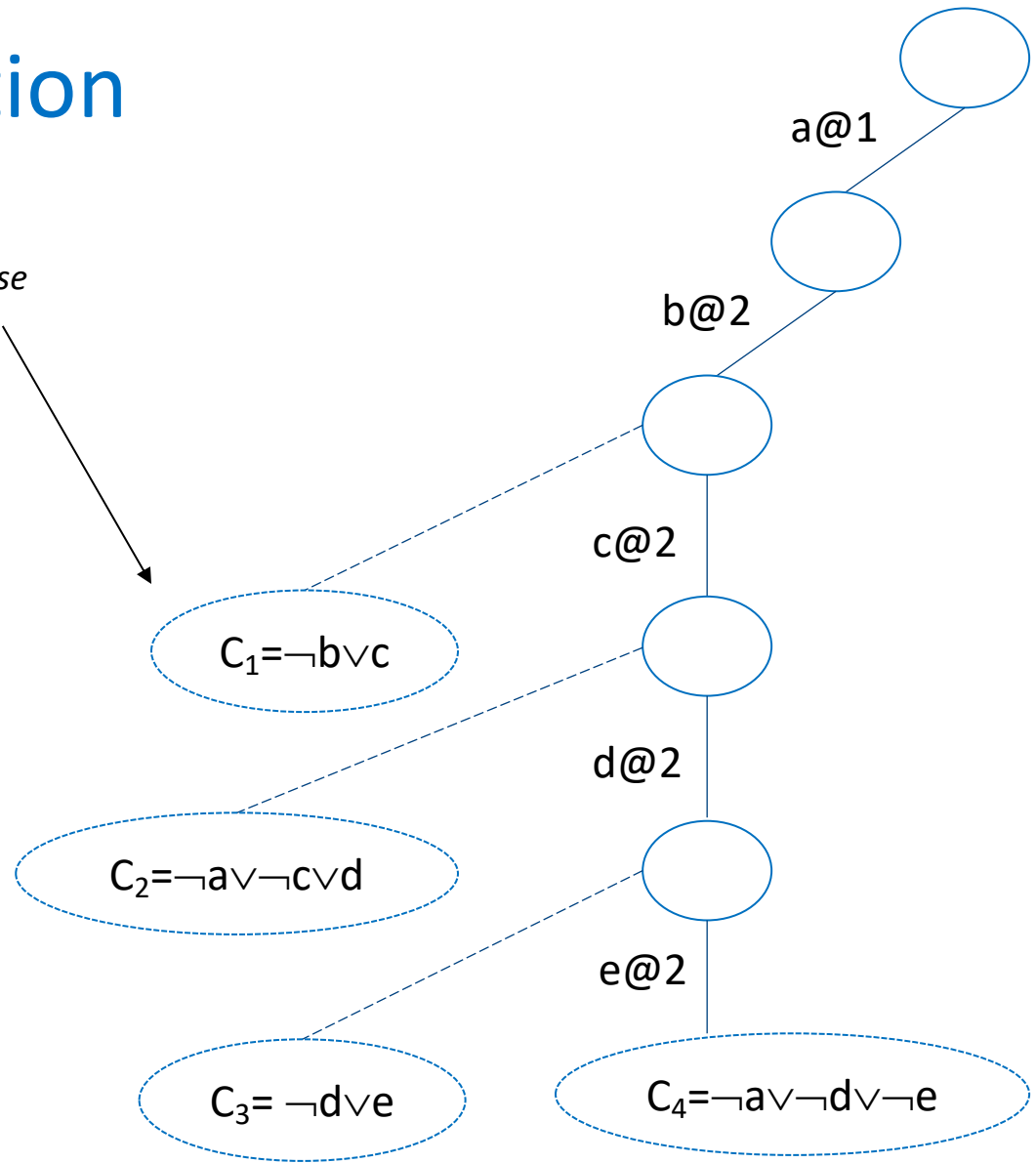
1. On-the-fly subsumption
2. Failed literal probing
3. Hyper-binary resolution

Conflict Analysis as Resolution

- $C_1 = \neg b \vee c$
- $C_2 = \neg a \vee \neg c \vee d$
- $C_3 = \neg d \vee e$
- $C_4 = \neg a \vee \neg d \vee \neg e$



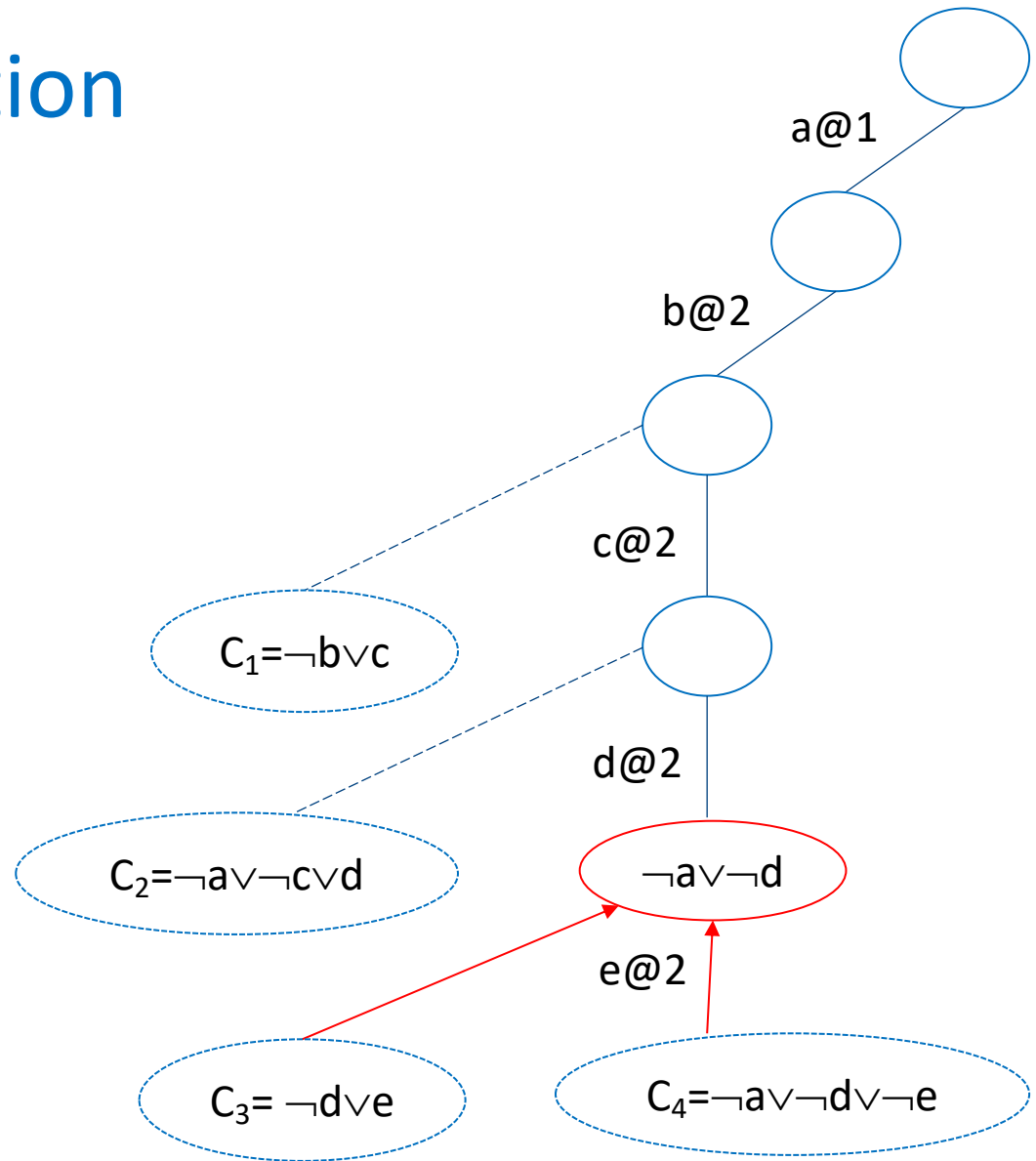
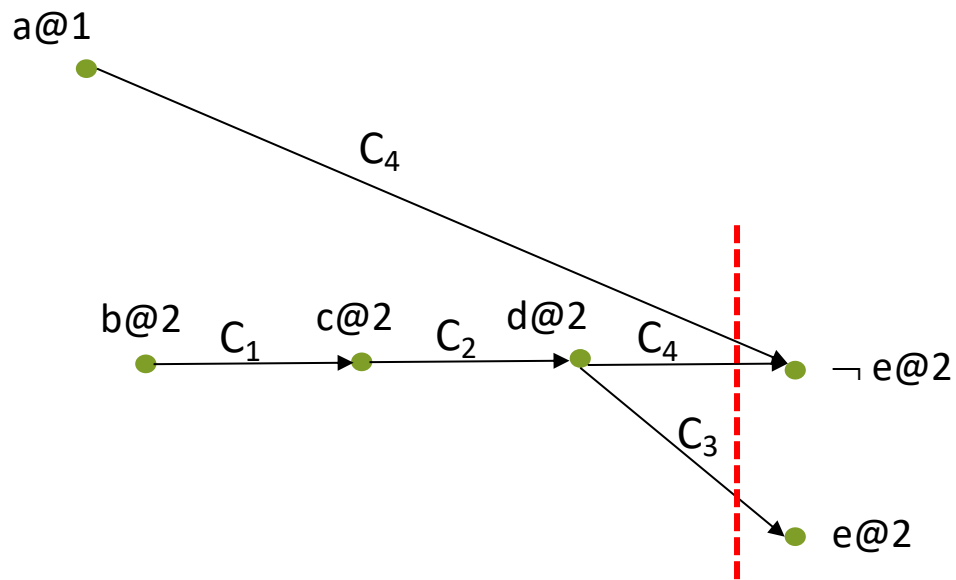
Parent Clause



- The conflict clause is, in fact, derived by resolution (from the conflict upwards)
- Rightmost cut in the graph = resolution over the conflicting variable
- The cut goes one step leftwards = resolution goes one step upwards

