软件分析与架构设计

布尔可满足性问题（SAT）

何冬杰

重庆大学

Boolean Formula

- **A logical expression composed of Boolean variables and logical operators, which evaluates to true or false under a given assignment**
  - Logical operators: NOT (¬), AND (∧), OR (∨), IMPLIES (→), …
  - E.g., $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi, \ldots$
- **Negation-Normal Form (NNF):**
  - A formula is in NNF iff
    - all negations are directly in front of variables, and
    - the only logical connectives are NOT (¬), AND (∧), and OR (∨).
  - A **literal** is a variable or its negation
  - IMPLY: $a \rightarrow b \equiv \neg a \vee b$
  - Convert to NNF by pushing negations inward:

$$\neg(P \wedge Q) \Leftrightarrow (\neg P \vee \neg Q)$$
$$\neg(P \vee Q) \Leftrightarrow (\neg P \wedge \neg Q)$$

(De Morgan's Laws)

  - Example: $\neg(p \rightarrow q) \vee \neg(r \wedge s) \quad \Leftrightarrow \neg(\neg p \vee q) \vee \neg(r \wedge s) \Leftrightarrow (p \wedge \neg q) \vee \neg r \vee \neg s$

Boolean Formula

❑Disjunctive Normal Form (DNF):

- A formula is in DNF iff: it is a disjunction of conjunctions of literals

$$\underbrace{(\ell_{11} \wedge \ell_{12} \wedge \ell_{13})}_{\text{conjunction 1}} \vee \underbrace{(\ell_{21} \wedge \ell_{22} \wedge \ell_{23})}_{\text{conjunction 2}} \vee \underbrace{(\ell_{31} \wedge \ell_{32} \wedge \ell_{33})}_{\text{conjunction 3}}$$

- Every formula in DNF is also in NNF
- A simple (but inefficient) way convert to DNF:
  - Make a truth table for the formula φ
  - Each row where φ is true corresponds to a conjunct
  - Example: $\neg(r \wedge s)$

r	s	$\neg(r \wedge s)$
0	0	1
0	1	1
1	0	1
1	1	0

$\xrightarrow{\text{convert to DNF}}$ $(\neg r \wedge \neg s) \vee (\neg r \wedge s) \vee (r \wedge \neg s)$

Boolean Formula

- **Conjunctive Normal Form (CNF):**
  - A formula is in CNF iff: it is a conjunction of disjunctions of literals

$$\underbrace{(\ell_{11} \vee \ell_{12} \vee \ell_{13})}_{\text{clause 1}} \wedge \underbrace{(\ell_{21} \vee \ell_{22} \vee \ell_{23})}_{\text{clause 2}} \wedge \underbrace{(\ell_{31} \vee \ell_{32} \vee \ell_{33})}_{\text{clause 3}}$$

  - Any formula can be converted to CNF
    - But the resulting equivalent CNF can be exponentially larger
  - Tseitin transformation: A method for converting an arbitrary Boolean formula into a CNF in linear size, by introducing fresh auxiliary variables
    - The resulting CNF is **equisatisfiable** with the original formula
    - Not logically equivalent, but satisfiable under the same conditions
    - Widely used in SAT solvers to avoid exponential blow-up

Tseitin transformation to CNF

- **General Idea:**
  - Introduce new variables to represent each non-atomic subformula, and add clauses enforcing the equivalence
- **Example:** $\boldsymbol{\varphi} = (\mathrm{p} \vee \mathrm{q}) \wedge \neg \mathrm{r}$
  - Step 1: Introduce Fresh Variables such that $\boldsymbol{\varphi}$ becomes $\mathrm{x}_1 \wedge \mathrm{x}_2$
    - Equivalences: $\mathrm{x}_1 \leftrightarrow (\mathrm{p} \vee \mathrm{q})$, and $\mathrm{x}_2 \leftrightarrow \neg \mathrm{r}$
  - Step 2: Encode each Equivalence in CNF
    - $\mathrm{x}_1 \leftrightarrow (p \vee q) \equiv \left(\mathrm{x}_1 \rightarrow (p \vee q)\right) \wedge \left((p \vee q) \rightarrow \mathrm{x}_1\right)$
      $\equiv (\neg x_1 \vee \mathrm{p} \vee \mathrm{q}) \wedge (\neg \mathrm{p} \vee x_1) \wedge (\neg \mathrm{q} \ \vee x_2)$
    - $\mathrm{x}_2 \leftrightarrow \neg \mathrm{r} \equiv (\mathrm{x}_2 \rightarrow \neg \mathrm{r}) \wedge (\neg \mathrm{r} \rightarrow \mathrm{x}_2) \equiv (\neg x_2 \vee \neg \mathrm{r}) \wedge (\neg \mathrm{r} \ \vee x_2)$
  - Step 3: add clauses and $\boldsymbol{\varphi}$ becomes：
    - $\mathrm{x}_1 \wedge x_2 \wedge (\neg x_1 \vee \mathrm{p} \vee \mathrm{q}) \wedge (\neg \mathrm{p} \vee x_1) \wedge (\neg \mathrm{q} \ \vee x_2) \wedge (\neg x_2 \vee \neg \mathrm{r}) \wedge (\neg \mathrm{r} \ \vee x_2)$

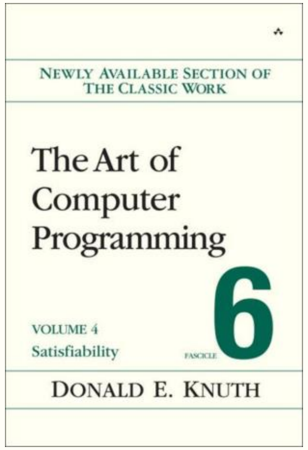
Problem of Boolean Satisfiability (SAT)

- **SAT problem:** Given a boolean formula φ, does there exist an assignment that satisfies φ?
- **Example：**
  - $C_1 = (\neg \mathrm{a} \vee \mathrm{b}) \wedge (\neg \mathrm{b} \vee \mathrm{c})$: satisfiable, {a=0, b = 0, c = 0}
  - $C_2 = (\neg \mathrm{a} \vee \mathrm{b}) \wedge (\neg \mathrm{b} \vee \mathrm{c}) \wedge \mathrm{a} \wedge \neg \mathrm{c}$: not satisfiable
    - $\equiv (\neg \mathrm{a} \vee \mathrm{c}) \wedge (\mathrm{a} \ \wedge \neg \mathrm{c}) \equiv (\neg \mathrm{a} \wedge \mathrm{a} \wedge \neg \mathrm{c}) \vee (\mathrm{c} \ \wedge \mathrm{a} \neg \mathrm{c}) \equiv \mathrm{false}$
- **SAT problem is NP-complete**
  - Proved through Cook-Levin theorem [1970s]
  - worst case is $O(2^n)$ (n variables has 2^n possible assignments)
- **Many problems reduce to SAT**
  - Formal verification, CAD, VLSI, Optimization,
  - AI, planning, automated deduction, …

