Understanding and using SAT solvers

A practitioner perspective

Daniel Le Berre [1]

CRIL-CNRS UMR 8188

Summer School 2009: Verification Technology, Systems & Applications
Nancy, October 12-16, 2009

1. Contains material provided by Prof. Joao Marques Silva

Agenda

Introduction to SAT

Early approaches to tackle SAT problems

The CDCL framework

From GRASP to CHAFF

Anatomy of a modern CDCL SAT solver

Practicing SAT

Some results from the SAT Competition 2009

Disclaimer

- Not a complete view of the subject
- Limited to one branch of SAT research (CDCL solvers)
- From an AI background point of view
- From a SAT solver designer
- For a broader picture of the area, see the handbook edited this year by the community

Disclaimer : continued

- Remember that the best solvers for practical SAT solving in the 90's where based on local search
- This decade has been the one of Conflict Driven Clause Learning solvers.
- The next one may rise a new kind of solvers ...

Agenda

The SAT problem

Definition

Input : A set of clauses built from a propositional language with n variables.

Output : Is there an assignment of the n variables that satisfies all those clauses ?

The SAT problem

Definition

Input : A set of clauses built from a propositional language with n variables.

Output : Is there an assignment of the n variables that satisfies all those clauses ?

Example

$$C_1 = \{\neg a \vee b, \neg b \vee c\} = (\neg a \vee b) \wedge (\neg b \vee c) = (a' + b).(b' + c)$$

$$C_2 = C_1 \cup \{a, \neg c\} = C_1 \wedge a \wedge \neg c$$

For C_1, the answer is yes, for C_2 the answer is no

$$C_1 \models \neg(a \wedge \neg c) = \neg a \vee c$$

Where are clauses coming from ? AI point of view

Suppose :

a *I like free software*

b *I should start a free software project*

c *I should use a free software language*

Then C_1 could represent the beliefs :

- $a \rightarrow b$: *If I like free software, then I should start a free software project.*
- $b \rightarrow c$: *If I start a free software project, then I should use a free software language.*

What happens if I like free software and I do not use a free software language $(a \wedge \neg c)$? This is inconsistent with my beliefs. From C_1 I can deduce $a \rightarrow c$: *If I like free software, then I should use a free software language.*

Typical application for SAT nowadays : reachability analysis

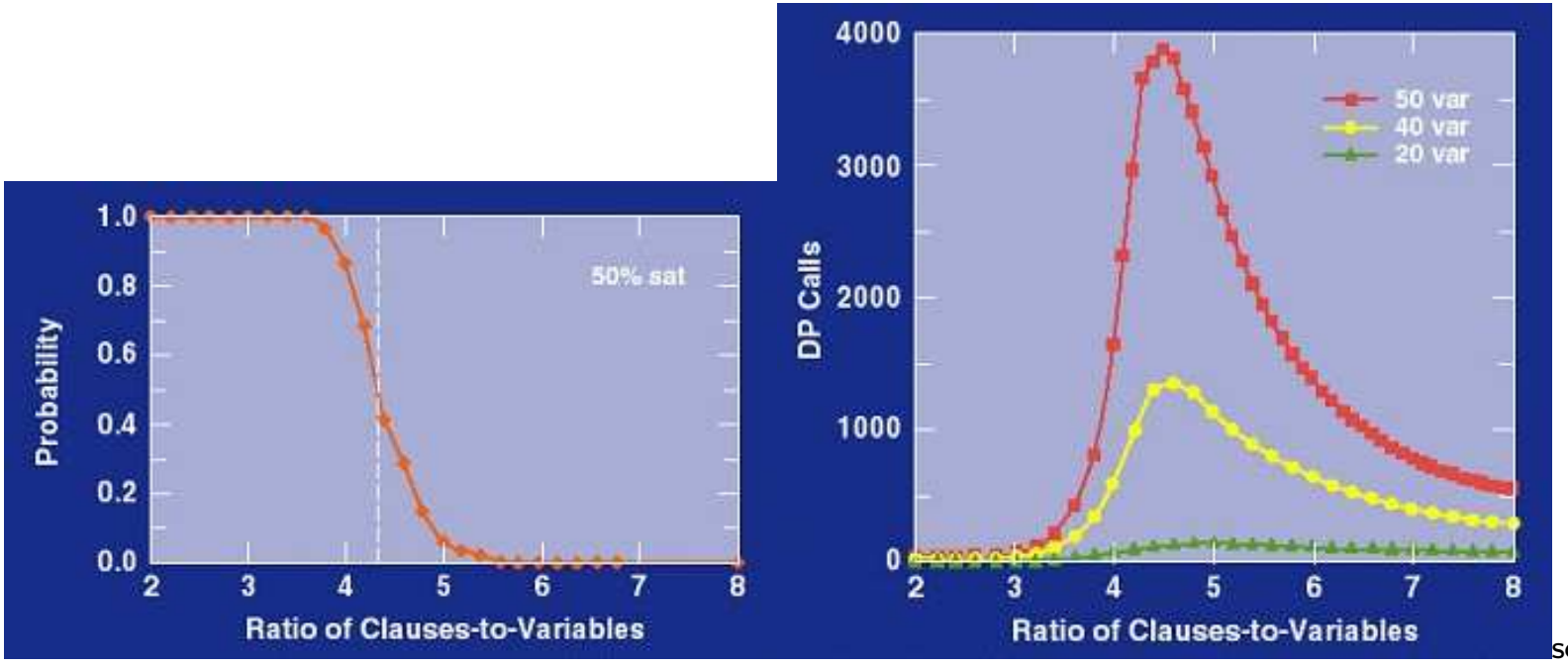
Definition

Given an initial state s_0, a state transition relation ST, a goal state g and a bound k.
Is there a way to reach g from s_0 using ST within k steps ?
Is there a succession of states $s_0, s_1, s_2, ..., s_k = g$ such that $\forall\ 0 \leq i < k\ (s_{i-1}, s_i) \in ST$?