Conflict Analysis as Resolution

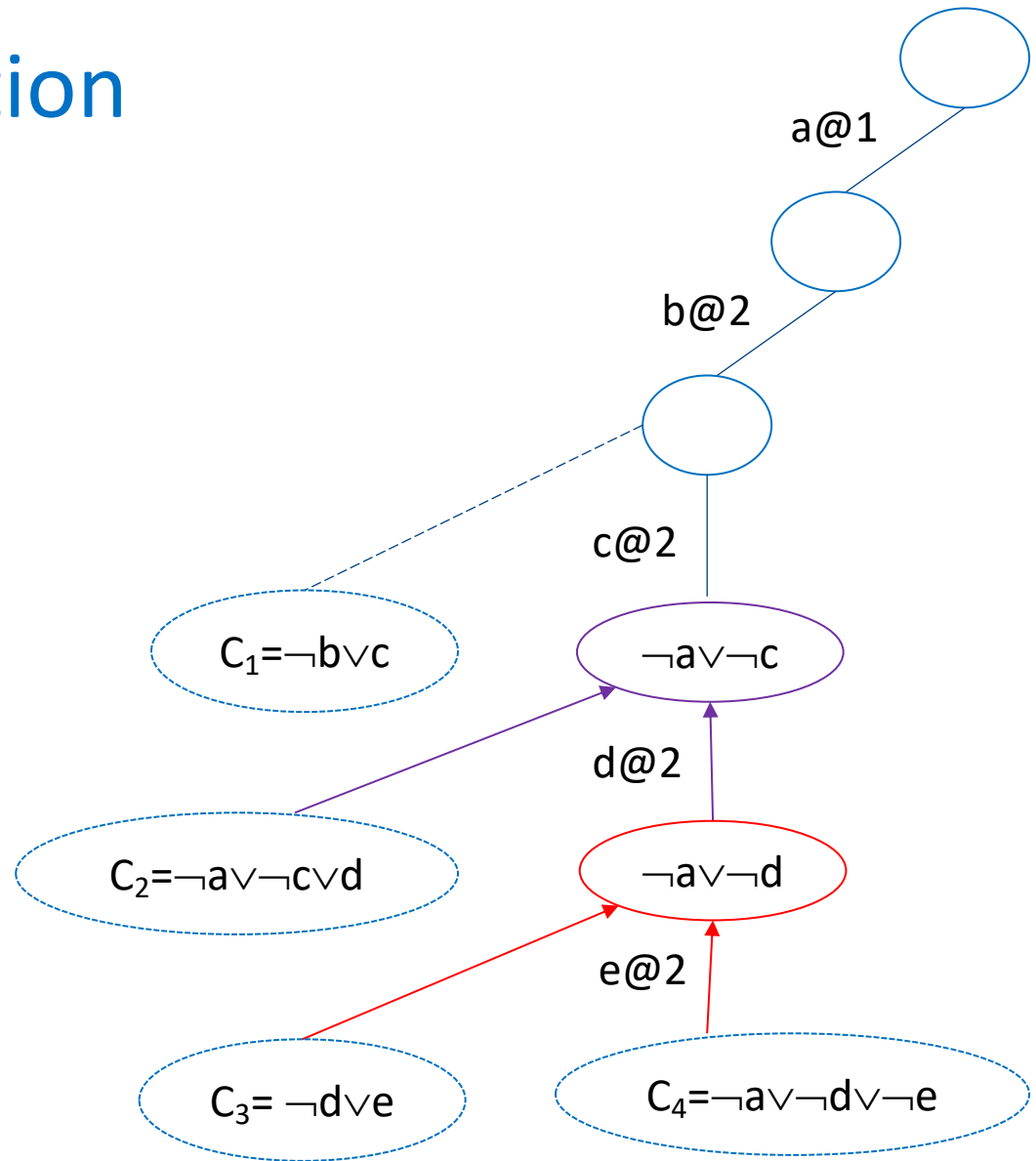
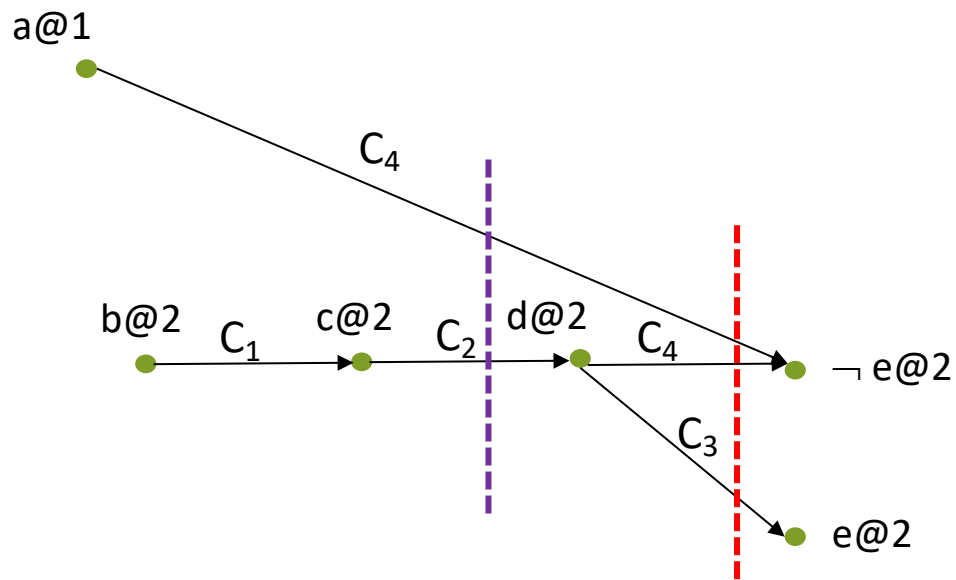
- $C_1 = \neg b \vee c$
- $C_2 = \neg a \vee \neg c \vee d$
- $C_3 = \neg d \vee e$
- $C_4 = \neg a \vee \neg d \vee \neg e$



- The conflict clause is, in fact, derived by resolution (from the conflict upwards)
- Rightmost cut in the graph = resolution over the conflicting variable
- The cut goes one step leftwards = resolution goes one step upwards

Conflict Analysis as Resolution

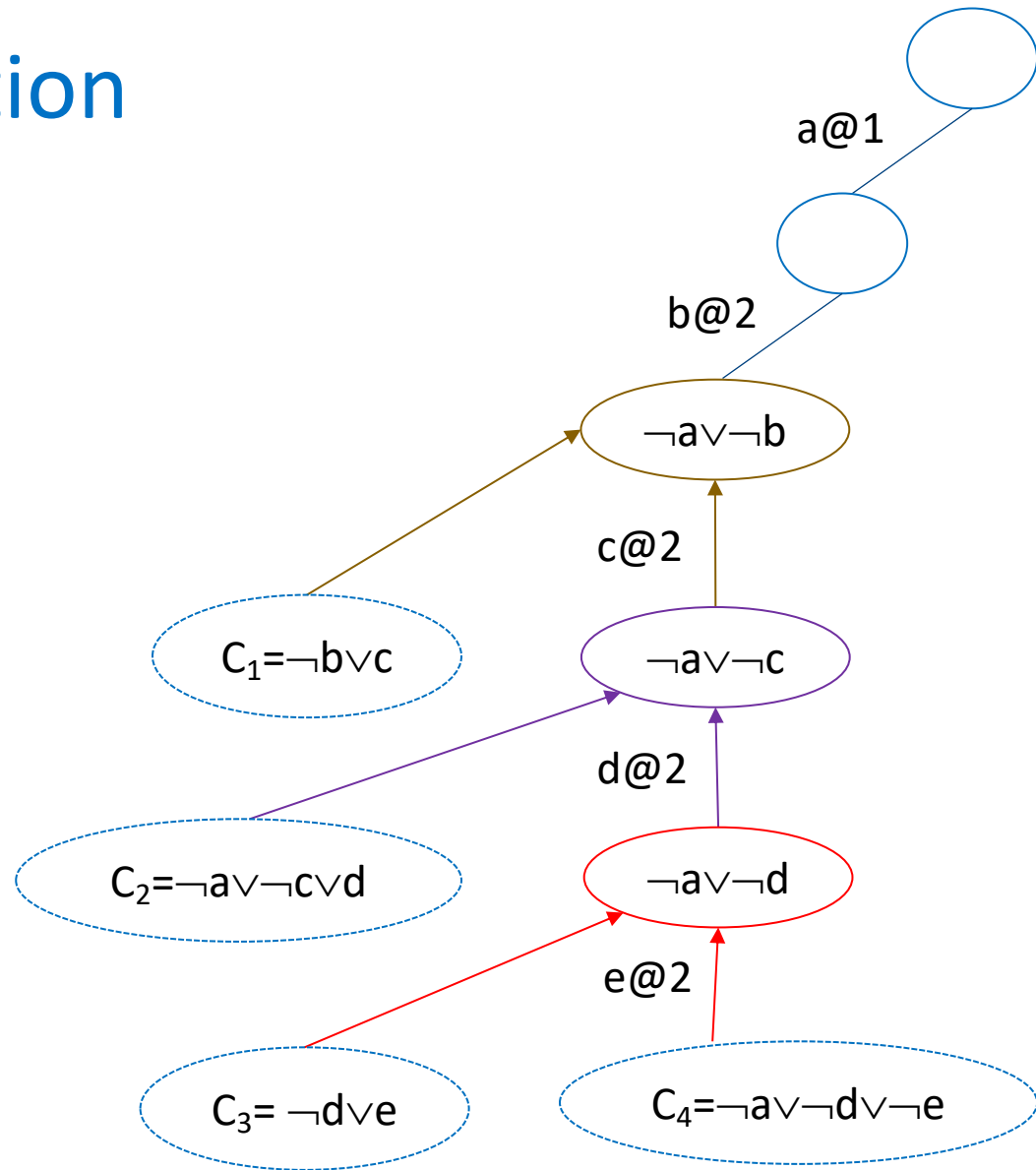
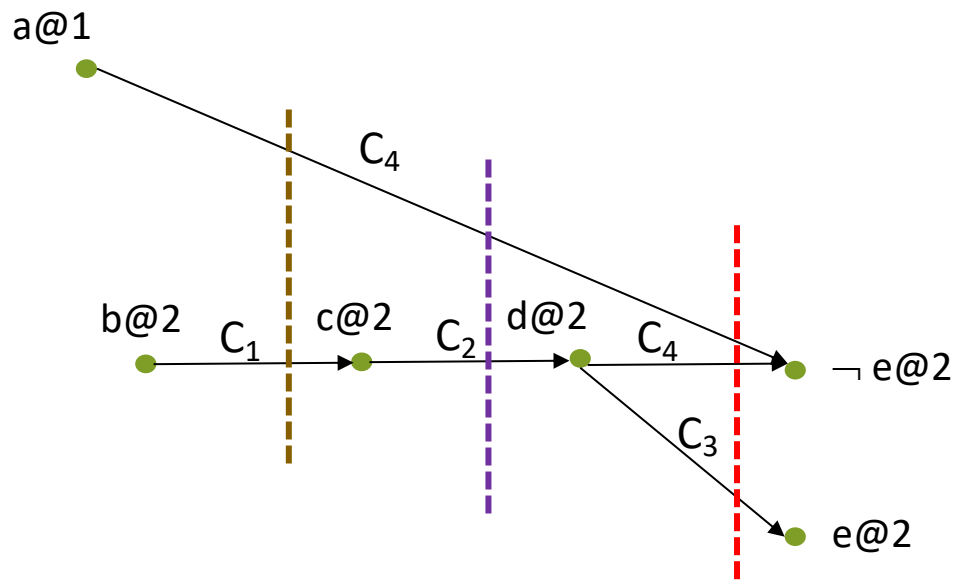
- $C_1 = \neg b \vee c$
- $C_2 = \neg a \vee \neg c \vee d$
- $C_3 = \neg d \vee e$
- $C_4 = \neg a \vee \neg d \vee \neg e$



- The conflict clause is, in fact, derived by resolution (from the conflict upwards)
- Rightmost cut in the graph = resolution over the conflicting variable
- The cut goes one step leftwards = resolution goes one step upwards

Conflict Analysis as Resolution

- $C_1 = \neg b \vee c$
- $C_2 = \neg a \vee \neg c \vee d$
- $C_3 = \neg d \vee e$
- $C_4 = \neg a \vee \neg d \vee \neg e$



- The conflict clause is, in fact, derived by resolution (from the conflict upwards)
- Rightmost cut in the graph = resolution over the conflicting variable
- The cut goes one step leftwards = resolution goes one step upwards

