SAT Solving
--- Basis and CDCL

Shaowei Cai

Institute of Software, Chinese Academy of Sciences

Constraint Solving (2022. Autumn)

SAT

Propositional Satisfiability (SAT): Given a propositional formula φ, test whether there is an assignment to the variables that makes φ true.

e.g., $\varphi = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4)$

- Boolean variables: $x_1, x_2, \ldots$
- A literal is a Boolean variable x (positive literal) or its negation $\neg x$ (negative literal)
- A clause is a disjunction ($\vee$) of literals

  $x_2 \vee x_3$,

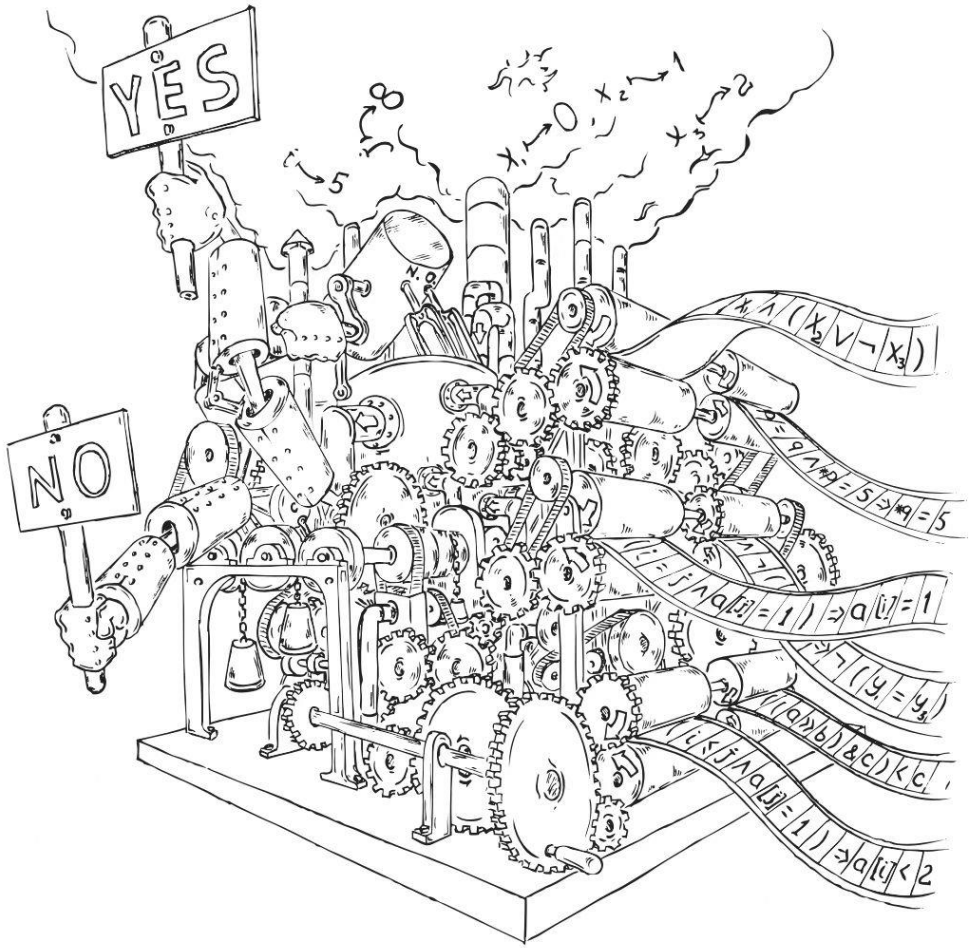
  $\neg x_1 \vee \neg x_3 \vee x_4$ $(\{\neg x_1, \neg x_3, x_4\})$

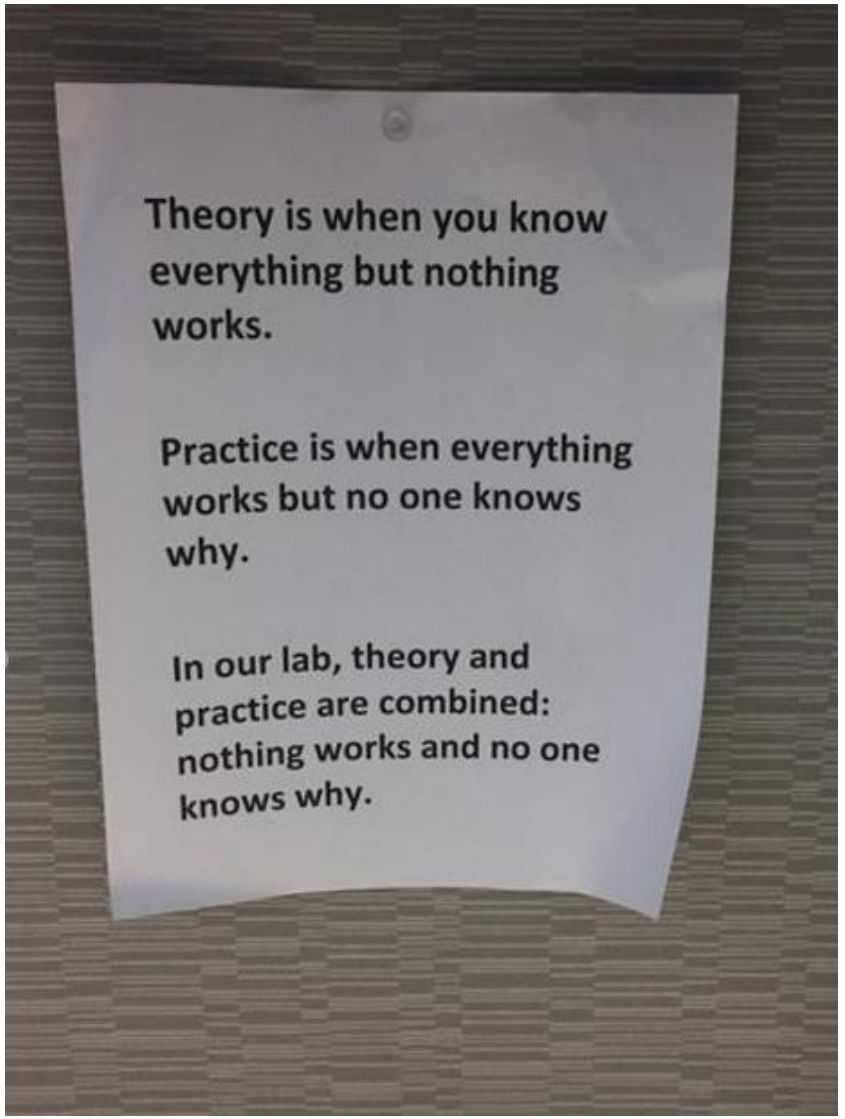
- A Conjunctive Normal Form(CNF) formula is a conjunction ($\wedge$) of clauses.

e.g., $\varphi = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4)$

Every propositional formulas can be converted into CNF efficiently.

Solvers 求解器


YES
NO


Theory is when you know everything but nothing works.
Practice is when everything works but no one knows why.
In our lab, theory and practice are combined: nothing works and no one knows why.

SAT solvers

- Using SAT solvers
  - To find a certain structure
  - To prove something

Constraints
Scheduling,
Resource allocation,
Logistics,
Hardware design,
Software Engineering,
…

Reasoning
Theorem proving,
System verification,
Knowledge representation,
Decision procedure,
Agents,
…

ty

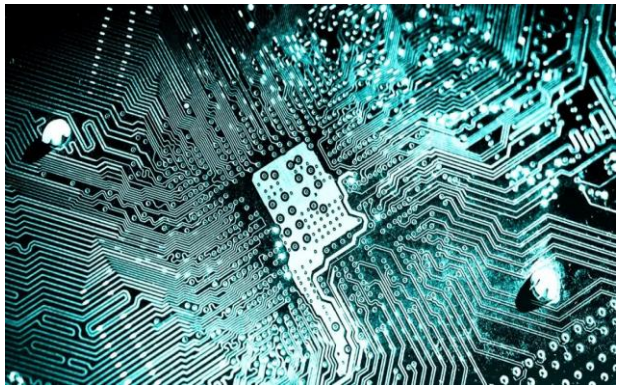
Finding a solution

Prove that
$x \geq 1, y \geq 1 \Rightarrow x + y \geq 2$

Is
$x \geq 1, y \geq 1, not(x + y \geq 2)$
UNSAT?