SAT Solvers

- **Given a Boolean formula, SAT solvers**
  - Find an assignment if φ is satisfiable or answer no (or unknown)
- **Still active areas of research**

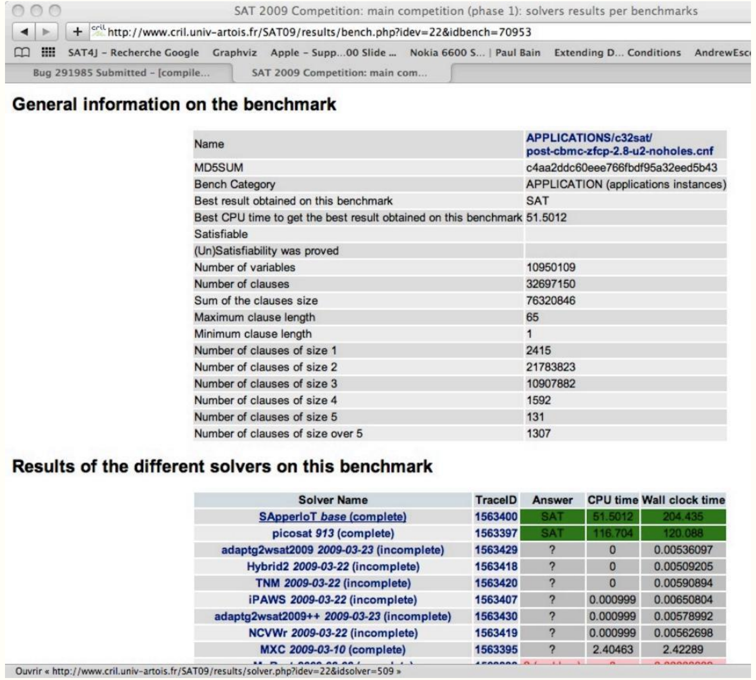
> "The story of satisfiability is the tale of a triumph of software engineering, blended with rich doses of beautiful mathematics."



- **How big is the search space of SAT solvers?**
  - How big is 10^{80}?
    - The number of protons in the observable universe
  - Let $10^3 \approx 2^{10}$, then $10^{80} \approx 2^{267}$
    - $\approx$ the theoretical search space for 267 Boolean variables
  - Modern solvers are often efficient
    - Handle $10^5 \sim 10^7$ Boolean variables

Interesting facts about SAT Solvers

Got hyperdrive (超光速引擎)
in 2001



SAT 2009 Competition: main competition (phase 1): solvers results per benchmarks

http://www.cril.univ-artois.fr/SAT09/results/bench.php?idev=22&idbench=70953

SAT4J – Recherche Google Graphviz Apple – Supp...00 Slide ... Nokia 6600 S... | Paul Bain Extending D... Conditions AndrewEsc

Bug 291985 Submitted – [compile... SAT 2009 Competition: main com...

General information on the benchmark

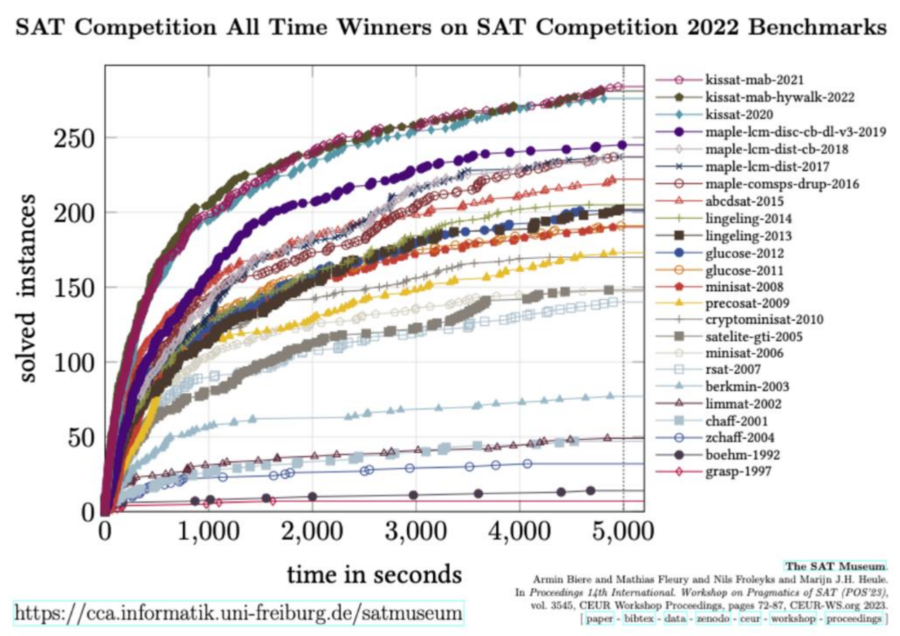
Name	APPLICATIONS/c32sat/ post-cbmc-zfcp-2.8-u2-noholes.cnf
MD5SUM	c4aa2ddc60eee766fbdf95a32eed5b43
Bench Category	APPLICATION (applications instances)
Best result obtained on this benchmark	SAT
Best CPU time to get the best result obtained on this benchmark	51.5012
Satisfiable	
(Un)Satisfiability was proved	
Number of variables	10950109
Number of clauses	32697150
Sum of the clauses size	76320846
Maximum clause length	65
Minimum clause length	1
Number of clauses of size 1	2415
Number of clauses of size 2	21783823
Number of clauses of size 3	10907882
Number of clauses of size 4	1592
Number of clauses of size 5	131
Number of clauses of size over 5	1307

Results of the different solvers on this benchmark

Solver Name	TraceID	Answer	CPU time	Wall clock time
SApperloT *base* (complete)	1563400	SAT	51.5012	204.435
picosat *913* (complete)	1563397	SAT	116.704	120.088
adaptg2wsat2009 *2009-03-23* (incomplete)	1563429	?	0	0.00536097
Hybrid2 *2009-03-22* (incomplete)	1563418	?	0	0.00509205
TNM *2009-03-22* (incomplete)	1563420	?	0	0.00590894
iPAWS *2009-03-22* (incomplete)	1563407	?	0.000999	0.00650804
adaptg2wsat2009++ *2009-03-23* (incomplete)	1563430	?	0.000999	0.00578992
NCVWr *2009-03-22* (incomplete)	1563419	?	0.000999	0.00562698
MXC *2009-03-10* (complete)	1563395	?	2.40463	2.42289