- The problems are generated for increasing k.
- For small k, the problems are usually UNSATISFIABLE
- For larger k, the problems can be either SAT or UNSAT.
- Complete SAT solvers are needed !

SAT is important in theory ...

- Canonical NP-Complete problem [Cook, 1971]
- Threshold phenomenon on randomly generated k-SAT instances [Mitchell,Selman,Levesque, 1992]



source :

http ://www.isi.edu/ szekely/antsebook/ebook/modeling-tools-and-techniques.htm

... and in practice : Computer Aided Verification Award 2009

awarded to

Conor F. Madigan
Sharad Malik
Joao Marques-Silva
Matthew Moskewicz
Karem Sakallah
Lintao Zhang
Ying Zhao

for

fundamental contributions to the development of high-performance Boolean satisfiability solvers.

Authors of GRASP SAT solver
Authors of CHAFF SAT solver

Verification Technology
Systems
Applications

Where can we find SAT technology today ?

- Formal methods :
  - Hardware model checking ; Software model checking ; Termination analysis of term-rewrite systems ; Test pattern generation (testing of software & hardware) ; etc.
- Artificial intelligence :
  - Planning ; Knowledge representation ; Games (n-queens, sudoku, social golpher's, etc.)
- Bioinformatics :
  - Haplotype inference ; Pedigree checking ; Analysis of Genetic Regulatory Networks ; etc.
- Design automation :
  - Equivalence checking ; Delay computation ; Fault diagnosis ; Noise analysis ; etc.
- Security :
  - Cryptanalysis ; Inversion attacks on hash functions ; etc.

Where can we find SAT technology today ? II

- Computationally hard problems :
  - Graph coloring ; Traveling salesperson ; etc.
- Mathematical problems :
  - van der Waerden numbers ; Quasigroup open problems ; etc.

- Core engine for other solvers : 0-1 ILP/Pseudo Boolean ; QBF ; #SAT ; SMT ; MAXSAT ; ...
- Integrated into theorem provers : HOL ; Isabelle ; ...
- Integrated into widely used software :
  - Suse 10.1 dependency manager based on a custom SAT solver.
  - Eclipse provisioning system based on a Pseudo Boolean solver.

Agenda

Boolean Formulas

- Boolean formula φ is defined over a set of propositional variables $x_1, \ldots, x_n$, using the standard propositional connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, and parenthesis
  - The domain of propositional variables is $\{T, F\}$
  - Example : $\varphi(x_1, \ldots, x_3) = ((\neg x_1 \wedge x_2) \vee x_3) \wedge (\neg x_2 \vee x_3)$
- A formula φ in conjunctive normal form (CNF) is a conjunction of disjunctions (clauses) of literals, where a literal is a variable or its complement
  - Example : $\varphi(x_1, \ldots, x_3) \equiv$
- A formula φ in disjunctive normal form (DNF) is a disjunction of conjunctions (terms) of literals
  - Example : $\varphi(x_1, \ldots, x_3) \equiv$
- Can encode any Boolean formula into Normal Form

Boolean Formulas

- Boolean formula φ is defined over a set of propositional variables $x_1, \ldots, x_n$, using the standard propositional connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, and parenthesis
  - The domain of propositional variables is $\{T, F\}$
  - Example : $\varphi(x_1, \ldots, x_3) = ((\neg x_1 \wedge x_2) \vee x_3) \wedge (\neg x_2 \vee x_3)$
- A formula φ in conjunctive normal form (CNF) is a conjunction of disjunctions (clauses) of literals, where a literal is a variable or its complement
  - Example : $\varphi(x_1, \ldots, x_3) \equiv (\neg x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee x_3)$
- A formula φ in disjunctive normal form (DNF) is a disjunction of conjunctions (terms) of literals
  - Example :
    $\varphi(x_1, \ldots, x_3) \equiv$
- Can encode any Boolean formula into Normal Form

Boolean Formulas

- Boolean formula φ is defined over a set of propositional variables $x_1, \ldots, x_n$, using the standard propositional connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, and parenthesis
  - The domain of propositional variables is $\{T, F\}$
  - Example : $\varphi(x_1, \ldots, x_3) = ((\neg x_1 \wedge x_2) \vee x_3) \wedge (\neg x_2 \vee x_3)$
- A formula φ in conjunctive normal form (CNF) is a conjunction of disjunctions (clauses) of literals, where a literal is a variable or its complement
  - Example : $\varphi(x_1, \ldots, x_3) \equiv (\neg x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee x_3)$
- A formula φ in disjunctive normal form (DNF) is a disjunction of conjunctions (terms) of literals
  - Example : $\varphi(x_1, \ldots, x_3) \equiv (\neg x_1 \wedge x_2 \wedge \neg x_2) \vee (x_3 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2 \wedge x_3) \vee x_3$
- Can encode any Boolean formula into Normal Form

The resolution principle and classical simplification rules

John Alan Robinson, "A Machine-Oriented Logic Based on the Resolution Principle", Communications of the ACM, 5 :23-41, 1965.