Checking validity

SAT revolution



EDA

original C code

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

        ⇓

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

optimized C code

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

        ⇑

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

How to check that these two versions are equivalent?

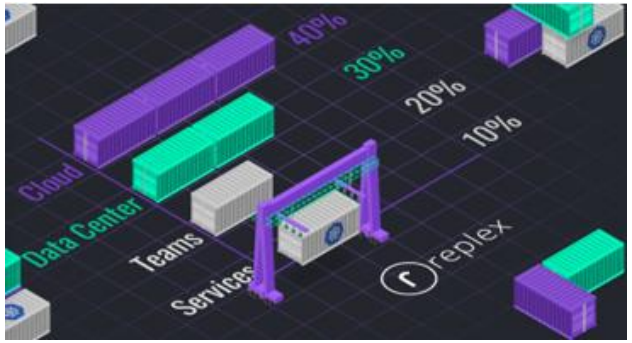
Program Analysis

Planning

cryptography

$x^m + y^m = z^m \pmod p$ *vdW(6) = 1132*
Schur's Theorem *Ramsey Theory*
Pythagorean Tuples Conjecture
3n+1 Conjecture?

Math



Resource Allocation

Using SAT Solvers

MiniSat: A open-source SAT solver widely used in industries.

Input file: DIMACS format.

```
c example
p cnf 4 4
1 -4 -3 0
1 4 0
-1 0
-4 3 0
```

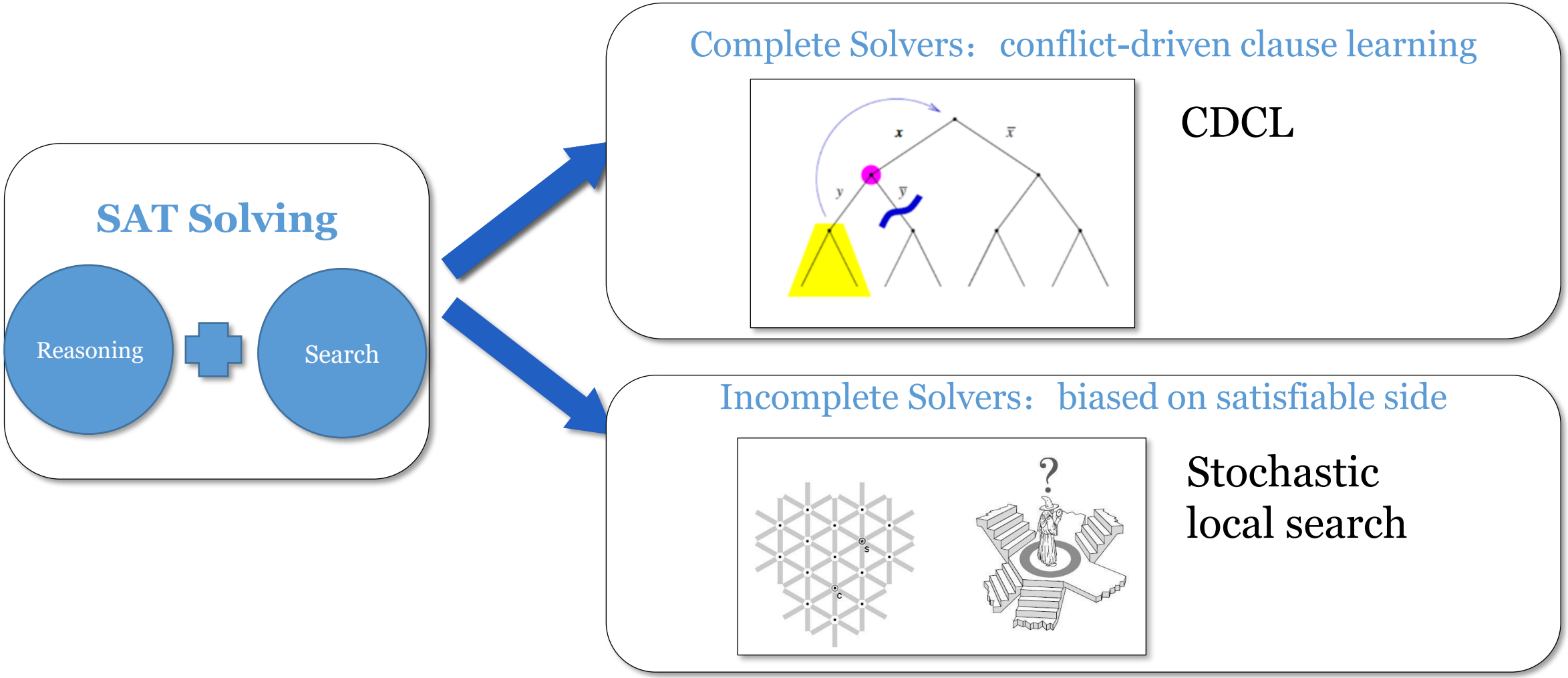
lines starting ”c” are comments and are ignored by the SAT solver.
a line starting with ”p cnf” is the problem definition line containing the number of variables and clauses.
the rest of the lines represent clauses, literals are integers (starting with variable 1), clauses are terminated by a zero.

Output format

c comments, usually stastitics about the solving
s SATISFIABLE
v 1 2 -3 -4 5 -6 -7 8 9 10
v -11 12 13 -14 15 0

the solution line (starting with ”s”) can contain SATISFIABLE, UNSATISFIABLE and UNKNOWN.
For SATISFIABLE case, the truth values of variables are printed in lines starting with ”v”, the last value is followed by a ”0”

SAT Solving Basis



SAT Solving Basis – Resolution 归结

$$\frac{C \vee x \qquad \bar{x} \vee D}{C \vee D} \ [\text{Res}]$$

- **Resolution.** If two clauses A and B have exactly one pair of complementary literals $a \in A$ and $\neg a \in B$, then the clause $A \cup B \backslash \{a, \neg a\}$ is called the resolvent of A and B (by a) and denoted by $R(A, B)$.

SAT Solving Basis - Resolution

Variable elimination by resolution

- Given a formula F and a literal a, the formula denoted $DP_a(F)$ is constructed from F
  - by adding all resolvents by a
  - and then removing all clauses that contain a or $\neg a$

Example. $F = (x \vee e) \wedge (y \vee e) \wedge (\bar{x} \vee z \vee \bar{e}) \wedge (y \vee \bar{e}) \wedge (y \vee z)$

Eliminating variable e by resolution:

- first add all resolvents upon e.

  $\{(x \vee e), (y \vee e)\}$ with $\{(\bar{x} \vee z \vee \bar{e}), (y \vee \bar{e})\}$ → 4 resolvents

  $$F \wedge (x \vee \bar{x} \vee z) \wedge (x \vee y) \wedge (y \vee \bar{x} \vee z) \wedge (y)$$

- remove all clauses that contain e to obtain

  $$(y \vee z) \wedge (x \vee \bar{x} \vee z) \wedge (x \vee y)\,(y \vee \bar{x} \vee z) \wedge (y)$$

SAT Solving Basis - Unit Propagation 单元传播

- Unit Clause: A Clause that all literals are falsified except one unassigned literal.
- Unit Propagation (UP): the unassigned literal in unit clause can only be assigned to single value to satisfy the clause.

Example:

	x_1	T	x_1	sat
	x_2		$\bar{x}_1 \vee \bar{x}_2$	
$Vars$:	x_3	$clauses$:	$\bar{x}_1 \vee \bar{x}_3$	
	x_4		$x_2 \vee \bar{x}_3 \vee \bar{x}_4$	
	x_5		$\bar{x}_1 \vee x_4 \vee x_5$	
	x_6		$x_2 \vee x_4 \vee x_6$	

SAT Solving Basis - Unit Propagation

- Unit Clause: A Clause that all literals are falsified except one unassigned literal.
- Unit Propagation (UP): the unassigned literal in unit clause can only be assigned to single value to satisfy the clause.