Conflict Analysis as Resolution

$$C_0 = \neg b \vee f$$

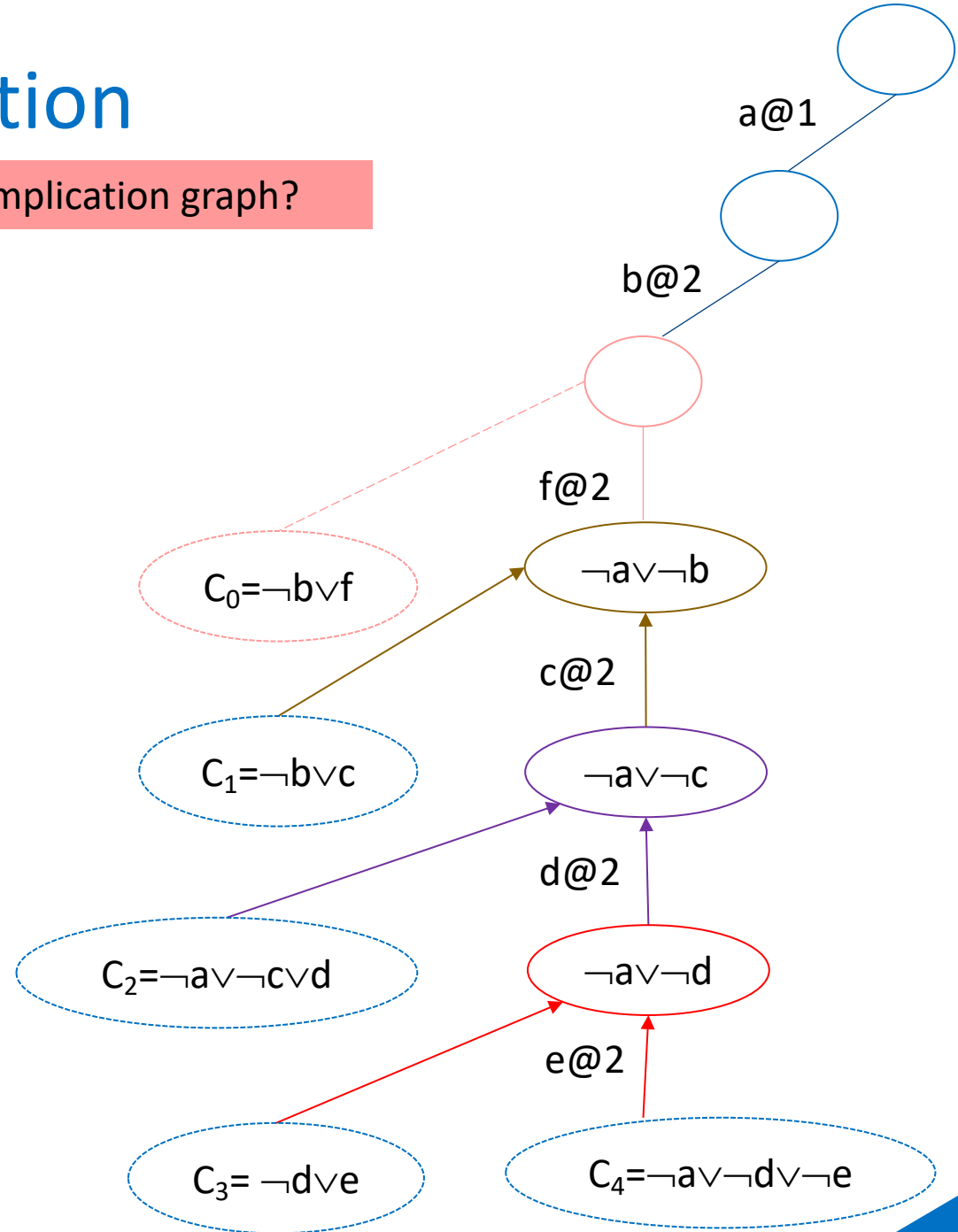
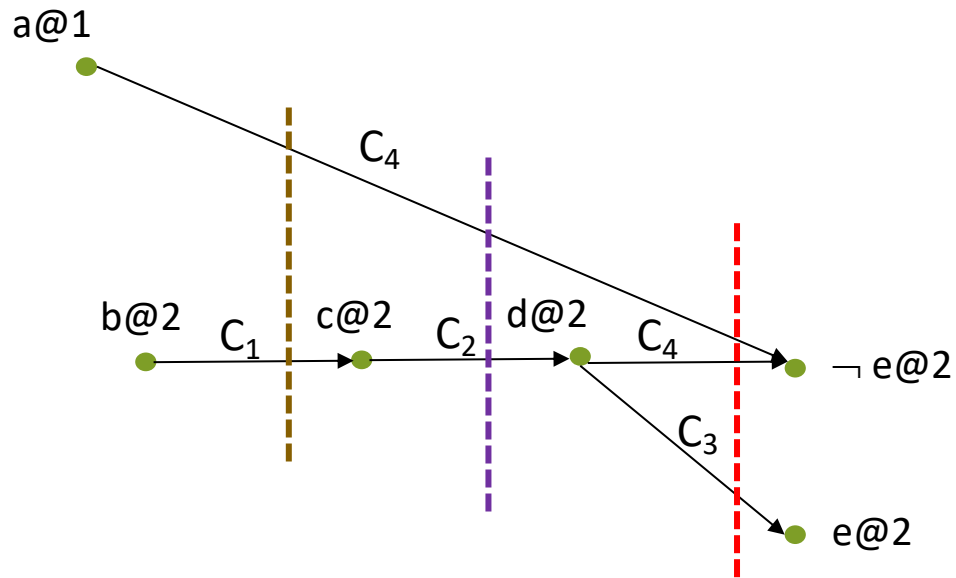
- Variable not in the implication graph?

$$C_1 = \neg b \vee c$$

$$C_2 = \neg a \vee \neg c \vee d$$

$$C_3 = \neg d \vee e$$

$$C_4 = \neg a \vee \neg d \vee \neg e$$



Conflict Analysis as Resolution

$$C_0 = \neg b \vee f$$

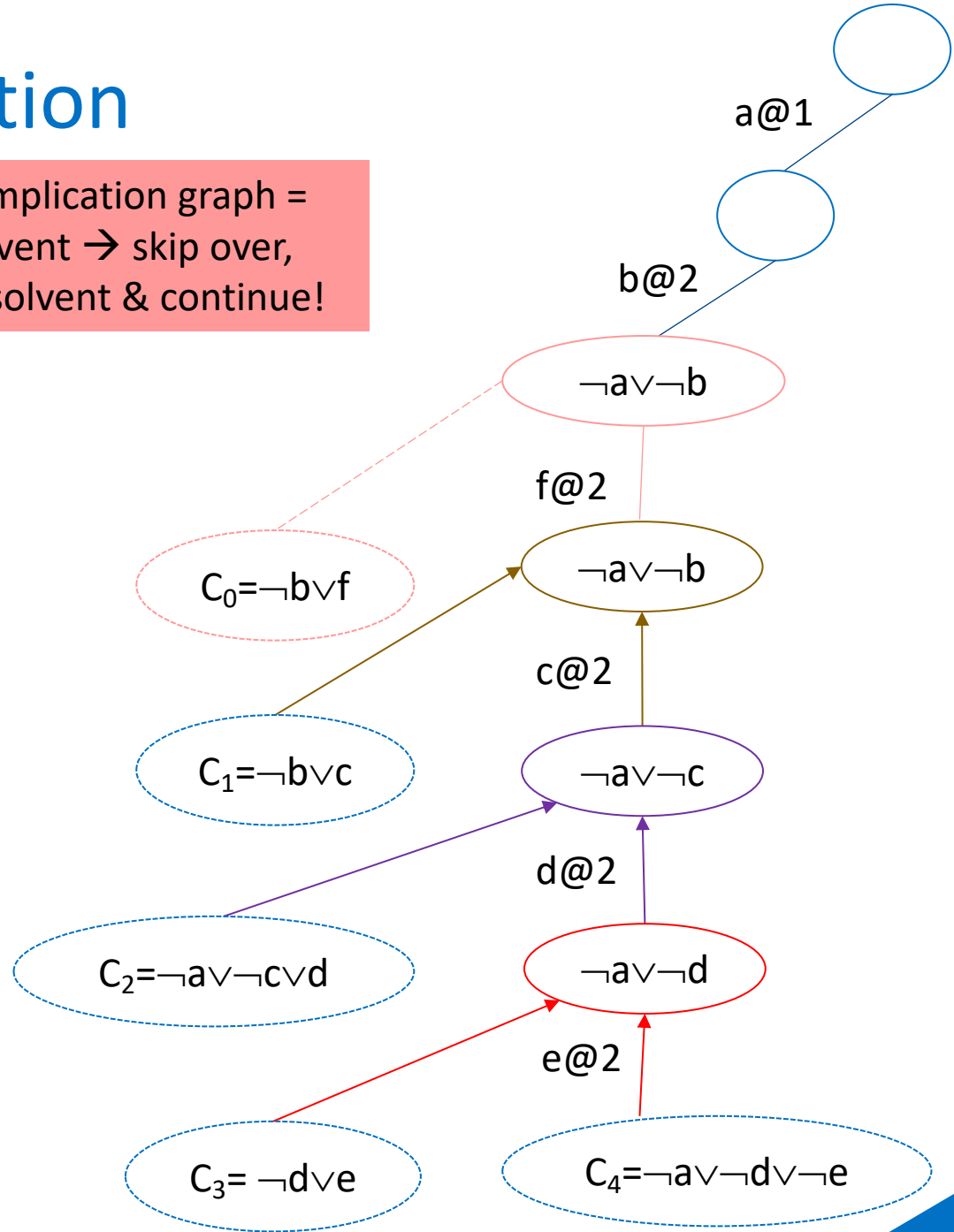
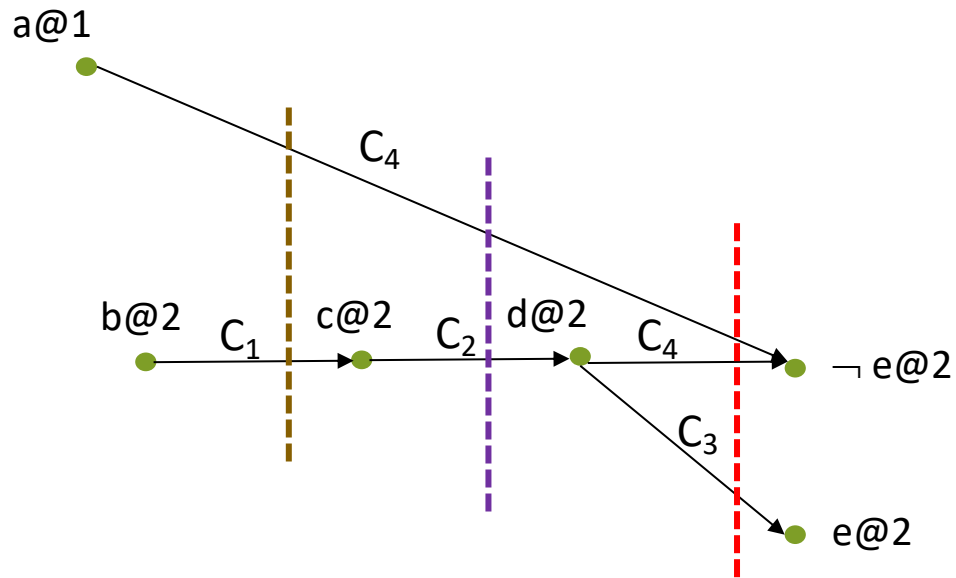
$$C_1 = \neg b \vee c$$

$$C_2 = \neg a \vee \neg c \vee d$$

$$C_3 = \neg d \vee e$$

$$C_4 = \neg a \vee \neg d \vee \neg e$$

- Variable not in the implication graph = not part of the resolvent \rightarrow skip over, don't change the resolvent & continue!