Ouvrir « http://www.cril.univ-artois.fr/SAT09/results/solver.php?idev=22&idsolver=509 »

Can handle 10,950,109 Boolean variables in 2009



Keep improving in Recent SAT Competition

DPLL: Efficient SAT solving in practice

- **Davis-Putnam-Logemann-Loveland (DPLL)，1961**
  - Handle formulas of $10^2 \sim 10^3$ Boolean variables
  - Input representation: A CNF formula φ
    - φ is represented by a set of clauses, empty set represents a true formula
    - a clause is represented by a set of literals, empty set represents a false clause
      - Assumption: No clause contains both a variable and its negation
    - a variable is represented by a positive integer
    - the negation of a variable is represented by the arithmetic negation of its number
  - Example: $(x1 \vee x2) \wedge (\neg x1 \vee \neg x2)$
    - Represented by {{1, 2}, {-1, -2}}
  - DPLL is still exponential in theory, but on many problems is much faster than trying all assignments
  - **Core Idea** of DPLL Solver:
    - **Logic part**: apply resolution-based eliminations, reducing variables
    - **Search part**: if no forced move exists, pick a variable x and try $x = \text{true or} = \text{false}$
      - $\text{F is SAT} \iff \text{F}[\text{x} \coloneqq \text{true}] \text{ is SAT} \vee \text{F}[\text{x} \coloneqq \text{false}] \text{ is SAT}$

Resolution-based Elimination

❑Unit Propagation

- ➢Unit Clause: Clause with exactly one literal
  - o Lemma 1: $(\mathrm{x}) \wedge \mathrm{F}$ is satisfiable $\Leftrightarrow \mathrm{F}[\mathrm{x} \coloneqq \mathrm{true}]$ is satisfiable
  - o Lemma 2: $(\neg\mathrm{x}) \wedge \mathrm{F}$ is satisfiable $\Leftrightarrow \mathrm{F}[\mathrm{x} \coloneqq \mathrm{false}]$ is satisfiable
- ➢Transformation: let x be a unit clause ($\neg x$ handles similarly)

Clause type	(x)	$(\mathrm{A} \vee \mathrm{x})$	$(\mathrm{B} \vee \neg\mathrm{x})$	Unrelated
resolution	Eliminated	Deleted	Becomes B	Kept
Unit Propagation	Assignment fixed	Clause satisfied	Literal removed	Kept

- ➢Algorithm: given a formula φ in CNF,
  - o If a clause in φ has exactly one literal, then assign it true, and simplify,
  - o Clauses are shortened, new unit clauses may appear
  - o Repeat until there are no more unit clauses
- ➢Example: $((\mathrm{x1} \vee \mathrm{x2}) \wedge (\neg\mathrm{x1} \vee \neg\mathrm{x2}) \wedge (\mathrm{x1}))$
  - o $\Rightarrow (\mathrm{true} \vee \mathrm{x2}) \wedge (\mathrm{false} \vee \neg\mathrm{x2}) \wedge (\mathrm{true}) \Rightarrow \neg\mathrm{x2} \Rightarrow \mathrm{true}$, therefore, is satisfiable

Resolution-based Elimination

❑Pure Literals

- ➢A literal is pure if only occurs as a positive literal or as a negative literal in a CNF formula
- ➢Example: $\varphi = (\neg x1 \vee x2) \wedge (x3 \vee \neg x2) \wedge (x4 \vee \neg x5) \wedge (x5 \vee \neg x4)$
  - $\neg x1$ and $x3$ are pure literals

❑Pure Literal Elimination

- ➢We can assign pure literals true, so that clauses containing pure literals can be eliminated, strictly reduce the number of clauses
- ➢Example: $\varphi = (\neg x1 \vee x2) \wedge (x3 \vee \neg x2) \wedge (x4 \vee \neg x5) \wedge (x5 \vee \neg x4)$
  - Reduced to $\varphi' = (x4 \vee \neg x5) \wedge (x5 \vee \neg x4)$,
  - which then is reduced to $\varphi'' = x4 \vee \neg x4 = \mathrm{true}$ (apply General Resolution)

$$\boxed{\dfrac{(A \vee x) \wedge (B \vee \neg x)}{A \vee B}}$$

Intuition: the two clauses fore “either A or B must be true”, and once that is true, we can always choose a value for x to satisfy both clauses

Resolution-based Elimination

❑The general form:

- ➢Let a CNF formula be partitioned as: $\varphi = \bigwedge_{\mathrm{i}}(A_i \vee \mathrm{x}) \wedge \bigwedge_{\mathrm{j}}(B_j \vee \neg\mathrm{x}) \wedge \mathrm{R}$
  - ○ $A_i, B_j, and\ R$ does not contain literal x or $\neg x$
- ➢Resolution-based elimination removes variable x and produces:
  - ○ $\varphi' = \bigwedge_{\mathrm{i,j}}(A_i \vee B_j) \wedge \mathrm{R}$
    - ❖Resolving every $(A_i \vee \mathrm{x})$ with every $(B_j \vee \neg\mathrm{x})$, producing resolvents $(A_i \vee B_j)$
    - ❖Deleting all clauses containing x or $\neg x$
  - ○ φ' preserves satisfiability of φ: φ is satisfiable $\Leftrightarrow$ φ' is satisfiable

❑Proof:

- ➢($\Longrightarrow$) Soundness: exists assignment α s.t., $\alpha \vDash \varphi$, then $\alpha' \vDash \varphi'$
  - ○ α' is α removing x, need to show $\forall \mathrm{i,j}.\, \alpha \vDash (A_i \vee \mathrm{x}) \wedge (B_j \vee \neg\mathrm{x}) \Rightarrow \alpha' \vDash A_i \vee B_j$
- ➢($\Longleftarrow$) Completeness: assume $\beta \vDash \varphi'$, show $\beta \cup \{\mathrm{x} \coloneqq ?\} \vDash \varphi$
  - ○ If exists $A_i = \mathrm{false}$, then $\beta \vDash \varphi' \Rightarrow$ all B_j are true, then $\beta \cup \{\mathrm{x} \coloneqq \mathrm{true}\} \vDash \varphi$
  - ○ If exists $B_j = \mathrm{false}$, then $\beta \vDash \varphi' \Rightarrow$ all $\mathrm{A_i}$ are true, then $\beta \cup \{\mathrm{x} \coloneqq \mathrm{false}\} \vDash \varphi$
  - ○ Otherwise, x can be either true or false, $\beta \cup \{\mathrm{x} \coloneqq ?\} \vDash \varphi$