Example:

$Vars$:		$clauses$:	
x_1	T	x_1	sat
x_2		$\bar{x}_1 \vee \bar{x}_2$	
x_3		$\bar{x}_1 \vee \bar{x}_3$	
x_4		$x_2 \vee \bar{x}_3 \vee \bar{x}_4$	
x_5		$\bar{x}_1 \vee x_4 \vee x_5$	
x_6		$x_2 \vee x_4 \vee x_6$	

SAT Solving Basis – DP Algorithm

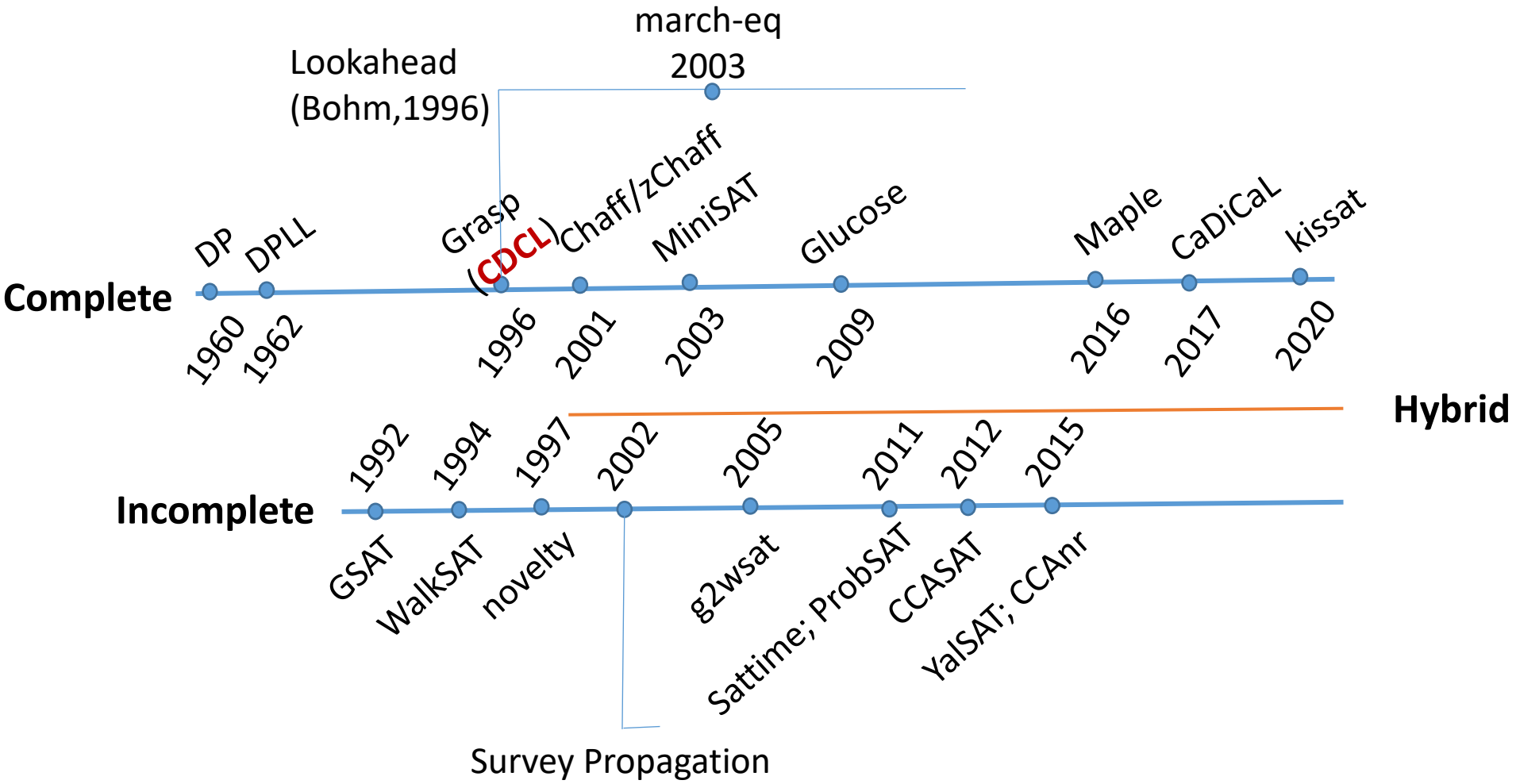
Davis-Putnam Algorithm [1960] % could find record in 1959

- Rule 1: Unit propagation
- Rule 2: Pure literal elimination
- Rule 3: Resolution at one variable

Apply deduction rules (giving priority to rules 1 and 2) until no further rule is applicable

Solver = Algorithmic framework + heuristics. [just a quick thinking, don't quote me…]

SAT Solving – Quest for Efficient SAT solving



A brief history of SAT solving

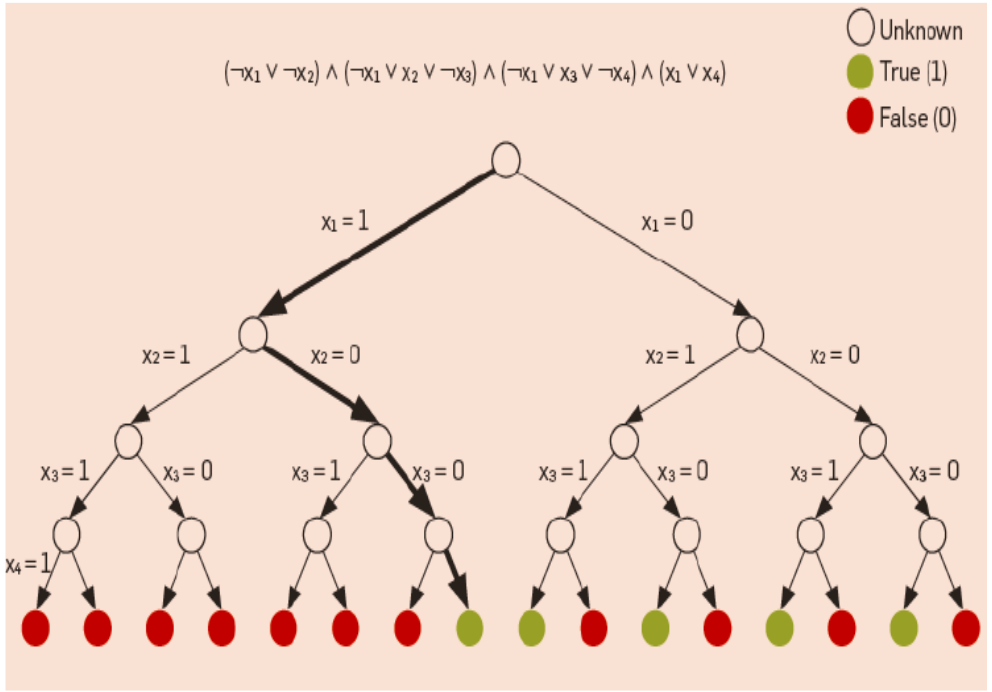
- 1960-1990
  - DP, DPLL
  - NP completeness, Complexity, Tractable subclass,…
  - Resolution: Stålmarck's Method (1989)
- 1990-2010
  - Local Search (1992): GSAT (1992), WalkSAT (1994)
  - Conflict Driven Clause Learning (1996): GRASP(1999), Chaff(2000), MiniSAT (2003), Glucose (2009)
  - Phase transition, Survey Propagation
  - Portfolio: SATzilla (2007)
- 2010~today
  - Modern local search
  - Advanced clause management and simplification
  - Hybridizing CDCL and local search (winners in 2020 and 2021 are of this type)
  - Parallel solving: cube and conquer

Millions of variables solved in 1 hour

DPLL Algorithm

Davis-Putnam-Logemann-Loveland (DPLL, 1962)

- Chronological backtracking + UP + Decision heuristics



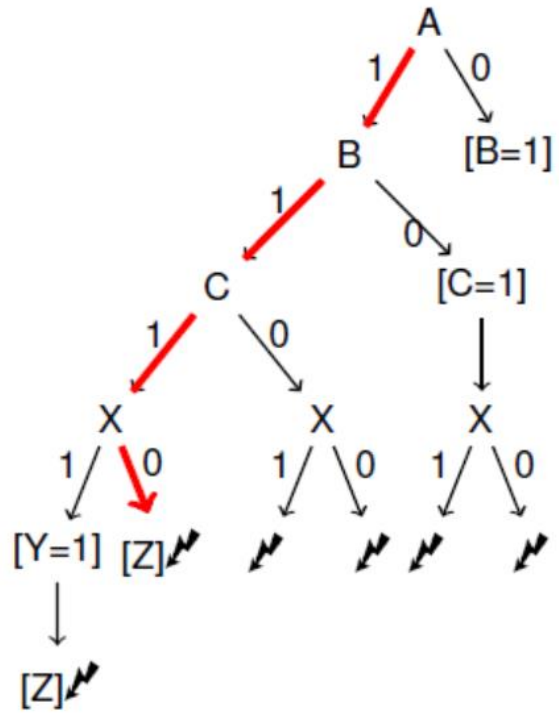
conditioning Δ on literal L: $\Delta|L = \{\alpha - \{\neg L\} \mid \alpha \in \Delta,\ L \notin \alpha\}.$

DPLL Algorithm

Davis-Putnam-Logemann-Loveland (DPLL, 1962)

- Chronological backtracking + UP + Decision heuristics

$\Delta =$
1. $\{A, B\}$
2. $\{B, C\}$
3. $\{\neg A, \neg X, Y\}$
4. $\{\neg A, X, Z\}$
5. $\{\neg A, \neg Y, Z\}$
6. $\{\neg A, X, \neg Z\}$
7. $\{\neg A, \neg Y, \neg Z\}$



Decision level

[UP] Decide [UP] Decide [UP] ….