On-the-fly Subsumption

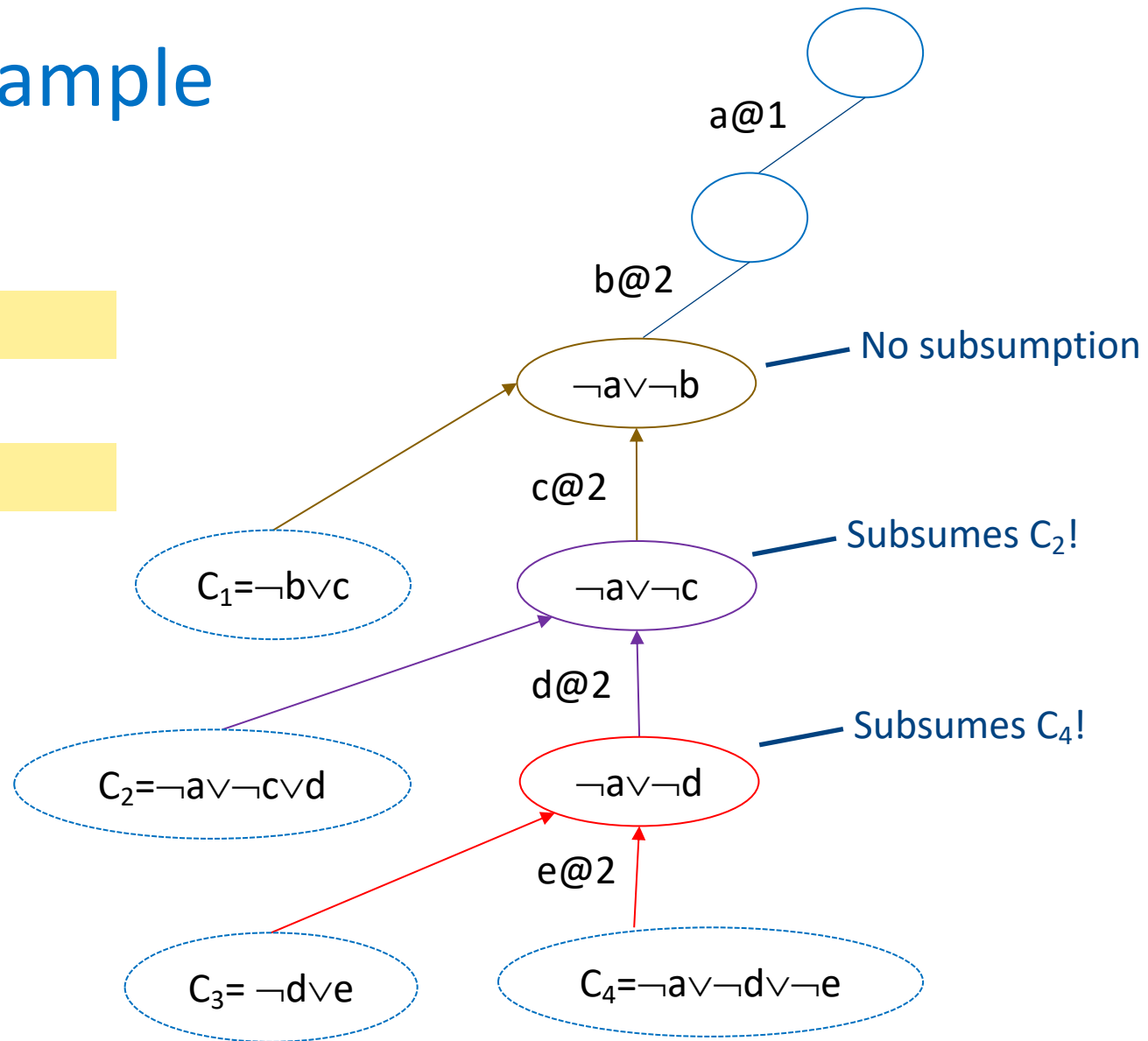
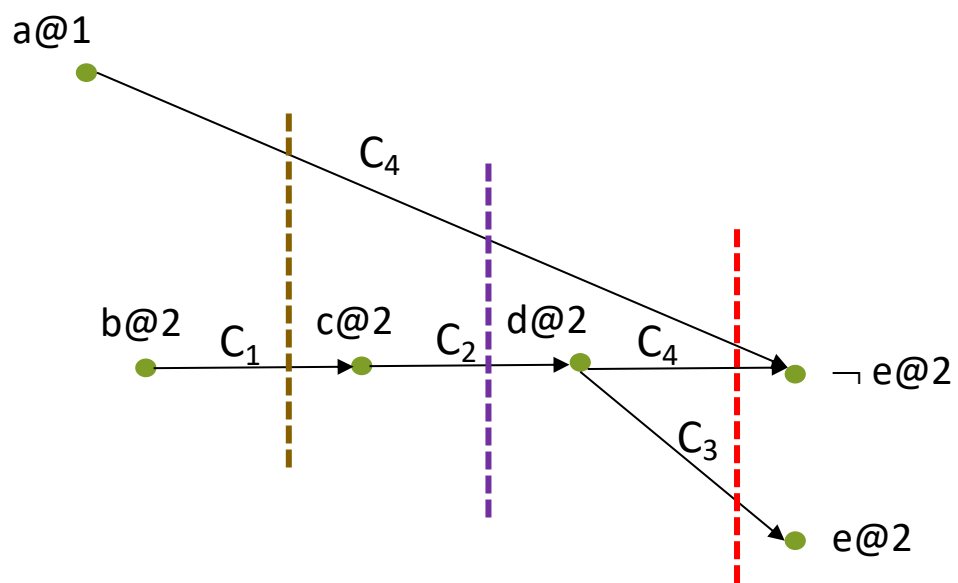
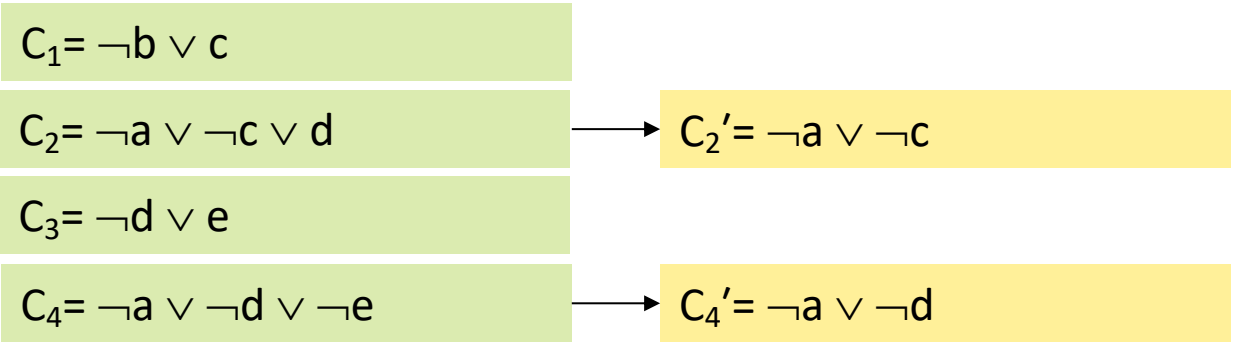
The idea: if an intermediate resolvent R subsumes clause $C \rightarrow$ replace C by R

Youssef Hamadi, Saïd Jabbour, Lakhdar Sais: Learning for Dynamic Subsumption. Int. J. Artif. Intell. Tools 19(4): 511-529 (2010)

HyoJung Han, Fabio Somenzi: On-the-Fly Clause Improvement. SAT 2009: 209-222

Applied by CryptoMinisat, Kissat, CaDiCaL, IntelSAT

On-the-fly Subsumption Example



Failed Literal Probing: the Basics

Chu Min Li, Anbulagan: Heuristics Based on Unit Propagation for Satisfiability Problems. IJCAI (1) 1997: 366-371

Daniel Le Berre: Exploiting the real power of unit propagation lookahead. Electron. Notes Discret. Math. 9: 59-80 (2001)

An *inprocessing* technique orthogonal to the backtrack search

- Used by CryptoMinisat, Kissat, CaDiCaL

Carried out at the beginning or after a restart

For every variable v

- Assign v and BCP
 - If contradiction, add the unit clause $(\neg v)$ and continue to the next loop iteration
- Assign $\neg v$ and BCP
 - If contradiction, add the unit clause (v) and continue to the next loop iteration
- For every literal l , implied by both v and $\neg v$, learn the unit clause (l)



Hyper-Binary Resolution

Fahiem Bacchus: *Enhancing Davis Putnam with Extended Binary Clause Reasoning*. *AAAI/IAAI 2002*: 613-619

Fahiem Bacchus, Jonathan Winter: *Effective Preprocessing with Hyper-Resolution and Equality Reduction*. *SAT 2003*: 341-355

Inês Lynce, João P. Marques Silva: *Probing-Based Preprocessing Techniques for Propositional Satisfiability*. *ICTAI 2003*

Hyper-binary resolution


$$\begin{array}{l} I_1 \vee I_2 \vee I_3 \vee \dots \vee I_n \\ \neg I_1 \vee I \\ \neg I_2 \vee I \\ \dots \\ \neg I_{n-1} \vee I \end{array} \left. \vphantom{\begin{array}{l} I_1 \vee I_2 \vee I_3 \vee \dots \vee I_n \\ \neg I_1 \vee I \\ \neg I_2 \vee I \\ \dots \\ \neg I_{n-1} \vee I \end{array}} \right\} \rightarrow I_n \vee I$$


Used during preprocessing (CryptoMinisat, Kissat)

- By manipulating the binary implication graph
 - A graph in which the edges correspond to binary clauses
 - Also handy to derive and merge equivalent literals

SAT Competition & Race Winners (CNF & Appl. & Seq. & Non-incr. & All-inst.)

2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
zChaff	Forklift	zChaff	SatELite GTI	MiniSat	RSAT	MiniSat	Precosat	Crypto MiniSat	Glucose
Moskewicz Madigan Zhao Zhang Malik	Goldberg Novikov	Moskewicz Madigan Zhao Zhang Malik	Eén Sörensson	Eén Sörensson	Pipatsrisa wat Darwiche	Eén Sörensson	Biere	Soos	Audemard Simon
2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Glucose	Lingeling	Lingeling	abcdSAT	Maple COMSPS	Maple LCMDist	Maple LCMDist ChronoBT	Maple LCMDist DLv3	Kissat	KissatMAB
Audemard Simon	Biere	Biere	Chen	Liang Oh Ganesh Czarnecki Poupart	Xiao Luo Li Manya Lu	Nadel Ryvchin	Kochemazov Zaikin Kondratiev Semenov	Biere Fazekas Fleury Heisinger	Cherif Habet Terrioux
2022	2023								
KissatMAB-HyWalk	SBVA-CaDiCaL								
Zheng He Chen Zhou Li	Haberlandt Green								

MiniSat-based:


Armin Biere's & derived:


Others:


Glucose

Gilles Audemard, Laurent Simon: On the Glucose SAT Solver. Int. J. Artif. Intell. Tools 27(1): 1840001:1-1840001:25 (2018)

A well-known and widely used solver

- Derived from Minisat
- Still in use in many incremental applications

Introduced the Literal Block Distance (LBD) measure for clause quality

LBD-based clause deletion and restart strategies

Changes in VSIDS implementation

Binary resolution during conflict analysis

Glucose's Literal Block Distance (LBD)

What makes a conflict clause a good one?