Resolution-based Elimination

- **Pure literal elimination and Unit propagation are special cases of the general form of resolution**
  - Pure literal elimination: the set of resolvents $\{(A_i \vee B_j)\}_{\mathrm{i,j}} = \emptyset$
    - If the pure literal is x, then there is no B_j
    - If the pure literal is $\neg x$, then there is no A_i
    - equivalent to removing all containing clauses
  - Unit propagation:
    - Without losing generality, assume the unit clause is $(x) \equiv (\mathrm{false} \vee x)$, then
    - the CNF formula is in this form: $\varphi = \mathrm{x} \wedge \bigwedge_{\mathrm{i}}(A_i \vee \mathrm{x}) \wedge \bigwedge_{\mathrm{j}}(B_j \vee \neg\mathrm{x}) \wedge \mathrm{R}$, and
    - $\mathrm{x} \wedge \bigwedge_{\mathrm{j}}(B_j \vee \neg\mathrm{x})$ is reduced to $\bigwedge_{\mathrm{j}}(B_j \vee \mathrm{false}) \equiv \bigwedge_{\mathrm{j}} B_j$, then
    - φ is reduced to $\bigwedge_{\mathrm{i}}(A_i \vee \mathrm{x}) \wedge \bigwedge_{\mathrm{j}} B_j \wedge \mathrm{R}$, then
    - apply pure literal elimination and obtain $\bigwedge_{\mathrm{j}} B_j \wedge \mathrm{R}$

A first try: Solving SAT without search?

- **Algorithm: Resolution with refinements**
  - Iteratively apply the following steps:
    - Apply the pure literal rule and unit propagation
    - Select variable x
    - Apply resolution between every pair of clauses of the form $(x \vee \alpha)$ and $(\neg x \vee \beta)$
    - Remove all clauses containing either x or ¬x
  - Terminate when
    - either the empty clause is derived, implying UNSAT
    - or the empty formula is derived, implying SAT
- **Example:**

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \models$$

$$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \models$$

$$(x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \models$$

$$x_3 \quad \models$$

$$\top$$

What is the problem of the first try?

❑Problems:

- In which order should the resolution steps be performed?
  - Which variable should be selected during the algorithm?
- Huge memory consumption
  - $\varphi = \bigwedge_{\mathrm{i}}(A_i \vee \mathrm{x}) \wedge \bigwedge_{\mathrm{j}}(B_j \vee \neg\mathrm{x}) \wedge \mathrm{R}$ is reduced to $\varphi' = \bigwedge_{\mathrm{i,j}}(A_i \vee B_j) \wedge \mathrm{R}$
  - The number of clauses changes from $\mathrm{i} + \mathrm{j} + |\mathrm{R}|$ to $\mathrm{i} * \mathrm{j} + |\mathrm{R}|$
  - Do not scale well even using a ROBDD representation

❑Variable elimination at work nowadays

- Variable elimination can be used as a preprocessing step or inprocessing step if the operation does not increase the number of clauses
- Pure literal elimination and unit propagation can be used
- Using backtrack search when clauses do not decrease

DPLL: Efficient SAT solving in practice

- **Key Innovation #1: unit propagation**

$$(b \vee c) \wedge (\mathbf{a}) \wedge (\neg a \vee c \vee d) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d \vee \neg a) \wedge (b \vee d)$$

  - In this example, a appears alone. It must be true. Reduced to

$$(b \vee c) \wedge (c \vee d) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d) \wedge (b \vee d)$$

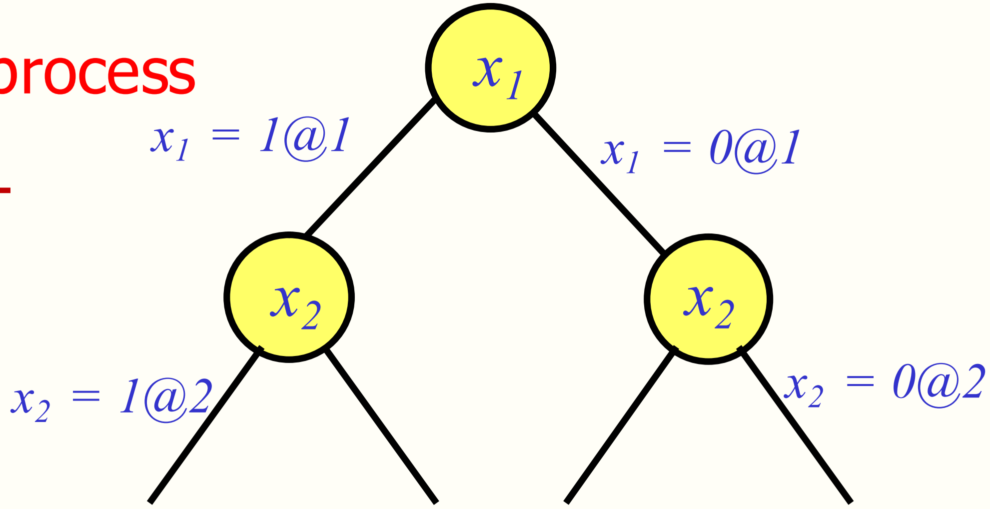
- **Key Innovation #2: pure literal elimination**
  - Note that b appears only positively. **Setting b to true** can only help us, not hurt us!
  - Reduced to $(c \vee d) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d)$
- **When we are stuck, guess and backtrack when necessary**
  - Step 1: guess c is true! Then we get $d \wedge \neg d$, which is unsatisfiable
  - Step 2: guess c is false! Then we get d, which is satisfiable

Review Terminology

- **Let φ be a CNF formula of n variables and m clauses**
- **Assignment*:* set of (*variable, value*) pairs**
  - Example: $\mathrm{A} = \{\mathrm{x}_1 = 0, x_2 = 1, x_3 = 0, x_4 = 1\}$ is an assignment of $\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3$, where $\omega_1 = (x_2 \vee x_3), \omega_2 = (\neg x_1 \vee \neg x_4), \omega_3 = (\neg x_2 \vee x_4)$
  - $|A| < n \rightarrow$ partial assignment $\{x_1 = 0, x_2 = 1, x_4 = 1\}$
  - $|A| = n \rightarrow$ complete assignment $\{x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 1\}$
  - $\varphi|_A = 0 \rightarrow$ falsifying assignment $\{x_1 = 1, x_4 = 1\}$
  - $\varphi|_A = 1 \rightarrow$ satisfying assignment $\{x_1 = 0, x_2 = 1, x_4 = 1\}$
  - $\varphi|_A = \mathrm{A} \rightarrow$ unresolved assignment $\{x_1 = 0, x_2 = 0, x_4 = 1\}$
- An assignment partitions the clauses into three classes:
  - Satisfied, falsified, unresolved
- **Free literal**: an unassigned literal
- **Unit clause**: has exactly one free literal