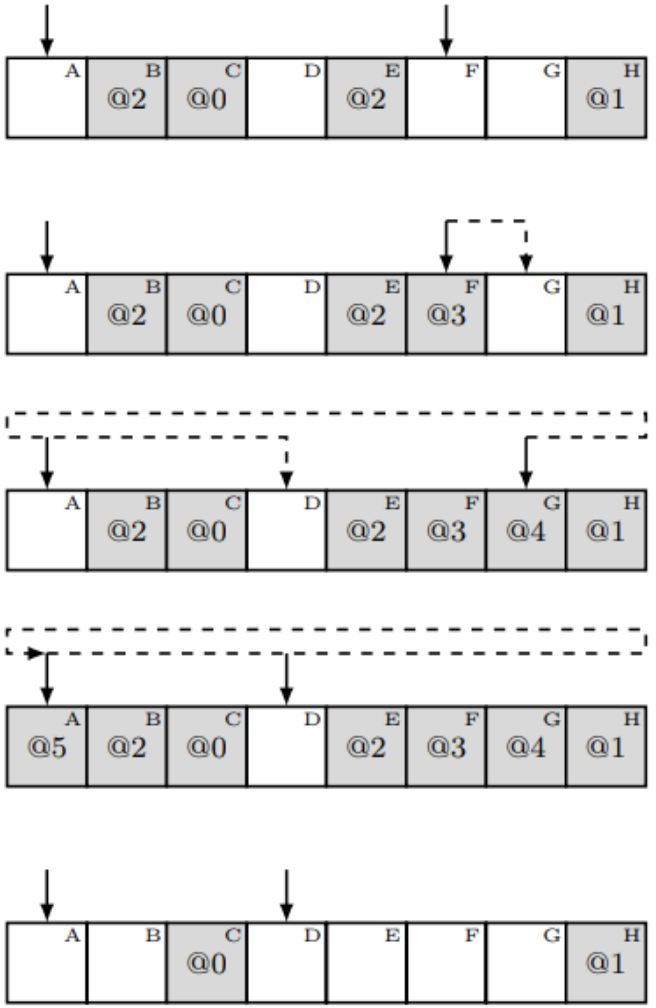
Level 0 Level 1

Lazy data structure for UP

[ZhangStickel' 00] [MoskewiczMadiganZhaoZhangMalik' 01]

Efficient UP: 2 watched literals

- In each non-satisfied clause "watch" two non-false literals
- For each literal remember all the clauses where it is watched



VSIDS Branching heuristics

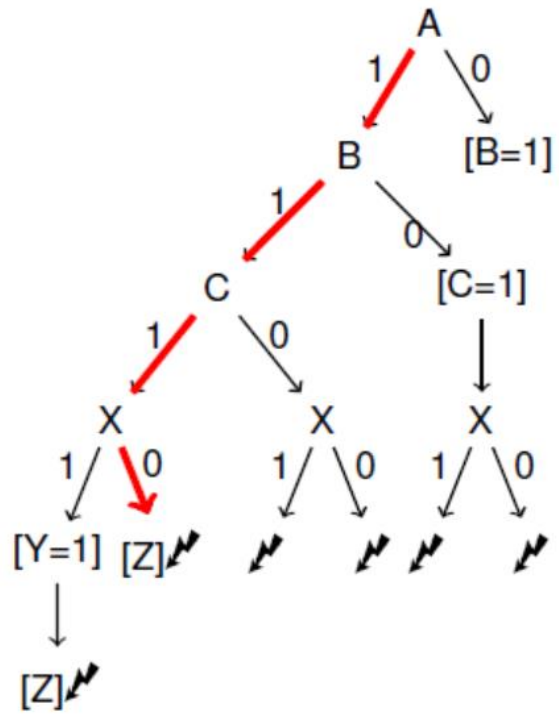
- Branching heuristics are used for deciding which variable to use when branching.
  - Solvers prefer the variable which may cause conflicts faster.
- Variable State Independent Decaying Sum (VSIDS) [MoskewiczMadiganZhaoZhangMalik'01]
  - **Compute score for each variable, select variable with highest score**
  - Initial variable score is number of literal occurrences.
  - For a new conflict clause c: score of all variables in c is incremented.
  - Periodically, divide all scores by a constant. // forgetting previous effects

VSIDS Branching heuristics

- Most popular: the exponential variant in MiniSAT (EVSIDS)
  - The scores of some variables are *bumped* with *inc*, and *inc* decays after each conflict.
  - Initialize *score* to 0, the bump score *inc* default to 1.
  - *inc* multiply 1/*decay* after each conflict, *decay* initialized to 0.8, increased by 0.01 every [5k] conflicts, the maximum of *decay* is 0.95.
  - The score of variables in conflict clause *c* are bumped with *inc*.

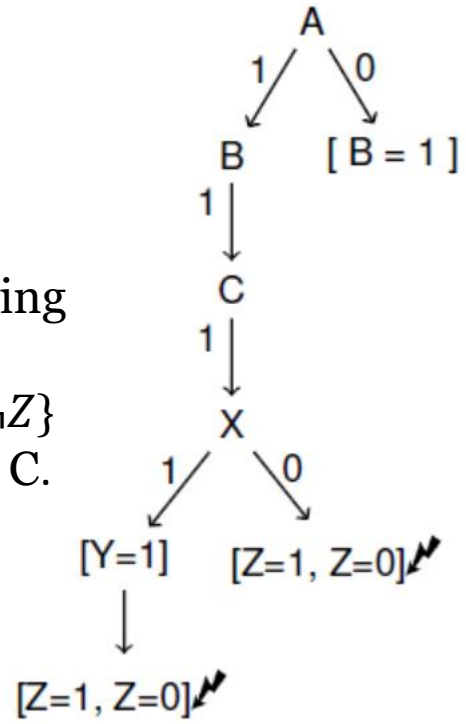
Backjumping

$$\Delta = \begin{array}{l} 1.\ \{A, B\} \\ 2.\ \{B, C\} \\ 3.\ \{\neg A, \neg X, Y\} \\ 4.\ \{\neg A, X, Z\} \\ 5.\ \{\neg A, \neg Y, Z\} \\ 6.\ \{\neg A, X, \neg Z\} \\ 7.\ \{\neg A, \neg Y, \neg Z\} \end{array}$$



Chronological Backtracking

The first two conflicting clauses $\{\neg A, \neg X, Y\}, \{\neg A, X, \neg Z\}$ do not involve B and C.