LBD: the number of decision levels in the clause

- Variables are propagated together: related and likely to be propagated together again

Recall: Locality

- Focus the search on the relevant data
- Learn strong clauses from the local context

LBD is:

- calculated when the conflict clause is created
- updated when a clause is visited during conflict analysis
- widely used by modern solvers

Glucose: LBD-based Clause Deletion

Delete half of the clauses based on LBD score

- Ties are broken, based on activity
- The deletion occurs every $2000 + 300 * x$ conflicts, where x is the number of clause deletions so far
- Deletion is postponed, if the clauses are “too good”
 - Median LBD $\leq 3 \rightarrow$ postponed by 1,000 conflicts
 - Highest LBD $\leq 5 \rightarrow$ postponed by 1,000 conflicts

Exceptions

- *Glue clauses* are kept forever
 - Glue clause: a clause with LBD=2
- Whenever the LBD goes down, the clause is kept for one more round
- Keep parent clauses (for correctness)
 - Glucose removes clauses not necessarily at decision level 0

Glucose: LBD-based Restart Strategy

Intuition: restart, when the latest clauses are bad (their LBD is too high)

When to restart:

- C: current LBD average over the latest 50 conflicts (since Glucose 2.1)
- G: global LBD average
- Restart when $C * 0.8 > G$
 - 0.8: since Glucose 2.1

Too aggressive:

- Yields restarts every 50 conflicts
- Might be performed too close to a satisfying assignment

Postpone restart when the number of assigned literals grows suddenly

- A^C : current average of assigned literals when a conflict occurs (over the latest 5000 conflicts)
- A^G : global average of assigned literals when a conflict occurs
- Postpone when $1.4 * A^C > A^G$

VSIDS in Glucose

VSIDS increments activity by an exponentially increasing $(g=1/f)^{\#conflict}$

- Minisat: $f=0.95 \rightarrow g \approx 1.05$

Since Glucose 2.3

- every 5000th conflict, f is increased by 0.01, starting at 0.8 until 0.95 is reached

More dynamic at the beginning of the search, stabilizes later

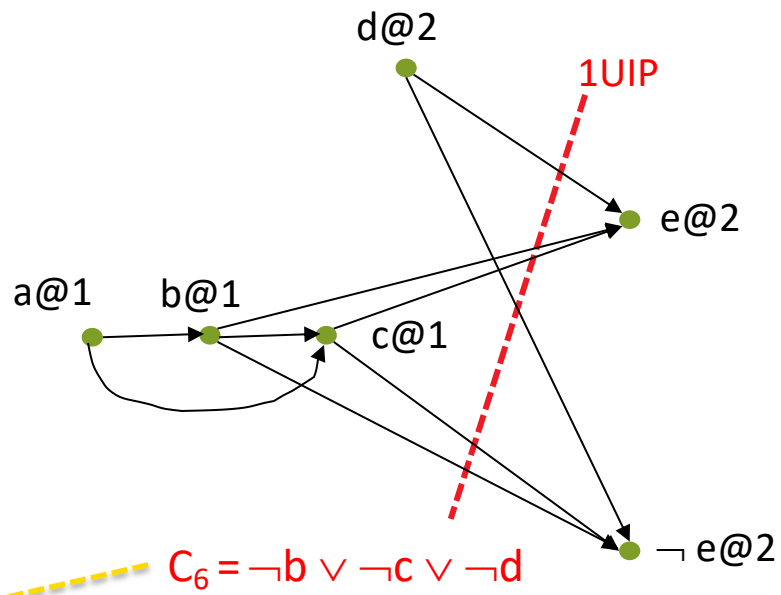
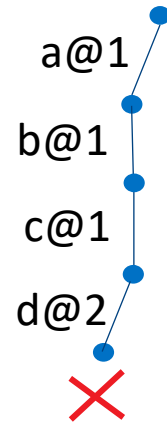
Binary Resolution during Conflict Analysis

The idea: given a learnt clause C , remove unnecessary literals from C by resolution with satisfied (non-parent!) binary clauses

Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure, Lakhdar Sais: RCL: Reduce learnt clauses. https://baldur.itk.kit.edu/sat-race-2010/descriptions/solver_10.pdf, 2010

Binary Resolution Example

- $C_1 = \neg a \vee b$
- $C_2 = \neg a \vee \neg b \vee c$
- $C_3 = \neg b \vee \neg c \vee \neg d \vee e$
- $C_4 = \neg b \vee \neg c \vee \neg d \vee \neg e$
- $C_5 = b \vee \neg c$



$C_7 = \neg c \vee \neg d$

- Minimization is unapplicable
- Binary resolution with C_5 works!
 - C_5 is a satisfied non-parent clause
- The implementation goes over binary watches of C 's literals

Binary Resolution Heuristic

Applied for newly learnt clauses for which both the following conditions hold:

- Maximal size of 30
- Maximal LBD of 6

Standard since Glucose

Glucose in Non-Incremental Mode


Preprocess() // Variable elimination & subsumption & self-subsuming resolution


While (true)


- Literal I = Decide() // Updated variable-based EVSIDS
- BCP(I)
- If (conflict)
 - ConflictAnalysisLoop() // Minimized 1UIP + binary resolution
- If (learned an empty clause)
 - Return UNSAT
- If (all the variables are assigned)
 - Return SAT
- Occasionally, restart // LBD-based
- Occasionally, delete conflict clauses // LBD-based

SAT Competition & Race Winners (CNF & Appl. & Seq. & Non-incr. & All-inst.)

2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
zChaff	Forklift	zChaff	SatELite GTI	MiniSat	RSAT	MiniSat	Precosat	Crypto MiniSat	Glucose
Moskewicz Madigan Zhao Zhang Malik	Goldberg Novikov	Moskewicz Madigan Zhao Zhang Malik	Eén Sörensson	Eén Sörensson	Pipatsrisa wat Darwiche	Eén Sörensson	Biere	Soos	Audemard Simon
2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Glucose	Lingeling	Lingeling	abcdSAT	Maple COMSPS	Maple LCMDist	Maple LCMDist ChronoBT	Maple LCMDist ChronoBT DLv3	Kissat	KissatMAB
Audemard Simon	Biere	Biere	Chen	Liang Oh Ganesh Czarnecki Poupart	Xiao Luo Li Manya Lu	Nadel Ryvchin	Kochemazov Zaikin Kondratiev Semenov	Biere Fazekas Fleury Heisinger	Cherif Habet Terrioux
2022	2023								
KissatMAB-HyWalk	SBVA-CaDiCaL								
Zheng He Chen Zhou Li	Haberlandt Green								

MiniSat-based:



Armin Biere's & derived:



Others:


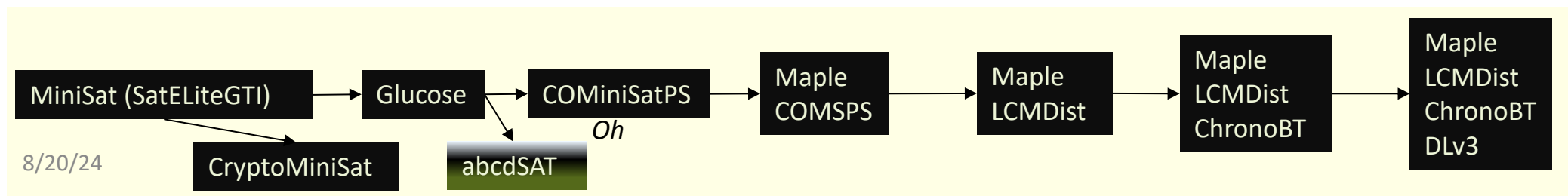
SAT Competition & Race Winners (CNF & Appl. & Seq. & Non-incr. & All-inst.)

2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
zChaff	Forklift	zChaff	SatELite GTI	MiniSat	RSAT	MiniSat	Precosat	Crypto MiniSat	Glucose
Moskewicz Madigan Zhao Zhang Malik	Goldberg Novikov	Moskewicz Madigan Zhao Zhang Malik	Eén Sörensson	Eén Sörensson	Pipatsrisa wat Darwiche	Eén Sörensson	Biere	Soos	Audemard Simon
2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Glucose	Lingeling	Lingeling	abcdSAT	Maple COMSPS	Maple LCMDist	Maple LCMDist ChronoBT	Maple LCMDist ChronoBT DLv3	Kissat	KissatMAB
Audemard Simon	Biere	Biere	Chen	Liang Oh Ganesh Czarnecki Poupart	Xiao Luo Li Manya Lu	Nadel Ryvchin	Kochemazov Zaikin Kondratiev Semenov	Biere Fazekas Fleury Heisinger	Cherif Habet Terrioux

MiniSat-based:


Armin Biere's & derived:


Others:




COMiniSatPS

Chanseok Oh: *Between SAT and UNSAT: The Fundamental Difference in CDCL SAT.* SAT 2015: 307-323

UNSAT Instance

- A proof is required
 - Strong conflict clauses are essential
- Refocus on locally useful variables & clauses to learn stronger clauses →
- **Aggressive** restart strategy and VSIDS score update

SAT Instance

- Can be solved instantly with a perfect oracle
 - Less need for conflict clauses in practice (demonstrated experimentally)
- Let the solver complete finding a potential model →
- **Slow** restart strategy and VSIDS score update

COMiniSatPS: combining SAT & UNSAT stages in every (long enough) solver invocation

Clause deletion: 3-tiered scheme

COMiniSatPS: Combining SAT & UNSAT Stages

UNSAT stage warm-up: 10,000 initial conflicts

$C = 100$

While (no solution)

- SAT stage: C conflicts
- UNSAT stage: $2 * C$ conflicts
- $C = C * 1.1$

	SAT Stage	UNSAT Stage
Restarts	No-restart	Glucose
VSIDS Score Increment	$(g=1/0.999)^{\#conflict}$	$(g=1/0.95)^{\#conflict}$

COMiniSatPS: 3-Tiered Clause Management & Deletion

Core: kept forever

- $LBD \leq 3$ (at creation or during conflict analysis)
- LBD threshold goes up to 5, if at 100,000 conflicts, there are <100 Core clauses

Tier2: bad clauses are relegated to Local

- $3 < LBD \leq 6$ (at creation or during conflict analysis)
- Every 10,000 conflicts, clauses not touched for 30,000 conflicts are relegated

Local: bad clauses are deleted

- $LBD > 6$ (at creation)
- Every 15,000 conflicts, the less active half of the clauses is deleted

SAT Competition & Race Winners (CNF & Appl. & Seq. & Non-incr. & All-inst.)