$$\textit{resolution:}\quad \frac{x_1 \vee x_2 \vee x_3 \qquad x_1 \vee \neg x_2 \vee x_4}{x_1 \vee x_1 \vee x_3 \vee x_4}$$

$$\textit{merging:}\quad \frac{x_1 \vee x_1 \vee x_3 \vee x_4}{x_1 \vee x_3 \vee x_4}$$

$$\textit{subsumption:}\quad \frac{\alpha \vee \beta \qquad \alpha}{\alpha}$$

The resolution principle and classical simplification rules

John Alan Robinson, "A Machine-Oriented Logic Based on the Resolution Principle", Communications of the ACM, 5 :23-41, 1965.

$$\textit{resolution:}\quad \frac{x_1 \vee x_2 \vee x_3 \qquad x_1 \vee \neg x_2 \vee x_4}{x_1 \vee x_1 \vee x_3 \vee x_4}$$

$$\textit{merging:}\quad \frac{x_1 \vee x_1 \vee x_3 \vee x_4}{x_1 \vee x_3 \vee x_4}$$

$$\textit{subsumption:}\quad \frac{\alpha \vee \beta \qquad \alpha}{\alpha}$$

What happens if we apply resolution between $\neg x_1 \vee x_2 \vee x_3$ and $x_1 \vee \neg x_2 \vee x_4$?

The resolution principle and classical simplification rules

John Alan Robinson, "A Machine-Oriented Logic Based on the Resolution Principle", Communications of the ACM, 5 :23-41, 1965.

$$\textit{resolution:}\quad \frac{x_1 \vee x_2 \vee x_3 \qquad x_1 \vee \neg x_2 \vee x_4}{x_1 \vee x_1 \vee x_3 \vee x_4}$$

$$\textit{merging:}\quad \frac{x_1 \vee x_1 \vee x_3 \vee x_4}{x_1 \vee x_3 \vee x_4}$$

$$\textit{subsumption:}\quad \frac{\alpha \vee \beta \qquad \alpha}{\alpha}$$

What happens if we apply resolution between $\neg x_1 \vee x_2 \vee x_3$ and $x_1 \vee \neg x_2 \vee x_4$?

A tautology : $x_2 \vee \neg x_2 \vee x_3 \vee x_4$.

Applying resolution to decide satisfiability

- Apply resolution between clauses with exactly one opposite literal
- possible outcome :
  - a new clause is derived : removed subsumed clauses
  - the resolvent is subsumed by an existing clause
- until empty clause derived or no new clause derived
- Main issues of the approach :
  - In which order should the resolution steps be performed ?
  - huge memory consumption !

The Davis and Putnam procedure : basic idea

Davis, Martin ; Putnam, Hillary (1960). "A Computing Procedure for Quantification Theory". Journal of the ACM 7 (3) : 201-215.

Resolution used for variable elimination : $(A \vee x) \wedge (B \vee \neg x) \wedge R$ is satisfiable iff $(A \vee B) \wedge R$ is satisfiable.

- Iteratively apply the following steps :
  - 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 $\neg x$
- Terminate when either the empty clause or the empty formula is derived

Proof system : ordered resolution

Variable elimination – An Example

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

Variable elimination – An Example

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

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

Variable elimination – An Example

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

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

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

Variable elimination – An Example

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

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

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

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

Variable elimination – An Example

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

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

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

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

$$x_3 \models$$

Variable elimination – An Example

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

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

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

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

$$x_3 \models$$

$$\top$$

- Formula is SAT

The Davis and Putnam procedure : the refinements

Add specific cases to order variable elimination steps

- 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 $\neg x$
- Terminate when either the empty clause or the empty formula is derived

Pure Literals

- A literal is pure if only occurs as a positive literal or as a negative literal in a CNF formula
  - Example :
    $\varphi = (\neg x_1 \vee x_2) \wedge (x_3 \vee \neg x_2) \wedge (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$
  - $\neg x_1$ and x_3 are pure literals
- Pure literal rule : eliminate first pure literals because no resolvant are produced !
- applying a variable elimination step on a pure literal strictly reduces the number of clauses !
- Preserve satisfiability, not logical equivalency !

Unit Propagation

- Specific case of resolution : only shorten clauses.

$$\textit{unit resolution:} \quad \frac{x_1 \vee x_2 \vee x_3 \qquad \neg x_2}{x_1 \vee x_3}$$

- Preserve logical equivalency :
  $(x_1 \vee x_2 \vee x_3) \wedge \neg x_2 \equiv (x_1 \vee x_3) \wedge \neg x_2$
- Since clauses are shortened, new unit clauses may appear. Empty clauses also !
- Unit propagation : apply unit resolution while new unit clauses are produced.

DP60 : The limits

- The approach runs easily out of memory.
- Even recent attempts using a ROBDD representation [Simon and Chatalic 2000] does not scale well.
- The solution : using backtrack search !

DLL62 : Preliminary definitions

- Propositional variables can be assigned value False or True
  - In some contexts variables may be unassigned
- A clause is satisfied if at least one of its literals is assigned value true

  $(x_1 \vee \neg x_2 \vee \neg x_3)$
- A clause is unsatisfied if all of its literals are assigned value false

  $(x_1 \vee \neg x_2 \vee \neg x_3)$
- A clause is unit if it contains one single unassigned literal and all other literals are assigned value False

  $(x_1 \vee \neg x_2 \vee \neg x_3)$
- A formula is satisfied if all of its clauses are satisfied
- A formula is unsatisfied if at least one of its clauses is unsatisfied

DLL62 : space efficient DP60

Davis, Martin ; Logemann, George, and Loveland, Donald (1962). "A Machine Program for Theorem Proving". Communications of the ACM 5 (7) : 394-397.