Non-Chronological Backtracking

CDCL: Conflict Driven Clause Learning

- A demonstration on clause learning

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

  - Assume decisions $c = 0$ and $f = 0$
  - Assign $a = 0$ and imply assignments
  - A conflict is reached: $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $(a = 0) \wedge (c = 0) \wedge (f = 0) \Rightarrow (\varphi = 0)$
  - $(\varphi = 1) \Rightarrow (a = 1) \vee (c = 1) \vee (f = 1)$

  - Learn new clause $(a \vee c \vee f)$

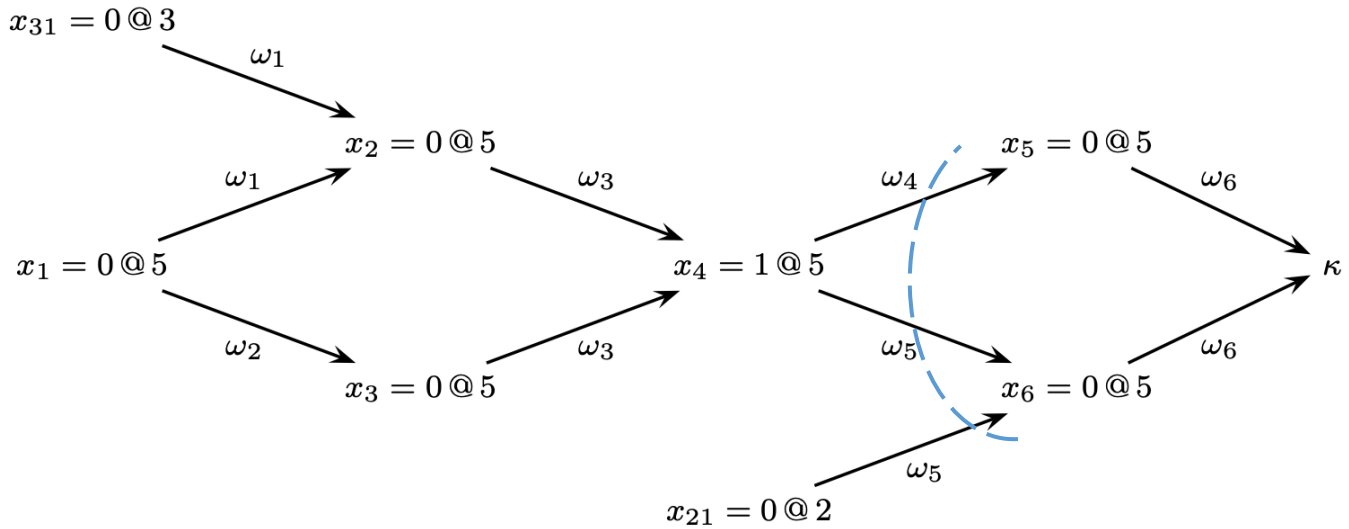
CDCL – Implication Graph

[MarqueSilvaSakallah' 96]

Implication Graph describes the decision and reasoning path.

- Vertex: (decision variable = value @decision level)
- Edge : unit clause used in UP (Reason Clause) .
- Conflict: all literals are falsified (under the current assignment).

$$\begin{aligned}\varphi_1 &= \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6 \\ &= (x_1 \vee x_{31} \vee \neg x_2) \wedge (x_1 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge \\ &\quad (\neg x_4 \vee \neg x_5) \wedge (x_{21} \vee \neg x_4 \vee \neg x_6) \wedge (x_5 \vee x_6)\end{aligned}$$



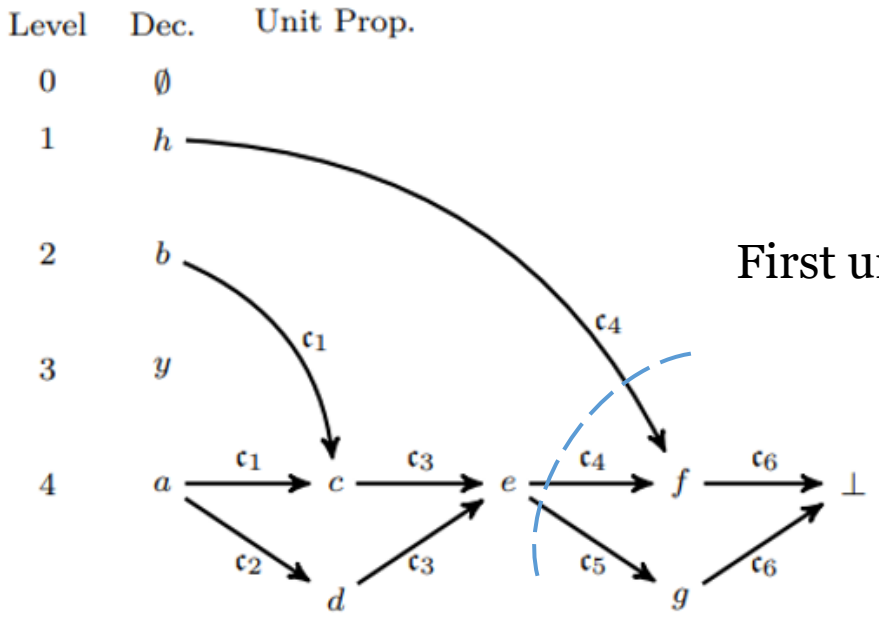
CDCL – Implication Graph

[MarqueSilvaSakallah' 96]

Implication Graph describes the decision and reasoning path.

- Vertex: (decision variable = value @decision level)
- Edge : unit clause used in UP (Reason Clause) .
- Conflict: all literals are falsified (under the current assignment).

$$\mathcal{F}_2 = \mathbf{c}_1 \wedge \mathbf{c}_2 \wedge \mathbf{c}_3 \wedge \mathbf{c}_4 \wedge \mathbf{c}_5 \wedge \mathbf{c}_6$$
$$= (\bar{a} \vee \bar{b} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{c} \vee \bar{d} \vee e) \wedge (\bar{h} \vee \bar{e} \vee f) \wedge (\bar{e} \vee g) \wedge (\bar{f} \vee \bar{g})$$

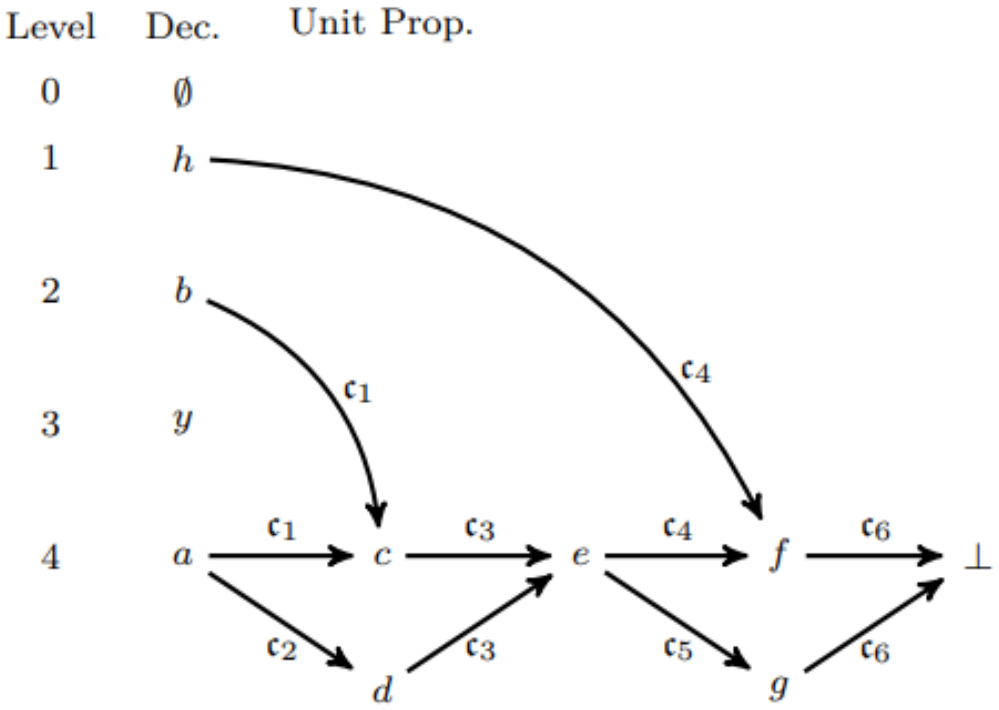


First unique implication point (UIP)

CDCL – Implication Graph

Implication Graph

$$\mathcal{F}_2 = \mathfrak{c}_1 \wedge \mathfrak{c}_2 \wedge \mathfrak{c}_3 \wedge \mathfrak{c}_4 \wedge \mathfrak{c}_5 \wedge \mathfrak{c}_6$$
$$= (\bar{a} \vee \bar{b} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{c} \vee \bar{d} \vee e) \wedge (\bar{h} \vee \bar{e} \vee f) \wedge (\bar{e} \vee g) \wedge (\bar{f} \vee \bar{g})$$



(a) Implication graph

$x \in \mathbb{V} \cup \{\bot\}$	$\nu(\cdot)$	$\delta(\cdot)$	$\alpha(\cdot)$
h	1	1	$\mathfrak{d}$
b	1	2	$\mathfrak{d}$
y	1	3	$\mathfrak{d}$
a	1	4	$\mathfrak{d}$
c	1	4	$\mathfrak{c}_1$
d	1	4	$\mathfrak{c}_2$
e	1	4	$\mathfrak{c}_3$
f	1	4	$\mathfrak{c}_4$
g	1	4	$\mathfrak{c}_5$
$\bot$	–	–	$\mathfrak{c}_6$