2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
zChaff	Forklift	zChaff	SatELite GTI	MiniSat	RSAT	MiniSat	Precosat	Crypto MiniSat	Glucose
Moskewicz Madigan Zhao Zhang Malik	Goldberg Novikov	Moskewicz Madigan Zhao Zhang Malik	Eén Sörensson	Eén Sörensson	Pipatsrisa wat Darwiche	Eén Sörensson	Biere	Soos	Audemard Simon
2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Glucose	Lingeling	Lingeling	abcdSAT	Maple COMSPS	Maple LCMDist	Maple LCMDist ChronoBT	Maple LCMDist ChronoBT DLv3	Kissat	KissatMAB
Audemard Simon	Biere	Biere	Chen	Liang Oh Ganesh Czarnecki Poupart	Xiao Luo Li Manya Lu	Nadel Ryvchin	Kochemazov Zaikin Kondratiev Semenov	Biere Fazekas Fleury Heisinger	Cherif Habet Terrioux
2022	2023								
KissatMAB-HyWalk	SBVA-CaDiCaL								
Zheng He Chen Zhou Li	Haberlandt Green								

MiniSat-based:


Armin Biere's & derived:


Others:

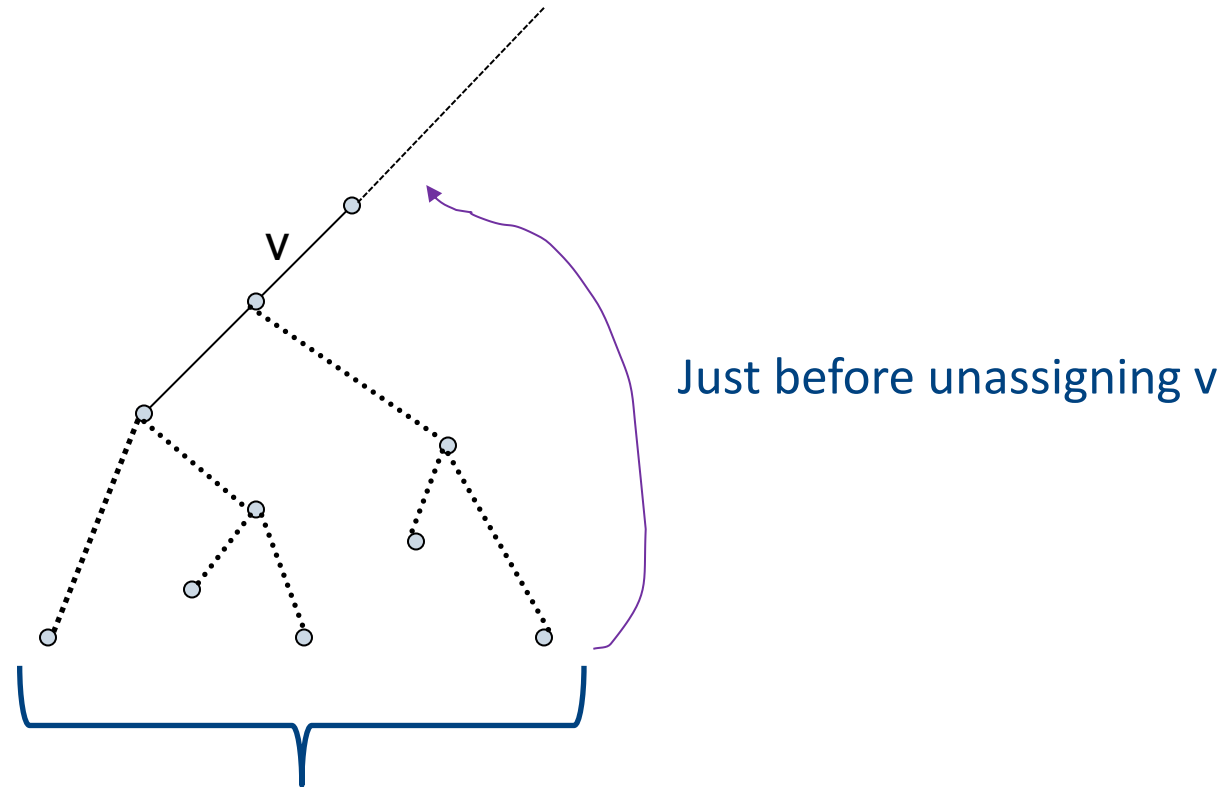

Learning Rate Based (LRB) Decision Heuristic

Jia Hui Liang, Vijay Ganesh, Pascal Poupart, Krzysztof Czarnecki:
Learning Rate Based Branching Heuristic for SAT Solvers. SAT 2016: 123-140

Similarly to VSIDS, choose and pick variables, based on activity

Boost variables, which made impact during their **latest assignment term**

Learning Rate Based (LRB)



C[v]: #conflicts in which v's score was updated during the latest assignment term

Visited during conflict analysis or belongs to the parents of literals in the new conflict clause

age[v]: the number of conflicts during the latest assignment term

LR[v]: $\text{age}[v] / C[v]$

LRB Details

When a variable is unassigned, adjust its activity, based on $LR[v]$

- $const\ double\ oldActivity \leftarrow activity[var];$
- $activity[var] \leftarrow P * LR[var] + (1 - P) * oldActivity;$
- P: from 0.4 down to 0.06, decremented by 0.000001 every conflict when LRB is used
 - Down to 0.06 after 340,000 conflicts when LRB is used

The update algorithm uses Exponential Recency Weighted Average (ERWA)

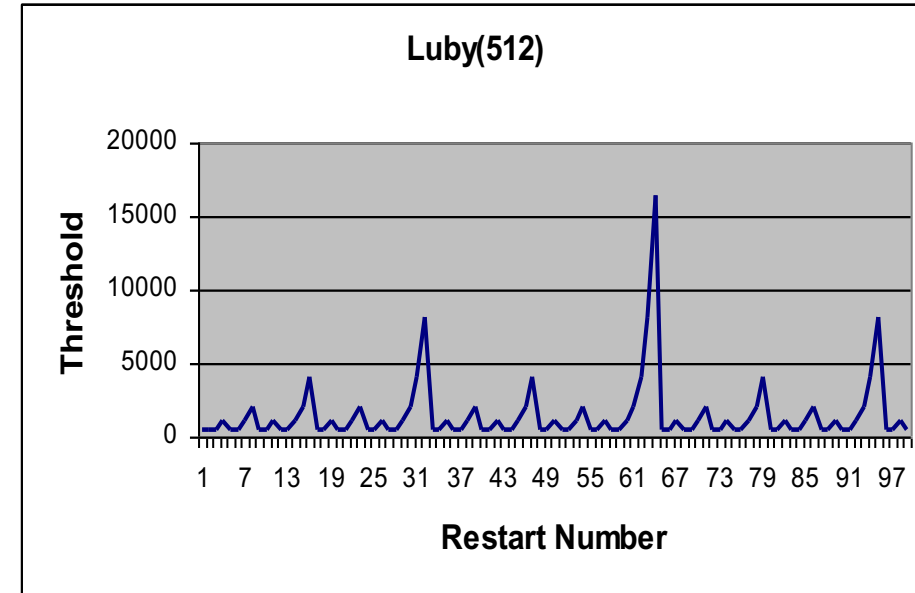
- used in nonstationary Multi-Armed Bandit (MAB) problems to estimate the average reward of different actions

Sutton, R. S., and Barto, A. G.: Reinforcement learning: An introduction, volume 1. MIT press Cambridge, 1998.

Summary: LRB considers the “local context” of the latest assignment term, more so in the beginning of the search

MapleCOMSPS: Combining SAT & UNSAT Stages

1. UNSAT stage warm-up: 10,000 initial conflicts
2. SAT stage till 2,500 sec. from the beginning
3. UNSAT stage forever





	SAT Stage	UNSAT Stage
Restarts	Luby	Glucose
Decision Heuristic	LRB	Glucose's EVSIDS (0.8→0.95)

SAT Competition & Race Winners (CNF & Appl. & Seq. & Non-incr. & All-inst.)

2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
zChaff	Forklift	zChaff	SatELite GTI	MiniSat	RSAT	MiniSat	Precosat	Crypto MiniSat	Glucose
Moskewicz Madigan Zhao Zhang Malik	Goldberg Novikov	Moskewicz Madigan Zhao Zhang Malik	Eén Sörensson	Eén Sörensson	Pipatsrisa wat Darwiche	Eén Sörensson	Biere	Soos	Audemard Simon
2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Glucose	Lingeling	Lingeling	abcdSAT	Maple COMSPS	Maple LCMDist	Maple LCMDist ChronoBT	Maple LCMDist ChronoBT DLv3	Kissat	KissatMAB
Audemard Simon	Biere	Biere	Chen	Liang Oh Ganesh Czarnecki Poupart	Xiao Luo Li Manya Lu	Nadel Ryvchin	Kochemazov Zaikin Kondratiev Semenov	Biere Fazekas Fleury Heisinger	Cherif Habet Terrioux
2022	2023								
KissatMAB-HyWalk	SBVA-CaDiCaL								
Zheng He Chen Zhou Li	Haberlandt Green								

MiniSat-based:


Armin Biere's & derived:


Others:


MapleLCMDist

DISTANCE decision heuristic for the initial stage (first 50,000 conflicts)

Fan Xiao, Chu-Min Li, Mao Luo, Felip Manyà, Zhipeng Lü, Yu Li: A branching heuristic for SAT solvers based on complete implication graphs. Sci. China Inf. Sci. 62(7): 72103:1-72103:13 (2019)

Vivification aka distillation aka learnt-clause-minimization

- *Cédric Piette, Youssef Hamadi, Lakhdar Sais: Vivifying Propositional Clausal Formulae. ECAI 2008: 525-529*
- *HyoJung Han, Fabio Somenzi: Alembic: An Efficient Algorithm for CNF Preprocessing. DAC 2007: 582-587*
- *Mao Luo, Chu-Min Li, Fan Xiao, Felip Manyà, Zhipeng Lü: An Effective Learnt Clause Minimization Approach for CDCL SAT Solvers. IJCAI 2017: 703-711*
- *Chu-Min Li, Fan Xiao, Mao Luo, Felip Manyà, Zhipeng Lü, Yu Li: Clause vivification by unit propagation in CDCL SAT solvers. Artif. Intell. 279 (2020)*

DISTANCE Decision Heuristic

Observation: at the beginning, variable scores are inaccurate, because they are based on very few conflicts

DISTANCE Heuristic:

- Yet another separate “activity” priority queue, used for the first 50,000 conflicts
- Increment v 's activity, depending on the longest distance between v and the conflict
 - The closer v to the conflict, the more v contributes

DISTANCE: Longest Distance to Conflict

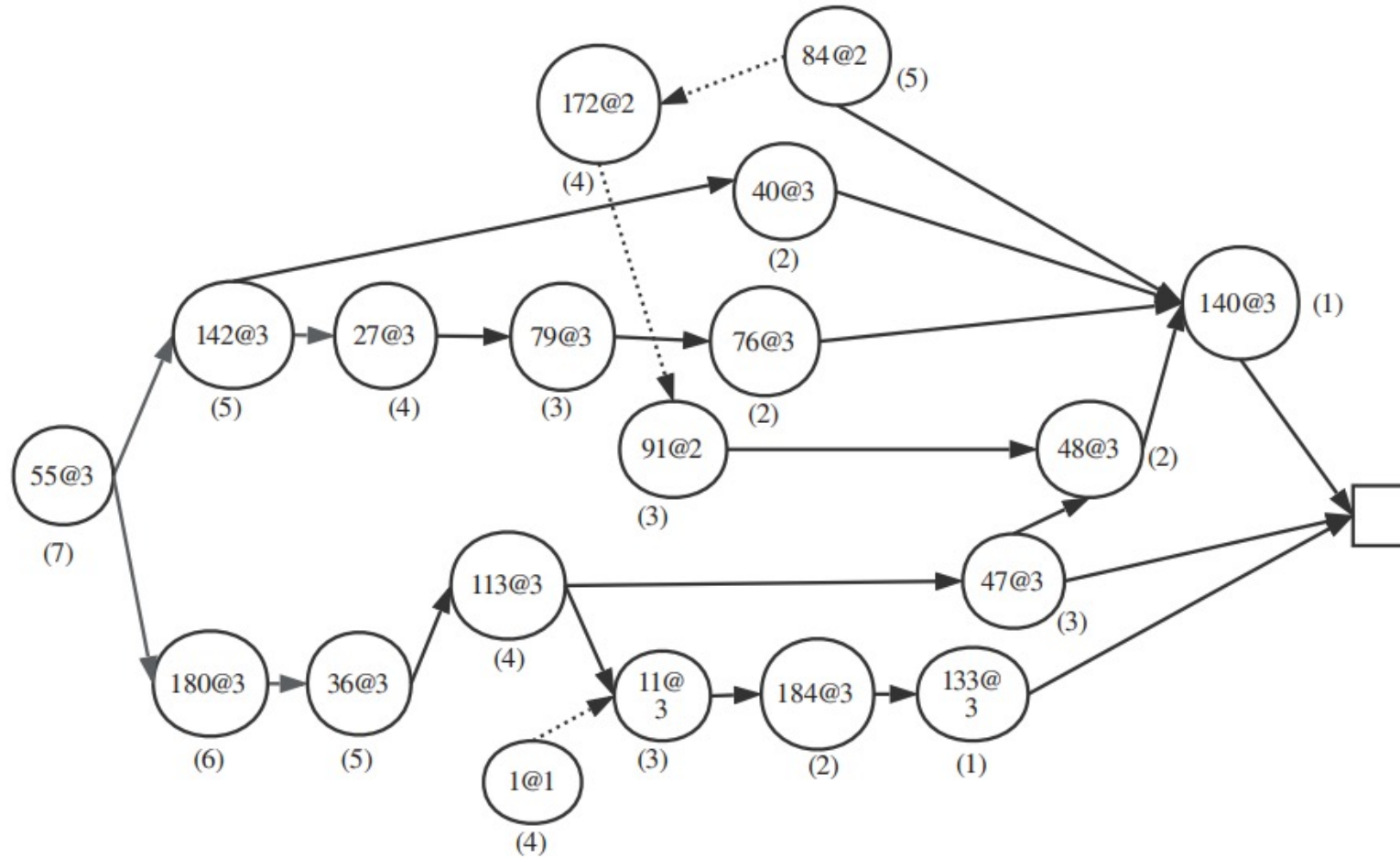


Figure 1 Complete implication graph implying the empty clause from clause $x_{70} \vee \neg x_{24} \vee \neg x_{67}$. Numbers in parentheses below vertices represent the longest distance from the vertex to \square .