- Standard backtrack search
- DPLL(F) :
  - Apply unit propagation
  - If conflict identified, return UNSAT
  - Apply the pure literal rule
  - If F is satisfied (empty), return SAT
  - Select decision variable x
    - If DPLL($F \wedge x$)=SAT return SAT
    - return DPLL($F \wedge \neg x$)

Proof system : tree resolution

Pure Literals in backtrack search

- Pure literal rule :
  Clauses containing pure literals can be removed from the formula (i.e. just satisfy those pure literals)
- Example :
  $\varphi = (\neg x_1 \vee x_2) \wedge (x_3 \vee \neg x_2) \wedge (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$
- The resulting formula becomes :
  $\varphi_{\neg x_1, x_3} = (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$

- if l is a pure literal in Σ, then $\Sigma_l \subset \Sigma$
- Preserve satisfiability, not logical equivalency !

Unit Propagation in backtrack search

- Unit clause rule in backtrack search :
  Given a unit clause, its only unassigned literal must be assigned value True for the clause to be satisfied
- Example : for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value False
- Unit propagation
  Iterated application of the unit clause rule

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

Unit Propagation in backtrack search

- Unit clause rule in backtrack search :
  Given a unit clause, its only unassigned literal must be assigned value True for the clause to be satisfied
- Example : for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value False
- Unit propagation
  Iterated application of the unit clause rule

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

Unit Propagation in backtrack search

- Unit clause rule in backtrack search :
  Given a unit clause, its only unassigned literal must be assigned value True for the clause to be satisfied
- Example : for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value False
- Unit propagation
  Iterated application of the unit clause rule

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

Unit Propagation in backtrack search

- Unit clause rule in backtrack search :
  Given a unit clause, its only unassigned literal must be assigned value True for the clause to be satisfied
- Example : for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value False
- Unit propagation
  Iterated application of the unit clause rule

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

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

Unit Propagation in backtrack search

- Unit clause rule in backtrack search :
  Given a unit clause, its only unassigned literal must be assigned value True for the clause to be satisfied
- Example : for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value False
- Unit propagation
  Iterated application of the unit clause rule

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

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

Unit Propagation in backtrack search

- Unit clause rule in backtrack search :
  Given a unit clause, its only unassigned literal must be assigned value True for the clause to be satisfied
- Example : for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value False
- Unit propagation
  Iterated application of the unit clause rule

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

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

Unit Propagation in backtrack search

- Unit clause rule in backtrack search :
  Given a unit clause, its only unassigned literal must be assigned value True for the clause to be satisfied
- Example : for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value False
- Unit propagation
  Iterated application of the unit clause rule

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

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

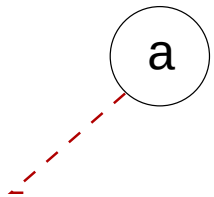
Unit propagation can satisfy clauses but can also unsatisfy clauses (i.e. conflicts)

An Example of DPLL

$$
\begin{aligned}
\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\
& (\neg b \vee \neg d \vee \neg e) \wedge \\
& (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\
& (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)
\end{aligned}
$$