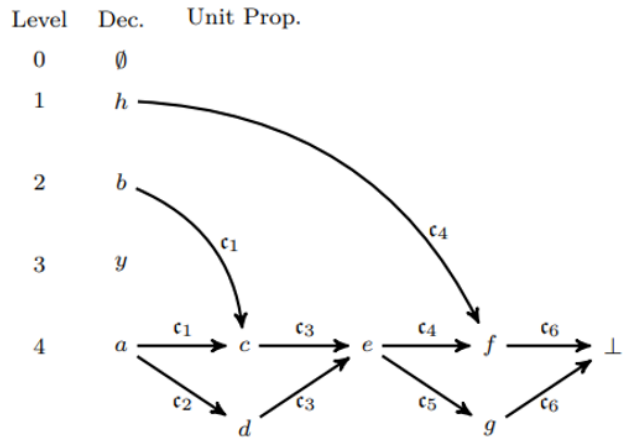
(b) State of variables

For variable x:

$v(x)$: *the value*
$\delta(x)$: *the decision level*
$\alpha(x)$: *the reason clause*

CDCL – Conflict Analysis

$$\mathcal{F}_2 = \mathfrak{c}_1 \wedge \mathfrak{c}_2 \wedge \mathfrak{c}_3 \wedge \mathfrak{c}_4 \wedge \mathfrak{c}_5 \wedge \mathfrak{c}_6$$
$$= (\bar{a} \vee \bar{b} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{c} \vee \bar{d} \vee e) \wedge (\bar{h} \vee \bar{e} \vee f) \wedge (\bar{e} \vee g) \wedge (\bar{f} \vee \bar{g})$$



$x \in \mathbb{V} \cup \{\bot\}$	$\nu(\cdot)$	$\delta(\cdot)$	$\alpha(\cdot)$
h	1	1	$\mathfrak{d}$
b	1	2	$\mathfrak{d}$
y	1	3	$\mathfrak{d}$
a	1	4	$\mathfrak{d}$
c	1	4	$\mathfrak{c}_1$
d	1	4	$\mathfrak{c}_2$
e	1	4	$\mathfrak{c}_3$
f	1	4	$\mathfrak{c}_4$
g	1	4	$\mathfrak{c}_5$
$\bot$	–	–	$\mathfrak{c}_6$

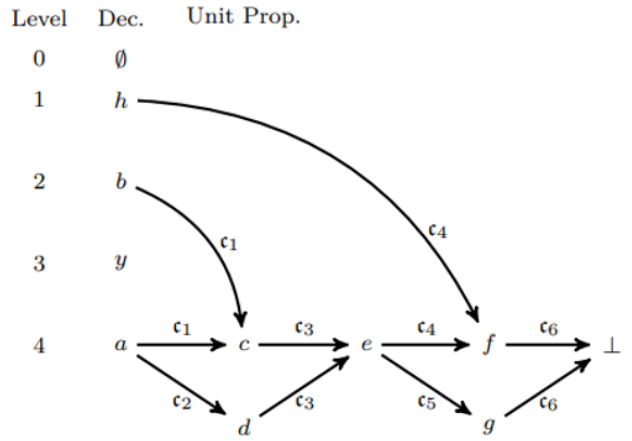
Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\bot$	$\mathfrak{c}_6$	$\emptyset$	$\{f, g\}$
1	$[f, g]$	f	$\mathfrak{c}_4$	$\{\bar{h}\}$	$\{e\}$
2	$[g, e]$	g	$\mathfrak{c}_5$	$\{\bar{h}\}$	$\emptyset$
3	$[e]$	e	$\mathfrak{c}_3$	$\{\bar{h}, \bar{e}\}$	$\emptyset$
6	[]	–	–	$\{\bar{h}, \bar{e}\}$	–

Variables are analyzed in an FIFO fashion, starting from the conflict.

In each step, for the reason clause, all literals assigned at decision levels smaller than the current one are added to (i.e. recorded in) the clause being learned, while the others are added to the queue for analyse.

CDCL – Conflict Analysis

$$\mathcal{F}_2 = \mathfrak{c}_1 \wedge \mathfrak{c}_2 \wedge \mathfrak{c}_3 \wedge \mathfrak{c}_4 \wedge \mathfrak{c}_5 \wedge \mathfrak{c}_6$$
$$= (\bar{a} \vee \bar{b} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{c} \vee \bar{d} \vee e) \wedge (\bar{h} \vee \bar{e} \vee f) \wedge (\bar{e} \vee g) \wedge (\bar{f} \vee \bar{g})$$



$x \in \mathbb{V} \cup \{\bot\}$	$\nu(\cdot)$	$\delta(\cdot)$	$\alpha(\cdot)$
h	1	1	$\mathfrak{d}$
b	1	2	$\mathfrak{d}$
y	1	3	$\mathfrak{d}$
a	1	4	$\mathfrak{d}$
c	1	4	$\mathfrak{c}_1$
d	1	4	$\mathfrak{c}_2$
e	1	4	$\mathfrak{c}_3$
f	1	4	$\mathfrak{c}_4$
g	1	4	$\mathfrak{c}_5$
$\bot$	–	–	$\mathfrak{c}_6$

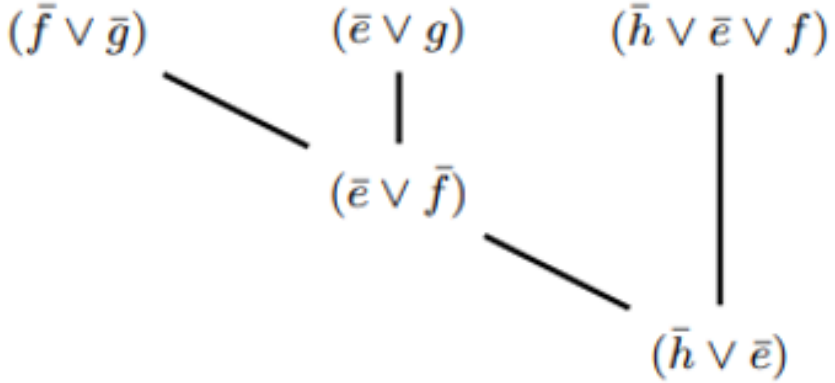
Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\bot$	$\mathfrak{c}_6$	$\emptyset$	$\{f, g\}$
1	$[f, g]$	f	$\mathfrak{c}_4$	$\{\bar{h}\}$	$\{e\}$
2	$[g, e]$	g	$\mathfrak{c}_5$	$\{\bar{h}\}$	$\emptyset$
3	$[e]$	e	$\mathfrak{c}_3$	$\{\bar{h}, \bar{e}\}$	$\emptyset$
6	$[]$	–	–	$\{\bar{h}, \bar{e}\}$	–

unique implication point (UIP)
in step 3, there exists only one variable to trace, e, it is a UIP.
a UIP is a dominator of the decision variable with respect to the conflict node $\bot$.

CDCL – Conflict Analysis

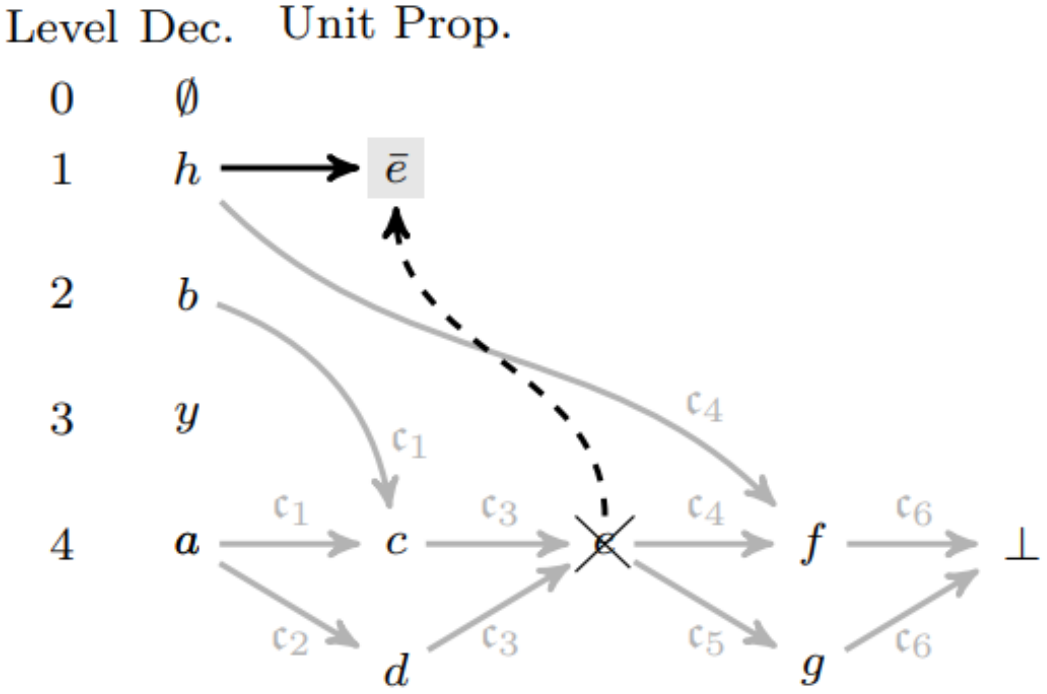
$$\mathcal{F}_2 = \mathfrak{c}_1 \wedge \mathfrak{c}_2 \wedge \mathfrak{c}_3 \wedge \mathfrak{c}_4 \wedge \mathfrak{c}_5 \wedge \mathfrak{c}_6$$
$$= (\bar{a} \vee \bar{b} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{c} \vee \bar{d} \vee e) \wedge (\bar{h} \vee \bar{e} \vee f) \wedge (\bar{e} \vee g) \wedge (\bar{f} \vee \bar{g})$$

Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\perp$	$\mathfrak{c}_6$	$\emptyset$	$\{f, g\}$
1	$[f, g]$	f	$\mathfrak{c}_4$	$\{\bar{h}\}$	$\{e\}$
2	$[g, e]$	g	$\mathfrak{c}_5$	$\{\bar{h}\}$	$\emptyset$
3	$[e]$	e	$\mathfrak{c}_3$	$\{\bar{h}, \bar{e}\}$	$\emptyset$
6	[]	–	–	$\{\bar{h}, \bar{e}\}$	–

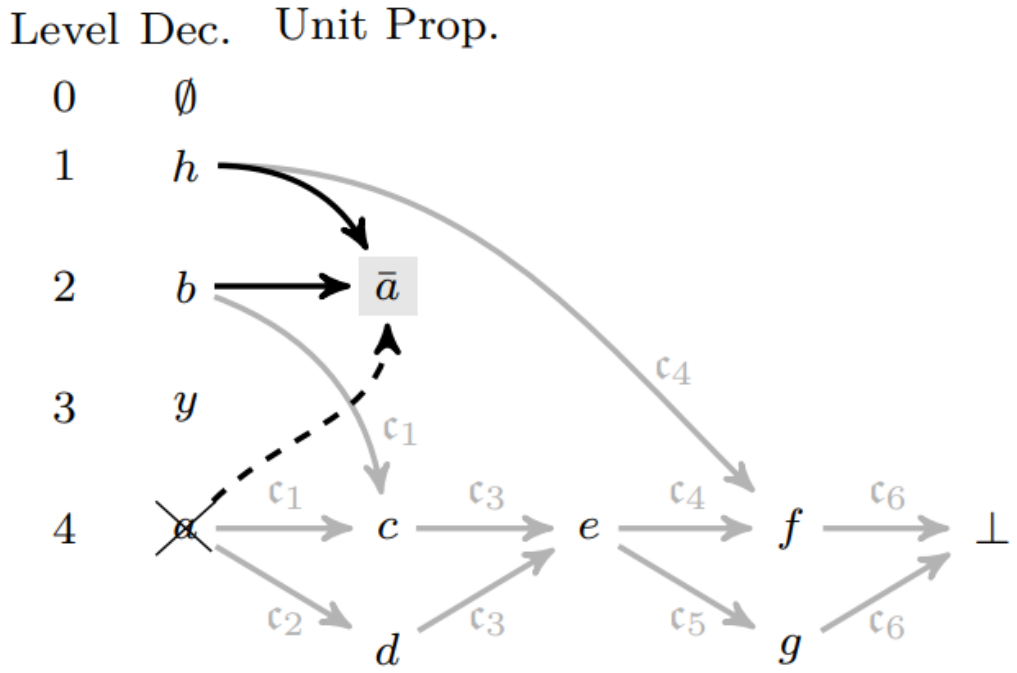


CDCL – Conflict Analysis

$$\mathcal{F}_2 = \mathbf{c}_1 \wedge \mathbf{c}_2 \wedge \mathbf{c}_3 \wedge \mathbf{c}_4 \wedge \mathbf{c}_5 \wedge \mathbf{c}_6$$
$$= (\bar{a} \vee \bar{b} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{c} \vee \bar{d} \vee e) \wedge (\bar{h} \vee \bar{e} \vee f) \wedge (\bar{e} \vee g) \wedge (\bar{f} \vee \bar{g}) \wedge (\overline{h} \vee \overline{e})$$



NBC with first UIP learnt clause $\overline{h} \vee \overline{e}$

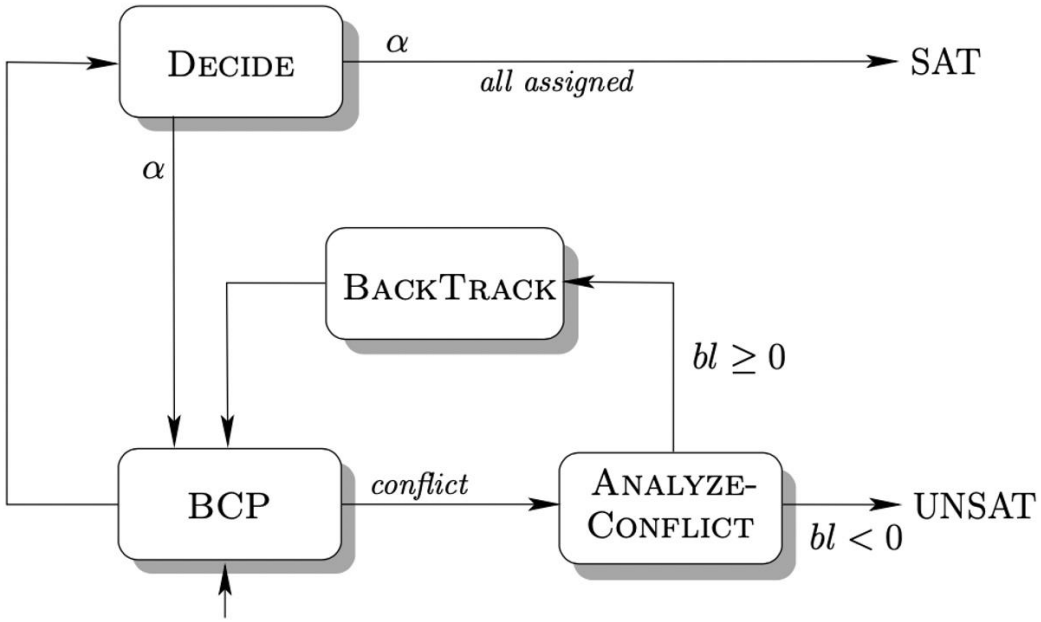


NBC with learnt clause $\overline{h} \vee \overline{b} \vee \overline{a}$

回退到学习子句的第二深（就是除了当前层之外，最深的一层）

CDCL Framework

- Analyze-Conflict : non-chronological backtracking + clause learning + vivification
- Decide : Branching strategy and phasing strategy