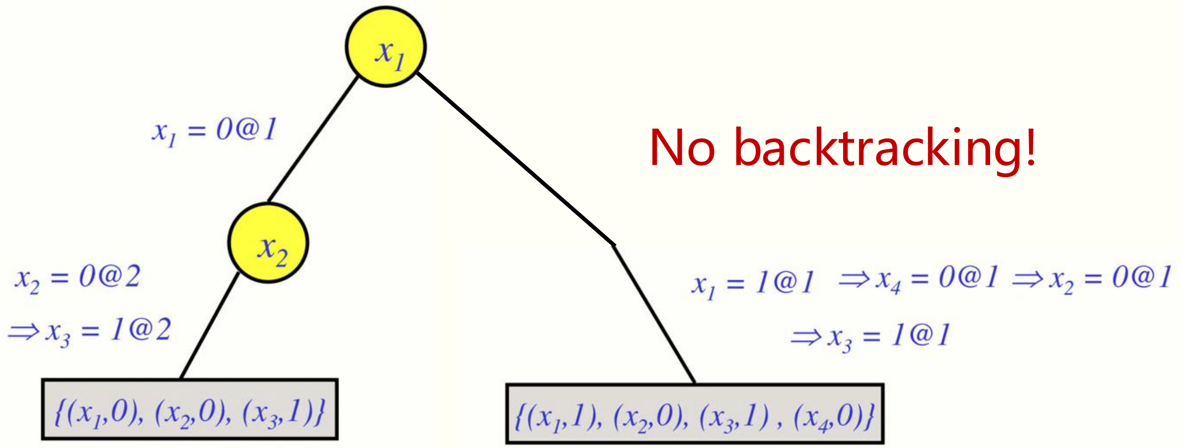
Basic Backtracking Search

- **Organize the search in the form of a decision tree**
  - Each node is a decision variable
  - Outgoing edges: assignment to the decision variable
  - Depth of node in decision tree is decision level $\delta(x)$
  - "x=v@d" means variable x is assigned value v at decision level d
- **Backtracking Search:**
  - Make new decision assignments
  - Infer implied assignments by a deduction process
    - E.g., unit propagation
  - May find a satisfying assignment, SAT
  - May find a falsifying assignment,
    - Also called conflicting assignments
      - Top level leads to UNSAT
      - Otherwise, leads to backtrack

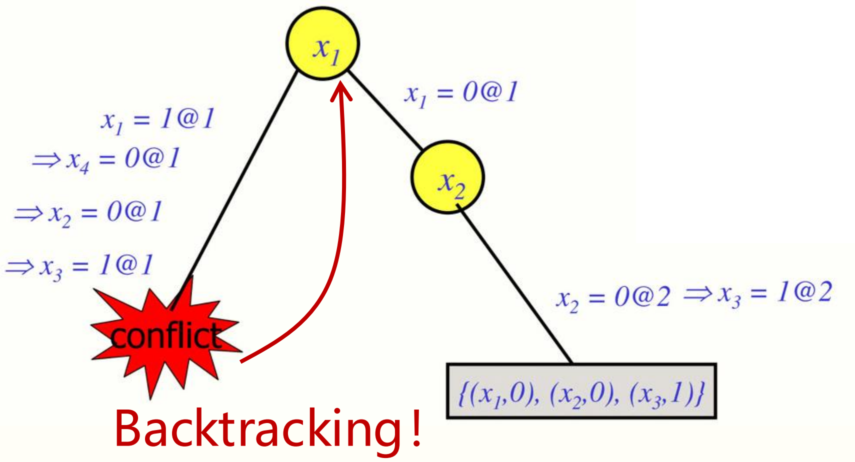


Basic Backtracking Search: Example

- $\varphi = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_4) \wedge (\neg x_2 \vee x_4)$



- $\varphi = (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_4) \wedge (\neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$



DPLL: Efficient SAT solving in practice

❑Davis-Putnam-Logemann-Loveland Algorithm

➢Conflict: an assignment **falsifies** a clause

```
DPLL(F):
  Apply unit propagation
  If conflict identified, return UNSAT
  Apply the pure literal rule
  If F is satisfied (and possibly empty), return SAT
  Select unassigned variable x
    If DPLL(F ∧ x)=SAT return SAT
    return DPLL(F ∧ ¬x)
```

Backtrack search

```
function DPLL(φ)
    if φ = true then
        return true
    end if
    if φ contains a false clause then
        return false
    end if
    for all unit clauses l in φ do
        φ ← UNIT-PROPAGATE(l, φ)
    end for
    for all literals l occurring pure in φ do
        φ ← PURE-LITERAL-ASSIGN(l, φ)
    end for
    l ← CHOOSE-LITERAL(φ)
    return DPLL(φ ∧ l) ∨ DPLL(φ ∧ ¬l)
end function
```

➢Memory Efficient than the first try

➢Commonly denote complete solvers for SAT

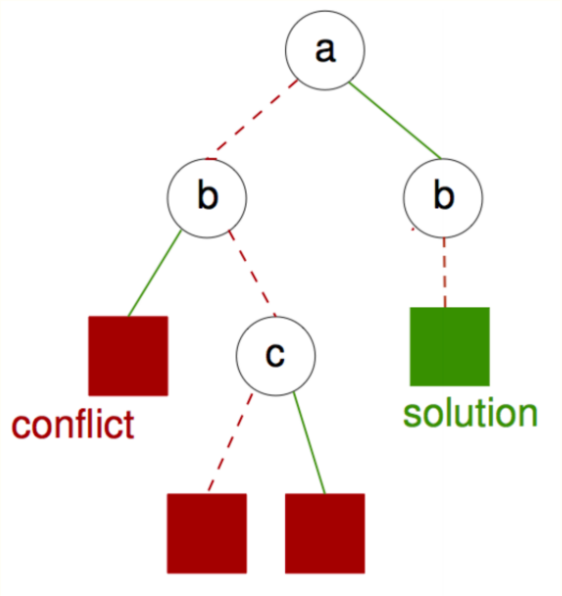
Variable Selecting Heuristics

- **Focus in 90’s was to improve the heuristics to select the variables**
- **Clause-set–based heuristics**
  - Prefer variables that appear in the shortest clauses
  - Prefer variables that occur most frequently
- **History-based heuristics**
  - Prefer variables that have caused conflicts before
- **Many others …**