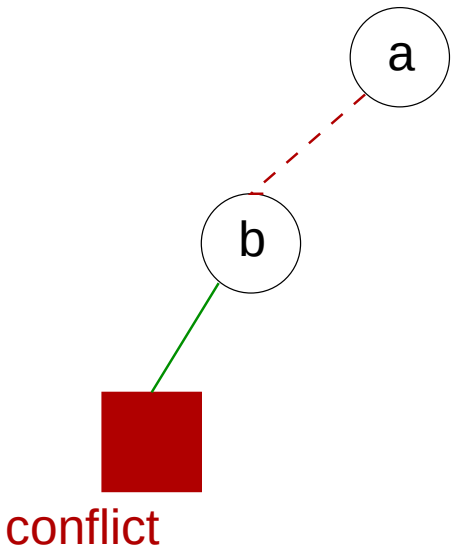
An Example of DPLL

$$\begin{aligned}\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



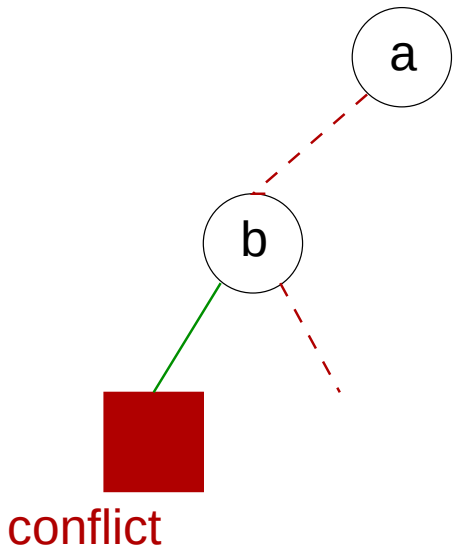
An Example of DPLL

$$\begin{aligned}\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



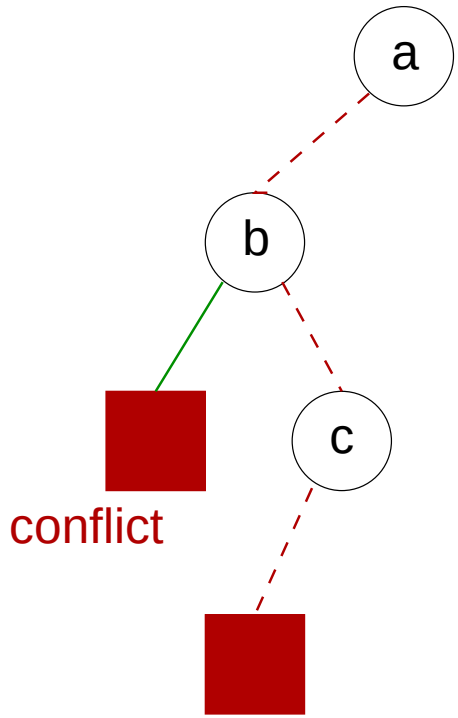
An Example of DPLL

$$\begin{aligned}\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



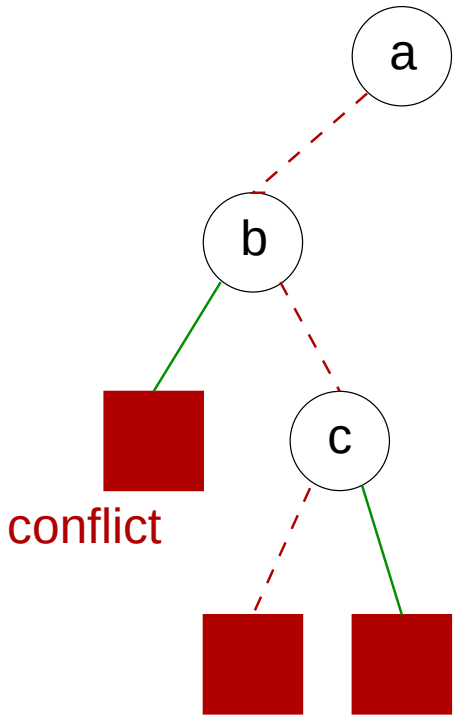
An Example of DPLL

$$\begin{aligned}\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



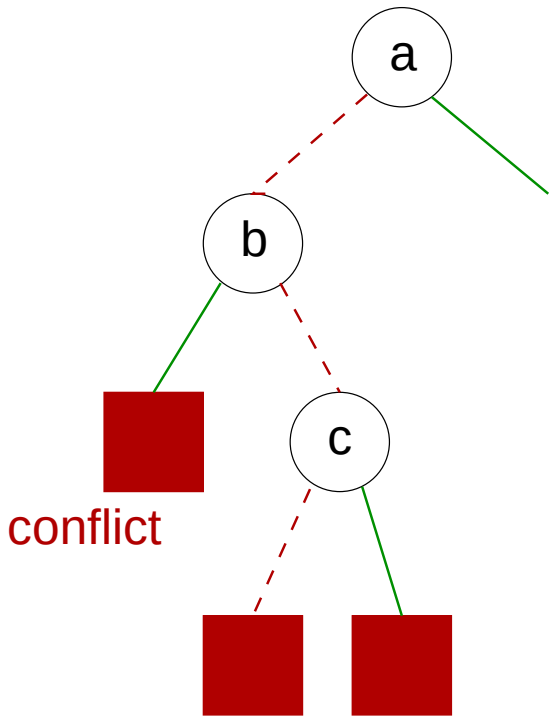
An Example of DPLL

$$
\begin{aligned}
\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\
& (\neg b \vee \neg d \vee \neg e) \wedge \\
& (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\
& (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)
\end{aligned}
$$



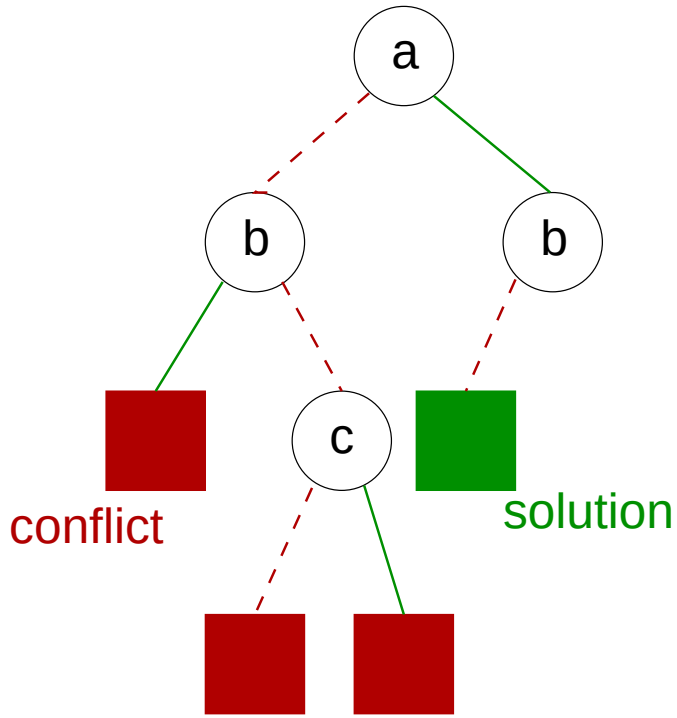
An Example of DPLL

$$\begin{aligned}\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



An Example of DPLL

$$\begin{aligned}\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



DP, DLL or DPLL ?