DISTANCE: Details

$\text{distAct}[v]$ is the DISTANCE activity, initialized to 0 for every v

$\text{longDist}[v]$: the longest distance to the conflict for the current conflict

When v contributes to a conflict, $\text{distAct}[v]$ is incremented by $\text{inc} \times 1 / \text{longDist}[v]$

- inc : give more weight to recent conflicts
- Start: $\text{inc} \leftarrow 1$
- After each conflict: $\text{inc} \leftarrow \text{inc} / 0.95$

Vivification: an Inprocessing Algorithms

At decision level 0 (*inprocessing*), go over the clauses and simplify them as follows:

Let $C = c_1 \vee c_2 \vee \dots \vee c_n$ be a clause

For i in $[1, 2, \dots, n]$


- If c_i is assigned 0, remove c_i from C and continue to the next loop iteration
 - $\neg c_1 \wedge \dots \wedge \neg c_{i-1} \Rightarrow \neg c_i \cong c_1 \vee \dots \vee c_{i-1} \vee \neg c_i$
 - Resolve C with $c_1 \vee \dots \vee c_{i-1} \vee \neg c_i$
- If c_i is assigned 1, replace C by $c_1 \vee \dots \vee c_{i-1} \vee c_i$ and stop
 - $\neg c_1 \wedge \dots \wedge \neg c_{i-1} \Rightarrow c_i \cong c_1 \vee \dots \vee c_{i-1} \vee c_i$
 - $c_1 \vee \dots \vee c_{i-1} \vee c_i$ subsumes C
- Assign the literal $\neg c_i$
- Run BCP


Controlling Vivification in MapleLCMDist


- Vivify only the learnt clauses
- When to apply vivification?
 - Immediately after each clause database reduction
 - #clauses: $2000 + 2 \times 300 \times \text{database-reductions-so-far}$
- When to vivify clause C ?
 - C has not yet been vivified, and
 - C is a learnt with a small LBD: consider only half of the learnt's, sorted by LBDs
- What is the best literal order in literal C
 - Default (as maintained by the solver)
 - More active literals come earlier because of WL management

SAT Competition & Race Winners (CNF & Appl. & Seq. & Non-incr. & All-inst.)

2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
zChaff	Forklift	zChaff	SatELite GTI	MiniSat	RSAT	MiniSat	Precosat	Crypto MiniSat	Glucose
Moskewicz Madigan Zhao Zhang Malik	Goldberg Novikov	Moskewicz Madigan Zhao Zhang Malik	Eén Sörensson	Eén Sörensson	Pipatsrisa wat Darwiche	Eén Sörensson	Biere	Soos	Audemard Simon
2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Glucose	Lingeling	Lingeling	abcdSAT	Maple COMSPS	Maple LCMDist	Maple LCMDist ChronoBT	Maple LCMDist ChronoBT DLv3	Kissat	KissatMAB
Audemard Simon	Biere	Biere	Chen	Liang Oh Ganesh Czarnecki Poupart	Xiao Luo Li Manya Lu	Nadel Ryvchin	Kochemazov Zaikin Kondratiev Semenov	Biere Fazekas Fleury Heisinger	Cherif Habet Terrioux
2022	2023								
KissatMAB-HyWalk	SBVA-CaDiCaL								
Zheng He Chen Zhou Li	Haberlandt Green								

MiniSat-based:


Armin Biere's & derived:


Others:


Up-to-date Conflict Analysis Algorithm

Covers GRASP & Chaff & Modern Solvers

1. Backtrack before conflict analysis: backtrack to the conflict level δ , if required
 - Required in GRASP and called Non-Chronological Backtracking (NCB) in GRASP
 - Not required in Chaff: current decision level \equiv conflict level
2. Learn an asserting clause $C=[c_1^{@ \delta}, c_2^{@ \beta < \delta}, c_3^{@ \leq \beta}, \dots, c_i^{@ \leq \beta}, \dots, c_{|C|}^{@ \leq \beta}]$
 - 1UIP clause in both GRASP & Chaff
3. Optionally, learn other clauses
 - GRASP: a clause for every other UIP of the conflict decision level
4. Backtrack: backtrack to a level in $[\beta, \beta+1, \dots, \delta-1]$ -- makes the asserting clause unit
 - GRASP -- always $\delta-1$: Chronological Backtracking (CB) in today's terminology
 - Chaff -- always β : Non-Chronological Backtracking (NCB) in today's terminology
5. Flip c_1 by implying it in C and run BCP

Conflict Analysis Evolvment

1996

2001

2018

GRASP

Chaff

Chaff's algorithm is the state-of-the-art

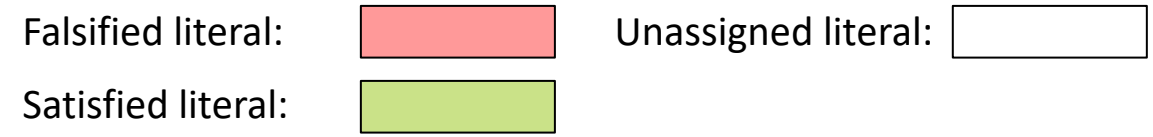
Maple_LCM
_Dist_Chro
noBT

Maple_LCM_Dist_ChronoBT: the return of Chronological Backtracking (CB)

Alexander Nadel, Vadim Ryvchin: Chronological Backtracking. *SAT 2018*: 111-121

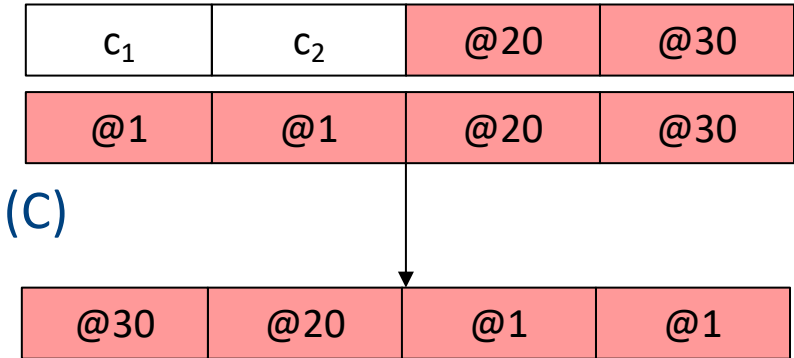
- A backtracking heuristic choosing between CB and NCB
 - First 4,000 conflicts (warm-up): NCB
 - After 4,000 conflicts: NCB iff backtrack level - conflict level ≤ 100
 - Today: Maple-based solvers & Cryptominisat & Kissat (no warm-up in Kissat)
 - CaDiCal & IntelSAT also combine NCB & CB, but differently
- CB algorithm is similar to GRASP's
- Integrating CB with post-GRASP BCP data structures turned out to be highly non-trivial
 - Because of simultaneous propagation at several levels
 - BCP must be adjusted to prevent correctness & performance issues
 - Useful BCP invariants are still violated!

Integrating CB and BCP



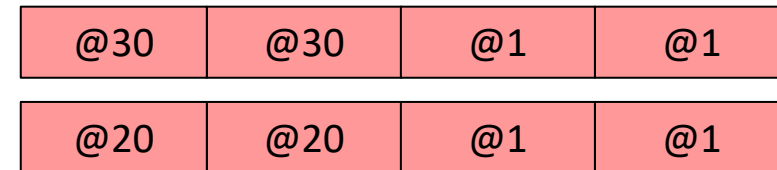
Example of a necessary adjustment

- $\neg c_1$ and $\neg c_2$ are assigned @1 \rightarrow the clause is visited by BCP twice
 - Impossible with NCB, where the assigned level is always $\geq \max_level(C)$
- Need to swap literals to have two highest literals watched
- Essential for correctness – in order not to miss conflicts after backtracking!



Useful invariants are still violated even with the adjustments:

- **lowest implication:** no assigned literal can be implied at a lower level
- **lowest conflict:** every conflict, BCP returns a clause falsified at the lowest possible level



Intel® SAT Solver (IntelSAT): a new formally proven BCP alg. with a possible solution

Alexander Nadel: *Introducing Intel® SAT Solver. SAT 2022.*

As of 2022: expecting new formal frameworks [SAT'2024: Robin Coutelier, Mathias Fleury and Laura Kovács](#)
Lazy Reimplication in Chronological Backtracking ([abstract](#))

SAT Competition & Race Winners (CNF & Appl. & Seq. & Non-incr. & All-inst.)

2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
zChaff	Forklift	zChaff	SatELite GTI	MiniSat	RSAT	MiniSat	Precosat	Crypto MiniSat	Glucose
Moskewicz Madigan Zhao Zhang Malik	Goldberg Novikov	Moskewicz Madigan Zhao Zhang Malik	Eén Sörensson	Eén Sörensson	Pipatsrisa wat Darwiche	Eén Sörensson	Biere	Soos	Audemard Simon
2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Glucose	Lingeling	Lingeling	abcdSAT	Maple COMSPS	Maple LCMDist	Maple LCMDist ChronoBT	Maple LCMDist ChronoBT DLv3	Kissat	KissatMAB
Audemard Simon	Biere	Biere	Chen	Liang Oh Ganesh Czarnecki Poupart	Xiao Luo Li Manya Lu	Nadel Ryvchin	Kochemazov Zaikin Kondratiev Semenov	Biere Fazekas Fleury Heisinger	Cherif Habet Terrioux
2022	2023								
KissatMAB-HyWalk	SBVA-CaDiCaL								
Zheng He Chen Zhou Li	Haberlandt Green								

MiniSat-based:


Armin Biere's & derived:


Others:


MapleLCMDistChronoBTDLv3

Duplicate Learnts: screen learnt clauses and add duplicates as permanent clauses

Stepan Kochemazov, Oleg Zaikin, Alexander A. Semenov, Victor Kondratiev: Speeding Up CDCL Inference with Duplicate Learnt Clauses. ECAI 2020: 339-346

Duplicate Learnts (DL)

Uses clause hash table

Hashes learnts with $LBD(C) \leq lbd_limit=12$

Repeated once or twice \rightarrow Tier2; Repeated 3-times \rightarrow Core forever

Hash size limit = 500,000. When the limit is reached:

- Purge all clauses repeated once
- Increase the limit by 10%

DL works well with vivification

- Vivification may reduce the LBD score of clauses to make them eligible for DL

MapleLCMDistChronoBTDLv3

Preprocess() // Variable elimination & subsumption & self-subsuming resolution


While (true)


- Literal I = Decide() // DISTANCE: 50,000 conf. → LRB: 2,500 sec. from start → EVSIDS
- BCP(I)
- If (conflict)
 - ConflictAnalysisLoop() // minimized 1UIP + binary resolution; Combine CB & NCB
- If (learned an empty clause)
 - Return UNSAT
- If (all the variables are assigned)
 - Return SAT
- Occasionally, restart // Luby: 2,500 sec. from start → LBD-based
- Occasionally, delete conflict clauses // 3-tiered LBD-based
- Occasionally, inprocess // vivification + restore duplicates

SAT Competition & Race Winners (CNF & Appl. & Seq. & Non-incr. & All-inst.)