DPLL SAT Solver: An Example

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

- **{a:=false}:** $(\neg b \vee d) \wedge (\neg b \vee e) \wedge (\neg b \vee \neg d \ \vee \neg e) \wedge (b \vee c \vee d) \wedge (b \ \vee c \ \vee \neg d) \wedge (b \vee \neg c \vee e) \wedge (b \vee \neg c \vee \neg e)$
  - {a:=false, b:=true}: $d \wedge e \ \wedge (\neg d \vee \neg e)$
    - Apply unit propagation: $e \wedge \neg e$, meet a conflict
  - {a:=false, b:=false}: $(c \vee d) \wedge (c \vee \neg d) \wedge (\neg c \vee e) \wedge (\neg c \ \vee \neg e)$
    - {a:=false, b:=false, c:=false}: $d \wedge \neg d$, meet a conflict
    - {a:=false, b:=false, c:=true}: $e \wedge \neg e$, meet a conflict
- **{a:=true}:**$(\neg b \vee \neg d \ \vee \neg e) \wedge \neg b$
  - Apply unit propagation: true, SAT
- **DPLL uses chronological backtracking**
  - Backtrack one level up



Backtrack forms a Decision Tree

Conflict Driven Clause Learning Solvers

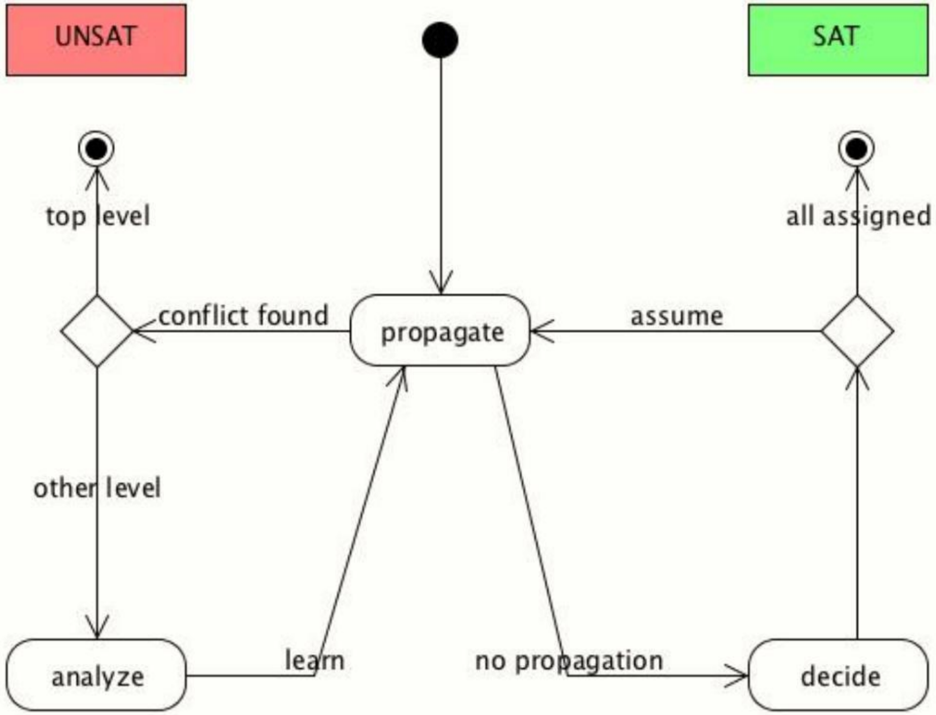
❑GRASP: An efficient CDCL SAT solver

- ➢Generalized seaRch Algorithm for the Satisfiability Problem (Silva, Sakallah, '99)

❑Key Features:

- ➢Implication graphs for Unit Propagation and conflict analysis
- ➢Learning of new clauses
- ➢Non-chronological backtracking
  - ○ 非按时间顺序的回溯

Joao Marques-Silva
关键算法CDCL提出人



GRASP architecture

Top-level of GRASP-like SAT Solver

- **Decision Heuristics:** usually choose the variable that satisfies the most clauses
- **Propagate:** unit propagation, force other liberals in an already satisfied clause to be false, and maintain an implication graph
- **Learn clauses from conflicts:** analyse the implication graph when a conflict arises, learn and add a new clause that would prevent the occurrence of the same conflict in the future

```
CurAsgn = {};
while (true) {
    while (value of φ under CurAsgn is unknown) {
        DecideLit();    // Add decision literal to CurAsgn.
        Propagate();    // Add forced literals to CurAsgn.
    }
    if (CurAsgn satisifies φ) {return true;}
    Analyze conflict and learn a new clause;
    if (the learned clause is empty) {return false;}
    Backtrack();
    Propagate(); // Learned clause will force a literal
}
```

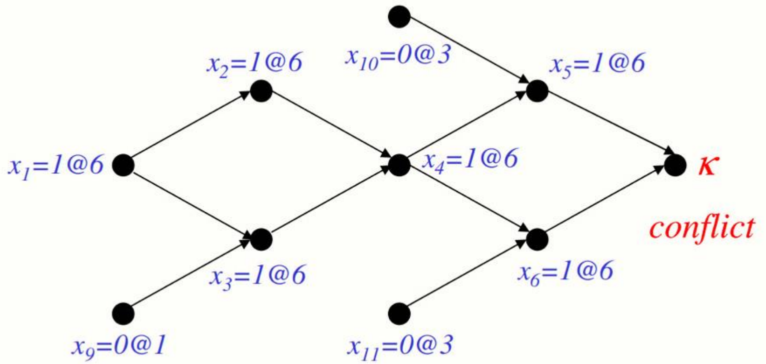
- **Non-chronological Backtrack:** determine decision level to backtrack to, and this might not be the immediate one

Implication Graphs

- **Antecedent assignment:**
  - If a variable x is forced by a clause during Unit Propagation, then assignment of 0 to all other literals in the clause is called the **antecedent assignment** $A(x)$
    - Variables directly responsible for forcing the value of x
    - Antecedent assignment of a decision variable is empty
  - Example: for $\omega = (x \vee y \vee \neg z)$
    - $A(x) = \{y = 0, z = 1\}, A(y) = \{x = 0, z = 1\}, A(z) = \{x = 0, y = 0\}$
- **Implication Graphs:** Depicts the antecedents of assigned variables
  - A **node** is an assignment (either via decision or implied) to a variable
  - Edges: Predecessors of x correspond to antecedent $A(x)$
    - No predecessors for decision assignments!