- DPLL = DP + DLL
- Acknowledge the principles in DP60 and their memory efficient implementation in DP62
- DPLL commonly used to denote complete solvers for SAT : no longer true for modern complete SAT solvers.
- The focus of researchers in the 90's was mainly to improve the heuristics to select the variables to branch on on randomly generated formulas.
- Introduction of non chronological backtracking and learning to solve structured/real world formulas

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

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

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

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

  - Assume decisions $c = \mathit{False}$ and $f = \mathit{False}$

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

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

  - Assume decisions $c = \mathit{False}$ and $f = \mathit{False}$
  - Assign $a = \mathit{False}$ and imply assignments

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

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

  - Assume decisions $c = \mathit{False}$ and $f = \mathit{False}$
  - Assign $a = \mathit{False}$ and imply assignments

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

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

  - Assume decisions $c = \mathit{False}$ and $f = \mathit{False}$
  - Assign $a = \mathit{False}$ and imply assignments
  - A conflict is reached : $(\neg d \vee \neg e \vee f)$ is unsatisfied

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

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

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

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

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

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

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

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

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

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

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$\begin{aligned}\varphi \quad = \quad & (a \vee b) \wedge (\neg b \vee \ c \ \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee \ f \ ) \wedge \\ & (a \vee \ c \ \vee \ f \ ) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)$$

  - Assume decisions $c = \mathit{False}$, $f = \mathit{False}$, $h = \mathit{False}$ and $i = \mathit{False}$

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)$$

  - Assume decisions $c = \mathit{False}$, $f = \mathit{False}$, $h = \mathit{False}$ and $i = \mathit{False}$
  - Assignment $a = \mathit{False}$ caused conflict $\Rightarrow$ learnt clause $(a \vee c \vee f)$ implies a

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)$$

  - Assume decisions $c = False$, $f = False$, $h = False$ and $i = False$
  - Assignment $a = False$ caused conflict $\Rightarrow$ learnt clause $(a \vee c \vee f)$ implies a

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)$$

  - Assume decisions $c = False$, $f = False$, $h = False$ and $i = False$
  - Assignment $a = False$ caused conflict $\Rightarrow$ learnt clause $(a \vee c \vee f)$ implies a

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)$$

  - Assume decisions $c = False$, $f = False$, $h = False$ and $i = False$
  - Assignment $a = False$ caused conflict $\Rightarrow$ learnt clause $(a \vee c \vee f)$ implies a
  - A conflict is again reached : $(\neg d \vee \neg e \vee f)$ is unsatisfied

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)$$

  - Assume decisions $c = False$, $f = False$, $h = False$ and $i = False$
  - Assignment $a = False$ caused conflict $\Rightarrow$ learnt clause $(a \vee c \vee f)$ implies a
  - A conflict is again reached : $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $\varphi \wedge \neg c \wedge \neg f \Rightarrow \bot$

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)$$

  - Assume decisions $c = \mathit{False}$, $f = \mathit{False}$, $h = \mathit{False}$ and $i = \mathit{False}$
  - Assignment $a = \mathit{False}$ caused conflict $\Rightarrow$ learnt clause $(a \vee c \vee f)$ implies a
  - A conflict is again reached : $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $\varphi \wedge \neg c \wedge \neg f \Rightarrow \bot$
  - $\varphi \Rightarrow c \vee f$

Non-Chronological Backtracking

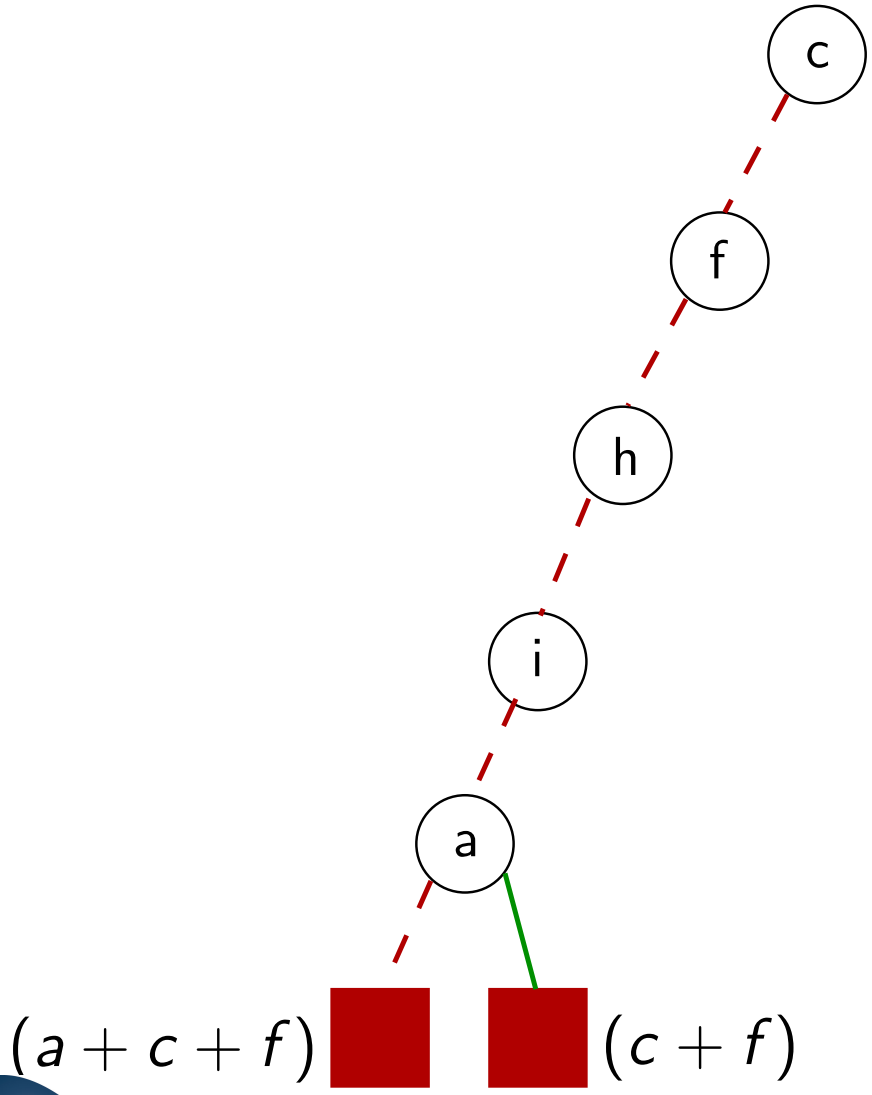
- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)$$