2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
zChaff	Forklift	zChaff	SatELite GTI	MiniSat	RSAT	MiniSat	Precosat	Crypto MiniSat	Glucose
Moskewicz Madigan Zhao Zhang Malik	Goldberg Novikov	Moskewicz Madigan Zhao Zhang Malik	Eén Sörensson	Eén Sörensson	Pipatsrisa wat Darwiche	Eén Sörensson	Biere	Soos	Audemard Simon
2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Glucose	Lingeling	Lingeling	abcdSAT	Maple COMSPS	Maple LCMDist	Maple LCMDist ChronoBT	Maple LCMDist ChronoBT DLv3	Kissat	KissatMAB
Audemard Simon	Biere	Biere	Chen	Liang Oh Ganesh Czarnecki Poupart	Xiao Luo Li Manya Lu	Nadel Ryvchin	Kochemazov Zaikin Kondratiev Semenov	Biere Fazekas Fleury Heisinger	Cherif Habet Terrioux
2022	2023								
KissatMAB-HyWalk	SBVA-CaDiCaL								
Zheng He Chen Zhou Li	Haberlandt Green								

MiniSat-based:


Armin Biere's & derived:


Others:


Kissat vs. Maple-based Solvers

Variable decision heuristic: VMTF for the UNSAT stage

- Variable-Move-To-Front (VMTF): store variables in a conflict-driven stack and pick the top-most variable from there
L. Ryan: Efficient algorithms for clause-learning SAT solvers. Masters thesis, Simon Fraser University, February 2004

Advanced inprocessing

- Vivification: the only technique available also in Maple
- Failed literal probing and hyper-binary resolution (*removed in 2022*)
- Bounded variable elimination, equivalent literal substitution, blocked clause elimination, bounded variable addition, ...

*Armin Biere, [Matti Järvisalo](#), [Benjamin Kiesl](#): **Preprocessing in SAT Solving**. *Handbook of Satisfiability 2021*: 391-435*

Low-level optimizations, not present in Maple

- Binary clauses are fully inlined
- Watch lists are contained in a contiguous buffer (requires occasional defragmentation)

New polarity selection algorithm: local search & target phases

- Subsequent slides: skipping today, tune in for Armin's tutorial tomorrow!

SAT Competition & Race Winners (CNF & Appl. & Seq. & Non-incr. & All-inst.)

2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
zChaff	Forklift	zChaff	SatELite GTI	MiniSat	RSAT	MiniSat	Precosat	Crypto MiniSat	Glucose
Moskewicz Madigan Zhao Zhang Malik	Goldberg Novikov	Moskewicz Madigan Zhao Zhang Malik	Eén Sörensson	Eén Sörensson	Pipatsrisa wat Darwiche	Eén Sörensson	Biere	Soos	Audemard Simon
2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Glucose	Lingeling	Lingeling	abcdSAT	Maple COMSPS	Maple LCMDist	Maple LCMDist ChronoBT	Maple LCMDist ChronoBT DLv3	Kissat	KissatMAB
Audemard Simon	Biere	Biere	Chen	Liang Oh Ganesh Czarnecki Poupart	Xiao Luo Li Manya Lu	Nadel Ryvchin	Kochemazov Zaikin Kondratiev Semenov	Biere Fazekas Fleury Heisinger	Cherif Habet Terrioux
2022	2023								
KissatMAB-HyWalk	SBVA-CaDiCaL								
Zheng He Chen Zhou Li	Haberlandt Green								

MiniSat-based:



Armin Biere's & derived:



Others:



KissatMAB

The change is in the decision heuristic

Every restart, the solver chooses between:

1. VSIDS
2. CHB: predecessor of LRB

Using Multi-Armed Bandit (MAB) framework

Reward function: *“we choose a reward function that estimates the ability of a heuristic to reach conflicts quickly and efficiently.”*

The reward function to maximize: $\log_2(\text{decisions}) / \text{dVars}$


(a) decisions : the number of decisions


(b) dVars: the number of variables used as decision variables at least once


As many decisions as possible over the same variables → locality principle!

SAT Competition & Race Winners (CNF & Appl. & Seq. & Non-incr. & All-inst.)

2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
zChaff	Forklift	zChaff	SatELite GTI	MiniSat	RSAT	MiniSat	Precosat	Crypto MiniSat	Glucose
Moskewicz Madigan Zhao Zhang Malik	Goldberg Novikov	Moskewicz Madigan Zhao Zhang Malik	Eén Sörensson	Eén Sörensson	Pipatsrisa wat Darwiche	Eén Sörensson	Biere	Soos	Audemard Simon
2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Glucose	Lingeling	Lingeling	abcdSAT	Maple COMSPS	Maple LCMDist	Maple LCMDist ChronoBT	Maple LCMDist ChronoBT DLv3	Kissat	KissatMAB
Audemard Simon	Biere	Biere	Chen	Liang Oh Ganesh Czarnecki Poupart	Xiao Luo Li Manya Lu	Nadel Ryvchin	Kochemazov Zaikin Kondratiev Semenov	Biere Fazekas Fleury Heisinger	Cherif Habet Terrioux
2022	2023								
KissatMAB-HyWalk	SBVA-CaDiCaL								
Zheng He Chen Zhou Li	Haberlandt Green								

MiniSat-based:


Armin Biere's & derived:


Others:



KissatMAB-HyWalk


The change is in the local search component


“Combines BandSAT, FPS, and some other local search algorithms with different random walking or say local optimal escaping strategies”

SAT Competition & Race Winners (CNF & Appl. & Seq. & Non-incr. & All-inst.)

2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
zChaff	Forklift	zChaff	SatELite GTI	MiniSat	RSAT	MiniSat	Precosat	Crypto MiniSat	Glucose
Moskewicz Madigan Zhao Zhang Malik	Goldberg Novikov	Moskewicz Madigan Zhao Zhang Malik	Eén Sörensson	Eén Sörensson	Pipatsrisa wat Darwiche	Eén Sörensson	Biere	Soos	Audemard Simon
2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Glucose	Lingeling	Lingeling	abcdSAT	Maple COMSPS	Maple LCMDist	Maple LCMDist ChronoBT	Maple LCMDist ChronoBT DLv3	Kissat	KissatMAB
Audemard Simon	Biere	Biere	Chen	Liang Oh Ganesh Czarnecki Poupart	Xiao Luo Li Manya Lu	Nadel Ryvchin	Kochemazov Zaikin Kondratiev Semenov	Biere Fazekas Fleury Heisinger	Cherif Habet Terrioux
2022	2023								
KissatMAB-HyWalk	SBVA-CaDiCaL								
Zheng He Chen Zhou Li	Haberlandt Green								

MiniSat-based:


Armin Biere's & derived:


Others:


CaDiCaL and SBVA-CaDiCaL

CaDiCaL

- An incremental SAT solver
- Around since 2017
- 64-bit clause indexing
 - Allows one to handle huge instances, but increases the memory footprint
- First open-source solver with incremental preprocessing (since 2019)

*Katalin Fazekas, Armin Biere, Christoph Scholl: **Incremental Inprocessing in SAT Solving**. SAT 2019: 136-154*

- We have a closed-sourced solver Fiver with incremental preprocessing (since 2012)
*Alexander Nadel, Vadim Ryvchin, Ofer Strichman: **Preprocessing in Incremental SAT**. SAT 2012: 256-269*
- Fiver supports SatELite, whereas CaDiCaL supports more techniques

- Details: **CaDiCaL 2.0** Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks and Florian Pollitt, CAV'24

SBVA-CaDiCaL

- Smart implementation of Bounded Variable Addition (BVA) inprocessing on top of CaDiCaL

Incremental Solvers after Minisat

Glucose

A dedicated feature: ignores assumption literals in LBD calculations

CryptoMinisat

Maple-based SC winners since MapleCOMSPS (SC'16) aren't incremental, but MergeSAT is:

*Norbert Manthey: **The MergeSat Solver**. SAT 2021*

SC incremental tracks:

- 2017 – won by the 2015 SR winner abcdSAT
- 2020 – won by CryptoMinisat

Kissat isn't incremental, but CaDiCal is

- 2024 – CaDiCaL 2.0 shown to outperform CryptoMinisat on SC'20 and some other incremental bench's

Incremental Solvers after Minisat: IntelSAT

IntelSAT Alexander Nadel: *Introducing Intel(R) SAT Solver*. SAT 2022: 8:1-8:23

- An open-source CDCL solver written from scratch in C++20
- Optimized towards incremental app's with rapid mostly SAT queries
 - Triggered by performance needs of industrial optimization problems
- Application in the paper: anytime unweighted MaxSAT -- improves the state-of-the-art
- Various applications @ Intel: placement, routing, scheduling, etc.
- Incremental Lazy Backtracking (ILB):
 - Upon a new incremental query Solve(A), backtrack to the highest possible level, rather than 0
 - Let k be the maximal decision level, whose decision literal appears in A
 - ILB backtracks to k (instead of 0)
 - Implemented in CaDiCaL 2.0
- Chronological backtracking with reimplication: guarantees lowest implication & lowest conflict

Is Progress on SC Benchmarks Relevant to Incremental SAT?

Incremental SAT: no progress since 2013 [KIS, SAT'21]

- Glucose 3.0 (2013) is largely on par with a leading Maple-based solver (2021) on
 - Satisfiability-based MaxSAT: mostly SAT queries
 - Unsatisfiability-based MaxSAT: mostly UNSAT queries
 - Minimal Unsatisfiable Core (MUC) Extraction: mixed queries
- None of the 5 latest techniques which improve non-incremental SAT has a significant positive impact on incremental SAT

My experience at Intel till 2021: no progress on industrial optimization problems

- e.g., placement in physical design [CNR, TACAS'21]
- Triggered IntelSAT development in 2021

CaDiCaL 2.0 CAV'24 paper: progress on a variety of benchmarks, but no results on [KIS, SAT'21] bench's

My intuitive take (without rigorous empirical evidence) -- it depends on the application:

- *Heavy SAT invocations (e.g., BMC): non-incr. progress is relevant; incremental inprocessing can be helpful*
- *Light SAT invocations (e.g., MaxSAT, MUC extraction, PDR): non-incr. progress is less relevant; ILB can be helpful*

[CNR, TACAS'21] Aviad Cohen, Alexander Nadel, Vadim Ryvchin: **Local Search with a SAT Oracle for Combinatorial Optimization**. *TACAS (2) 2021*: 87-104

[KIS, SAT'21] Stepan Kochemazov, Alexey Ignatiev, João Marques-Silva: **Assessing Progress in SAT Solvers Through the Lens of Incremental SAT**. *SAT 2021*: 280-298

Conclusion

SAT is an unresolved mystery!

Yet, SAT solvers are scalable widely used tools

Main goals for today:

- Explain how modern SAT solvers work
- Convey intuition why they work in practice
- Provide examples of applying SAT