Implication Graphs： Example

- **Formula:** $\boldsymbol{\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6 \wedge \omega_7 \wedge \omega_8 \wedge \omega_9}$
  - $\omega_1 = (\neg x_1 \vee x_2), \omega_2 = (\neg x_1 \vee x_3 \vee x_9), \omega_3 = (\neg x_2 \vee \neg x_3 \vee x_4)$
  - $\omega_4 = (\neg x_4 \vee x_5 \vee x_{10}), \omega_5 = (\neg x_4 \vee x_6 \vee x_{11}), \omega_6 = (\neg x_5 \vee \neg x_6)$
  - $\omega_7 = (x_1 \vee x_7 \vee \neg x_{12}), \omega_8 = \{x_1 \vee x_8\}, \omega_9 = (\neg x_7 \vee \neg x_8 \vee \neg x_{13})$
- **Assume that**
  - Current true assignment: $\{x_9 = 0@1, x_{12} = 1@2, x_{13} = 1@2, x_{10} = 0@3, x_{11} = 0@3\}$
  - Current decision assignment: $\{x_1 = 1@6\}$
  - “x=v@d” means variable x is assigned value v at decision level d



Learning new clauses from conflicts

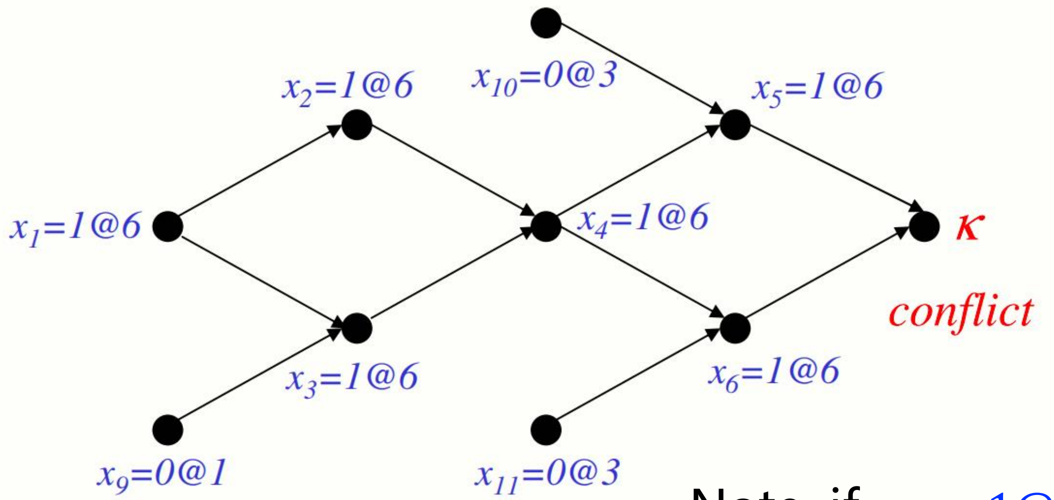
- ❑**Goal: Learning new clauses allows to deduce more forced literals, pruning the search space**
- ❑**New clauses learned from conflicting assignments**
  - ➢Let CA be the assignment of 0 to all literals in the falsified clause
  - ➢A literal l $\in$ CA is a unique implication point (UIP) iff every other literal in CA has an earlier decision level than l

```
func learning(CA):
  while(true):
    remove the most recently assigned literal from CA and replace it by its antecedent
    if (CA is empty or has a UIP) break;
  φ = false // save learned clause
  if CA is not empty:
    Let {L_1,…,L_n} = CA, φ = (¬L_1 ∨ ⋯ ∨ ¬L_n)
  return φ
```

  - ➢Add the new clause to the clause database permanently

Learning new clauses from conflicts

- **Example:** $\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6 \wedge \omega_7 \wedge \omega_8 \wedge \omega_9$
  - $\omega_1 = (\neg x_1 \vee x_2), \omega_2 = (\neg x_1 \vee x_3 \vee x_9), \omega_3 = (\neg x_2 \vee \neg x_3 \vee x_4)$
  - $\omega_4 = (\neg x_4 \vee x_5 \vee x_{10}), \omega_5 = (\neg x_4 \vee x_6 \vee x_{11}), \omega_6 = (\neg x_5 \vee \neg x_6)$
  - $\omega_7 = (x_1 \vee x_7 \vee \neg x_{12}), \omega_8 = \{x_1 \vee x_8\}, \omega_9 = (\neg x_7 \vee \neg x_8 \vee \neg x_{13})$
- $CA = \{x_9 = 0@1, x_{12} = 1@2, x_{13} = 1@2, x_{10} = 0@3, x_{11} = 0@3, x_1 = 1@6, \boldsymbol{x_2} = 1@6, \boldsymbol{x_3} = 1@6, \boldsymbol{x_4} = 1@6, \boldsymbol{x_5} = 1@6, \boldsymbol{x_6} = 1@6\}$ leads $\boldsymbol{\omega_6}$ to be a conflict



- **Learning**: CA = $\{\boldsymbol{x_5} = 1@6, \boldsymbol{x_6} = 1@6\}$
- Step 1: remove $\boldsymbol{x_6} = 1@6$, add its antecedent
  - CA = $\{\boldsymbol{x_5} = 1@6, \boldsymbol{x_4} = 1@6, x_{11} = 0@3\}$
- Step 2: remove $\boldsymbol{x_5} = 1@6$, add its antecedent
  - CA= $\{x_{10} = 0@3, \boldsymbol{x_4} = 1@6, x_{11} = 0@3\}$
- $\boldsymbol{x_4} = 1@6$ is now a UIP, we learn a new clause
  - ω' = $(x_{10} \vee \neg x_4 \vee x_{11})$

Note, if $\boldsymbol{x_4} = 1@6$ is removed at Step 2, we may learn another new clause:

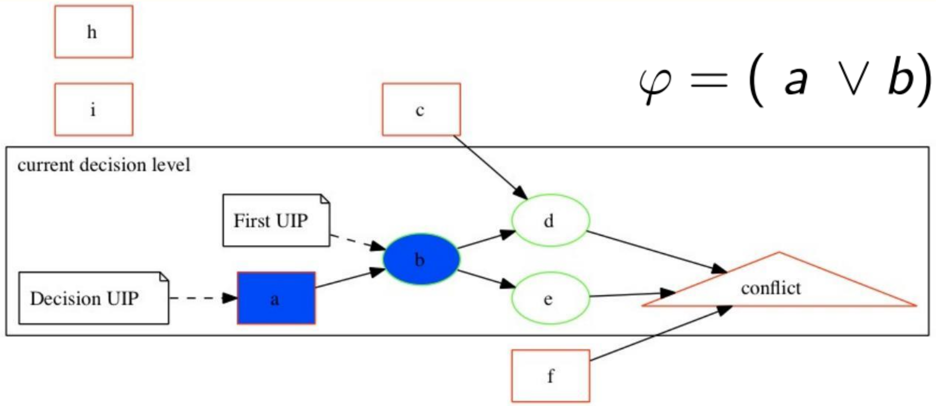
ω'' = $(\neg x_1 \vee x_9 \vee x_{10} \vee x_{11})$

Non-chronological backtrack

- **Normal backtracking flips most recent decision setting**
- **Latest decision may not have caused conflict**
  - Due to learned clauses
- **Non-chronological backtracking flips most recent open decision contributing to conflict**
  - learned clause becomes a unit clause after backtracking
- **Easy to implement along with learning**
  - For each variable setting, maintain decision level, height of decision stack at time of setting
  - Backtrack to latest decision level of literals in learned clause
- **Example:** $\mathrm{a} = 1@1, \mathrm{b} = 1@2, \mathrm{c} = 0@3, \mathrm{d} = 1@4, \mathrm{e} = 0@5, \mathrm{f} = 1@6, \mathrm{g} = 0@7, \mathrm{h} = 1@8$, **and the learned clause** $(\neg \mathrm{a} \vee \neg \mathrm{d} \vee \mathrm{g})$
  - a, d, and g are at levels 1, 4, and 7 respectively
  - Backjump directly to level 4, not 7 or 6 or 5

Some remarks about UIPs

- **Many possibilities to derive a clause using UIP**
  - As noted, if $x_4 = 1@6$ is removed at Step 2 in a previous slide, we may learn another new clause: ω'' = $(\neg x_1 \vee x_9 \vee x_{10} \vee x_{11})$



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

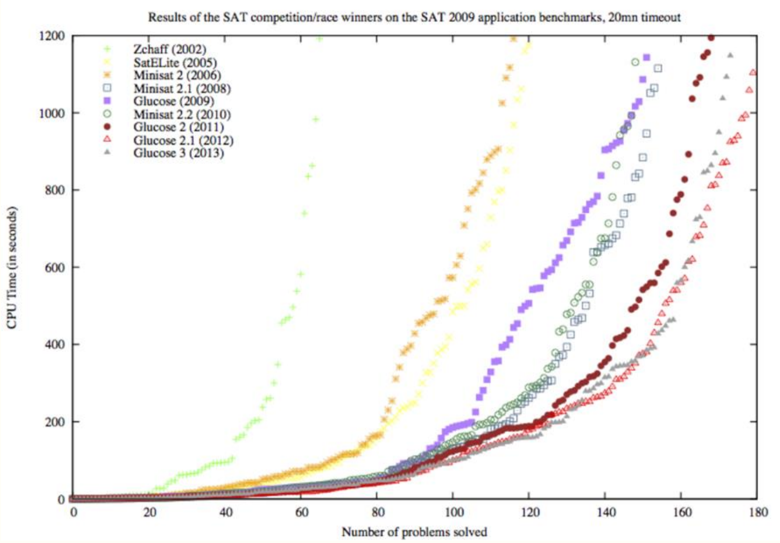
  - Decision UIP always flip the decision variable truth value: the search is thus driven by the heuristics
  - Using other UIP scheme, the value of any of the literal propagated at the current decision level may be flipped: the search is thus driven by the conflict analysis
  - Generic name for GRASP approach: Conflict Driven Clause Learning (**CDCL**) solver [Ryan 2004].

More CDCL SAT Solvers

Sharad Malik

❑CHAFF: Engineering an Efficient SAT Solver (DAC 2001)

- ➢ 2 order of magnitude speedup on UNSAT instances
- ➢ Based on careful analysis of GRASP internals
  - o New adaptative heuristic: Variable State Independent Decaying Sum
  - o New lazy data structure: Watched literals
  - o New conflict analysis approach: First UIP



❑Since CHAFF …

- ➢ International SAT race every year
- ➢ Many CDCL solvers have been released
  - o Minisat, Glucose,…
- ➢ SAT has been integrated into solve combinatorial problems
- ➢ Many papers published on the design of efficient SAT solvers
  - o … but a big part of the knowledge still lies in **source code**!

(Optional) Greedy SAT: Local Search

```
function GSAT(CNF c, int maxtries, int maxflips) {
  // DIVERSIFICATION STEP
  for (int i = 0; i < maxtries; i++) {
    m = randomAssignment();
    // INTENSIFICATION STEP
    for (int j=0; j < maxflips; j++) {
      if(m satisfies c) return SAT;
      flip (m) ;
    }
  }
  return UNKNOWN;
}
```

use of Random Walks for finding the local minima

- The decision procedure is very simple to implement and very fast!
- Efficiency depends on which literal to flip, and the parameter values
- GSAT is incomplete, cannot answer UNSAT
- Lesson: An agile (fast) SAT solver sometimes better than a clever one!
  - 本地搜索和CDCL, DPLL互补， 在一些CDCL,DPLL效果不好的场合能起到很好的作用

(Optional) 作业：

❑DPLL 算法应用：

- Show how DPLL (unit propagation, pure literal elimination, choosing a literal, backtracking) applies to the following formula:

$$(a \lor b) \land (a \lor c) \land (\neg a \lor c) \land (a \lor \neg c) \land (\neg a \lor \neg c) \land (\neg d)$$

❑GRASP中learning算法应用：

- $x_1 = 1@1, x_3 = 1@2, x_8 = 1@3, x_9 = 1@3$
- Falsifying ω_4
- Find the learning clause with implication graph

$\omega_1 = (\neg x_1 \lor x_8 \lor x_9)$

$\omega_2 = (\neg x_1 \lor x_8 \lor \neg x_9)$

$\omega_3 = (\neg x_1 \lor \neg x_8 \lor x_9)$

$\omega_4 = (\neg x_1 \lor \neg x_8 \lor \neg x_9)$

$\omega_5 = (x_1 \lor x_3)$

$\omega_6 = (x_1 \lor \neg x_3)$