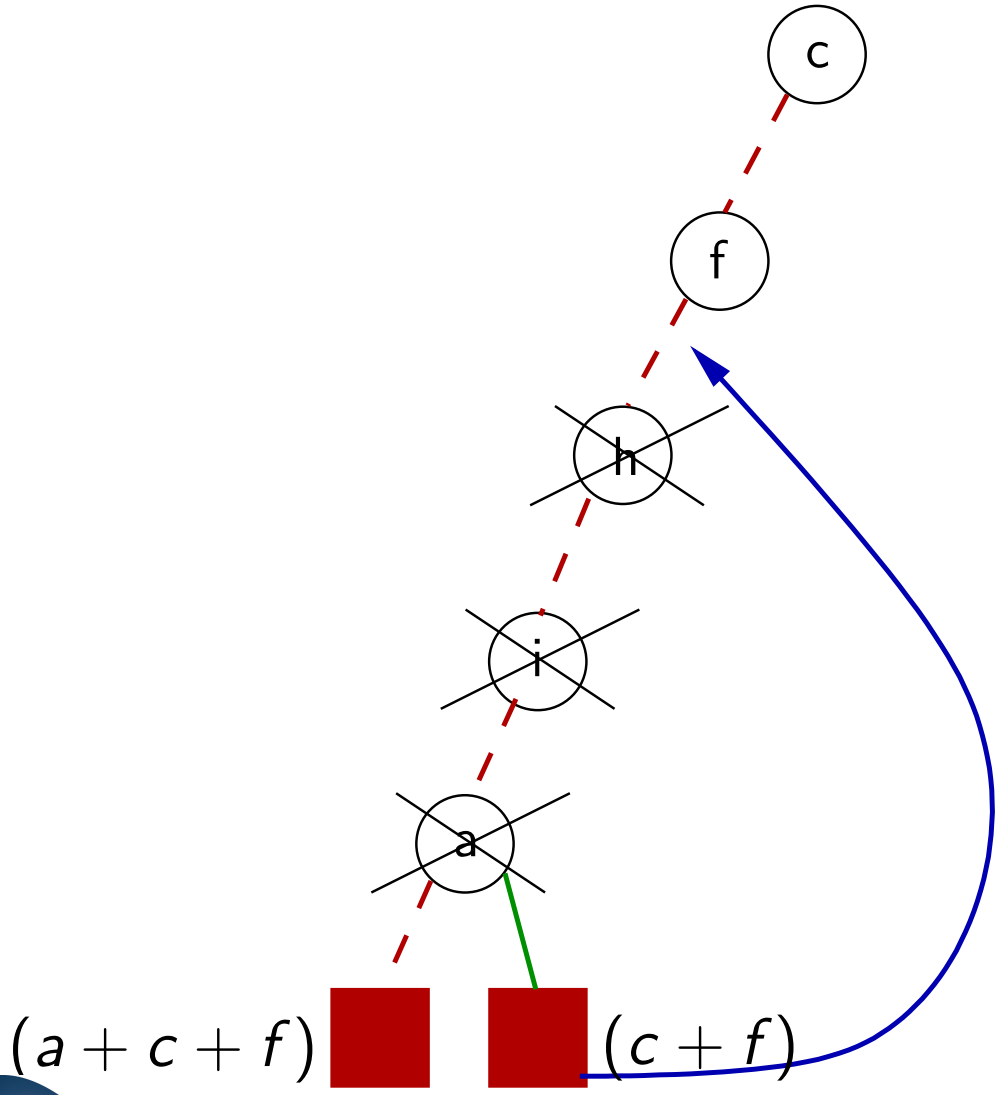
  - Assume decisions $c = False$, $f = False$, $h = False$ and $i = False$
  - Assignment $a = False$ caused conflict $\Rightarrow$ learnt clause $(a \vee c \vee f)$ implies a
  - A conflict is again reached : $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $\varphi \wedge \neg c \wedge \neg f \Rightarrow \bot$
  - $\varphi \Rightarrow c \vee f$

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

Non-Chronological Backtracking



Non-Chronological Backtracking



- Learnt clause : $(c \vee f)$
- Need to backtrack, given new clause
- Backtrack to most recent decision : $f = \mathit{False}$

- Clause learning and non-chronological backtracking are hallmarks of modern SAT solvers

How to implement NCB and Learning ? Resolution !

Perform resolution steps in reverse order of the assignments.
Propagations deriving from a : g,b,d, e

$$\begin{aligned}\varphi \quad = \quad & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\ & (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

Learned : $(a \vee c \vee f)$

$$(\neg d \vee \neg e \vee f)$$

How to implement NCB and Learning ? Resolution !