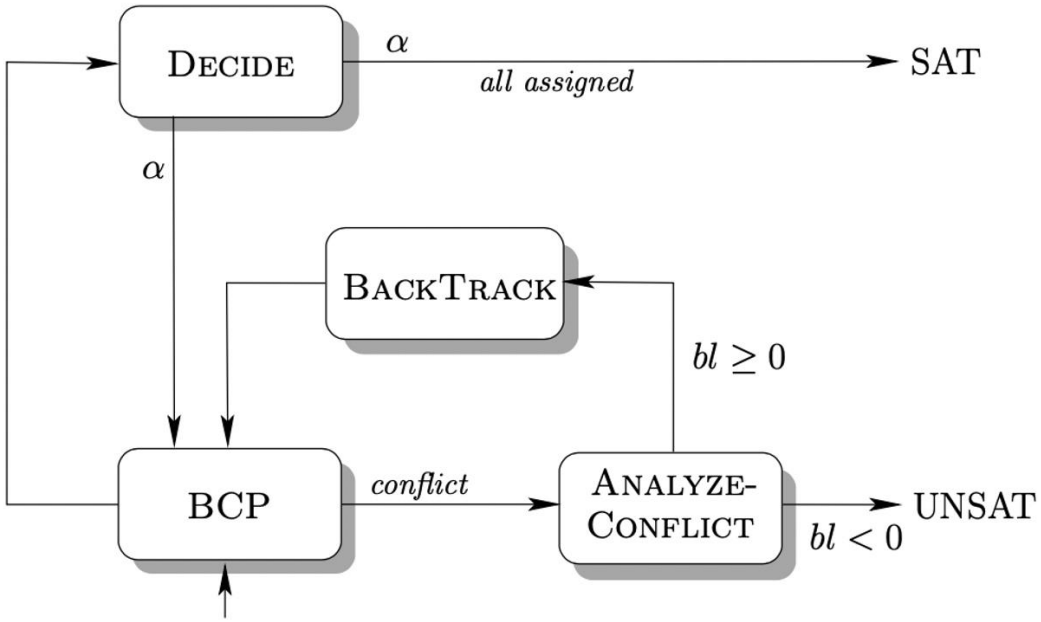


Algorithm 1: Typical CDCL algorithm: CDCL(F, α)

```
1  dl ← 0;                    //decision level
2  if UnitPropagation(F,α)==CONFLICT then return UNSAT
3  while ∃ unassigned variables do
       /* PickBranchVar picks a variable to assign and
          picks the respective value                    */
4      (x, v) ← PickBranchVar(F, α);
5      dl ← dl + 1;
6      α ← α ∪ {(x, v)};
7      if UnitPropagation(F,α)==CONFLICT then
8          bl ← ConflictAnalysis(F, α);
9          if bl < 0 then
10             return UNSAT;
11         else
12             BackTrack(F, α, bl);
13             dl ← bl;
14 return SAT;
```

CDCL Algorithm – CDCL

- Analyze-Conflict : non-chronological backtracking + clause learning + vivification
- Decide : Branching strategy and phasing strategy



Algorithm 1: Typical CDCL algorithm: CDCL(F, α)

```
1  dl ← 0;                    //decision level
2  if UnitPropagation(F,α)==CONFLICT then return UNSAT
3  while ∃ unassigned variables do
       /* PickBranchVar picks a variable to assign and
          picks the respective value
4      (x, v) ← PickBranchVar(F, α);
5      dl ← dl + 1;
6      α ← α ∪ {(x, v)};
7      if UnitPropagation(F,α)==CONFLICT then
8          bl ← ConflictAnalysis(F, α);
9          if bl < 0 then
10             return UNSAT;
11         else
12             BackTrack(F, α, bl);
13             dl ← bl;
14 return SAT;
```

- Clause learning
- Clause management
- Lazy data structures
- Restarting
- Branching
- Phasing
- Mode Switching
- …

CDCL Heuristics – Branching heuristics

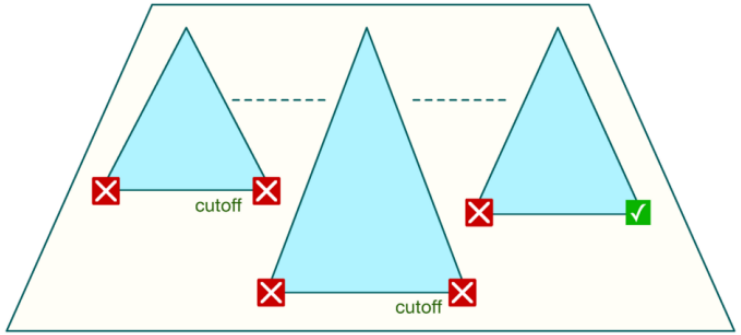
- Static branching heuristic: e.g. Ordered BDDs
- Dynamic branching heuristic considering current partial assignment.
  - dynamic literal individual sum heuristic (DLIS)
- Dynamic branching heuristic considering learning clauses
  - Variants of DLIS
  - Variable state independent decaying sum(VSIDS) and its variants
    - Normalized VSIDS(NVSIDS) : exponential moving average
    - Exponential VSIDS(EVSIDS) : proposed by MiniSAT
    - Literal state independent decaying sum(LSIDS)
  - Variable move to front(VMTF) [Ryan Thesis 2004]
  - Average conflict-index decision score(ACIDS)
- Reinforcement learning based branching heuristic: multi-armed bandit(MAB)
  - Conflict history-based branching(CHB): $(1-\alpha)s + \alpha \cdot r, r = \frac{multiplier}{nConf - lastConf_v + 1}$
  - Learning rate based branching(LRB)
- Dynamic switching between multiple heuristics
  - Kissat-MAB switching between CHB and VSIDS by Upper Confidence Bound(UCB)

CDCL Heuristics – Learnt Clause Removal

- Reason why **Clause Database Reduction**:
  - Not all of them are helpful;
  - UP gets slower with memory consumption.
- Measurement criteria of clauses:
  - least recently used (LRU) heuristics: discard clauses not involved in recent conflict clause generation
  - **Literal Block Distance(LBD): number of distinct decision levels in learnt clauses**, proposed in glucose. [AudemardSimon,09IJCAI]
  - 3-tired clause Learned clause management : *core* are clauses with LBD≤ 3; *mid_tire* retain recently used clause with LBD up to 6; *local* saving other clauses. [Chanseok Oh, 15SAT]
- Reduction Method in 3-tier method:
  - *core* never be removed;
  - Periodically remove half *local* clauses based on score.
  - Periodically move some recently not used clauses in *mid_tire* to *local*.
  - move clauses encounter in *local* many times to *mid_tire*, and same from *mid_tire* to *core*.

CDCL Heuristics – Effective Restart

- Periodical traceback to 0 decision level.
  - clause learning and search restarts correspond to a proof system as powerful as general resolution, and stronger than DPLL proof system; practically effective.

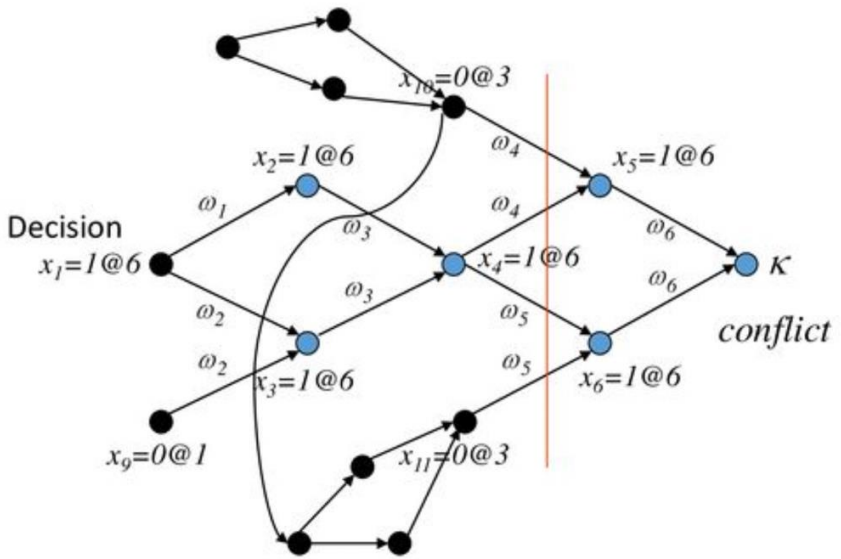


- Restart policies:
  - Luby series: 1 1 2 1 1 2 4 1 1 2 1 1 2 4 8 …
  - Glucose restart (rapid) : When average LBD of some current learnt clauses is great than the average LBD of all learnt clauses. [AudemardSimon,12CP]
- A conjecture: rapid restarts generally helps deriving a refutation proof, while remaining in the current branch increases the chance of reaching a model
  - interleave “stabilizing” mode (no restarts) and “focused” mode [Chanseok Oh,15SAT]

CDCL Heuristics – Clause Simplification

- Remove some literals which can be conducted by another literal in the clause.

- $reason(x_3) = \omega_4, reason(x_6) = \omega_5, \ldots$
- Local / General implication graph



Learnt Conflict clause: ($x_{10}, \neg x_4, x_{11}$)

$\overline{x_{10}} \rightarrow \overline{x_{11}}$

or, $x_{11} \rightarrow x_{10}$

Clause minimization: Drop x_{11}

Summary

Core information of this part:

- History of SAT solving
- Know key data structure and implementation of CDCL
  - Watching literals → UP
  - Implication graphs → clause learning
- Understand the core ideas of CDCL (know why)
- Know important engineering issues