Perform resolution steps in reverse order of the assignments.
Propagations deriving from a : g,b, d ,e

$$\begin{aligned}\varphi \quad = \quad & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\ & (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

Learned : $(a \vee c \vee f)$

$$(\neg b \vee \neg d \vee f)$$

How to implement NCB and Learning ? Resolution !

Perform resolution steps in reverse order of the assignments.
Propagations deriving from a : g, b ,d,e

$$\begin{aligned}\varphi \quad = \quad & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\ & (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

Learned : $(a \vee c \vee f)$

$$(\neg b \vee c \vee f)$$

How to implement NCB and Learning ? Resolution !

Perform resolution steps in reverse order of the assignments.
Propagations deriving from a : g ,b,d,e

$$\begin{aligned}\varphi \quad = \quad & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\ & (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

Learned : $(a \vee c \vee f)$

$$(\neg g \vee c \vee f)$$

How to implement NCB and Learning ? Resolution !

Perform resolution steps in reverse order of the assignments.
Propagations deriving from a : g,b,d,e

$$\begin{aligned}\varphi \quad = \quad & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\ & (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

Learned : $(a \vee c \vee f)$

$$(\neg a \vee c \vee f)$$

How to implement NCB and Learning ? Resolution !

Perform resolution steps in reverse order of the assignments.
Propagations deriving from a : g,b,d,e

$$\begin{aligned}\varphi \quad = \quad & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\ & (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

Learned : $(a \vee c \vee f)$

$$(c \vee f)$$

Implementation of NCB and Learning for SAT

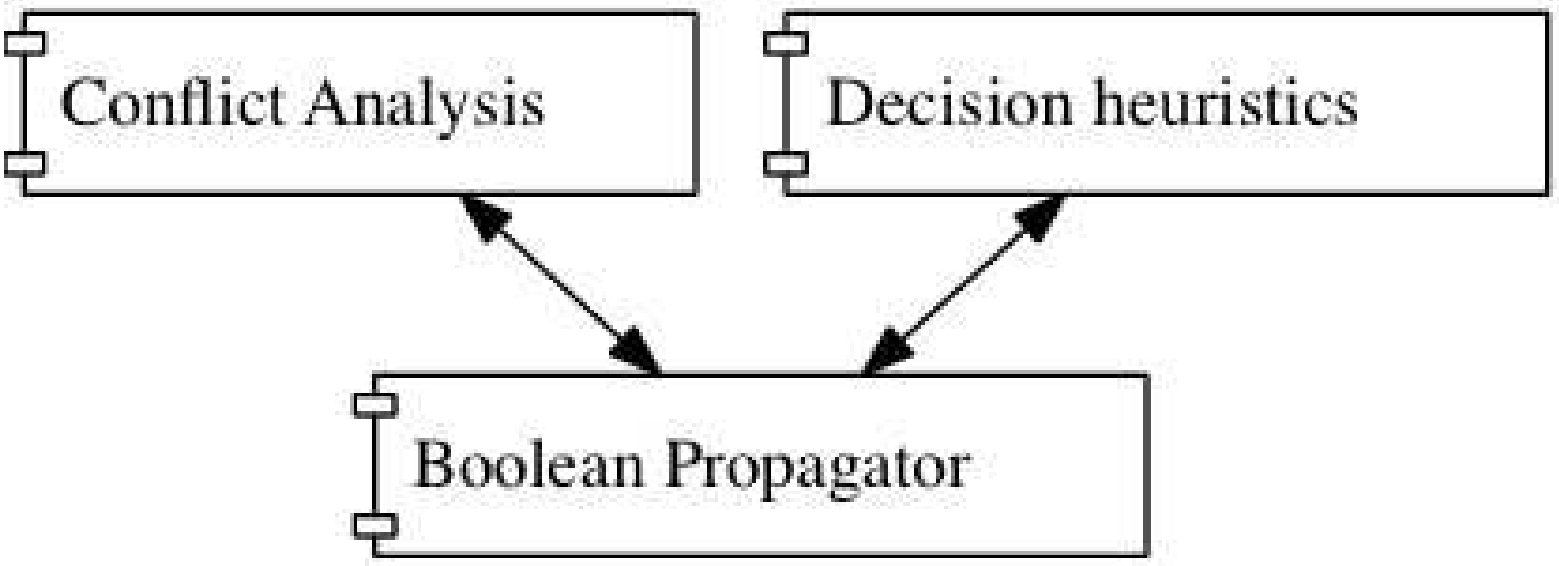
- Two approaches developed independently in two different research communities :
  GRASP/EDA by Marques-Silva and Sakallah (1996)
  - Resolution graph seen as a circuit
  - Conflict analysis thought as detecting faults in a circuit
  - Other sophisticated conflict analysis methods based on truth maintenance systems

  RELSAT/CSP by Bayardo and Schrag (1997)
  - Introduction of CSP based techniques into a SAT solver
  - Conflict Directed Backjumping aka non chronological backtracking [Prosser 93]
  - Size based and relevance based learning schemes
- Main difference : in GRASP's framework, the conflict analysis drives the search, while in RELSAT it is the heuristics (more later).

Agenda

GRASP architecture

João P. Marques Silva, Karem A. Sakallah : GRAPS : A Search Algorithm for Propositional Satisfiability. IEEE Trans. Computers 48(5) : 506-521 (1999)



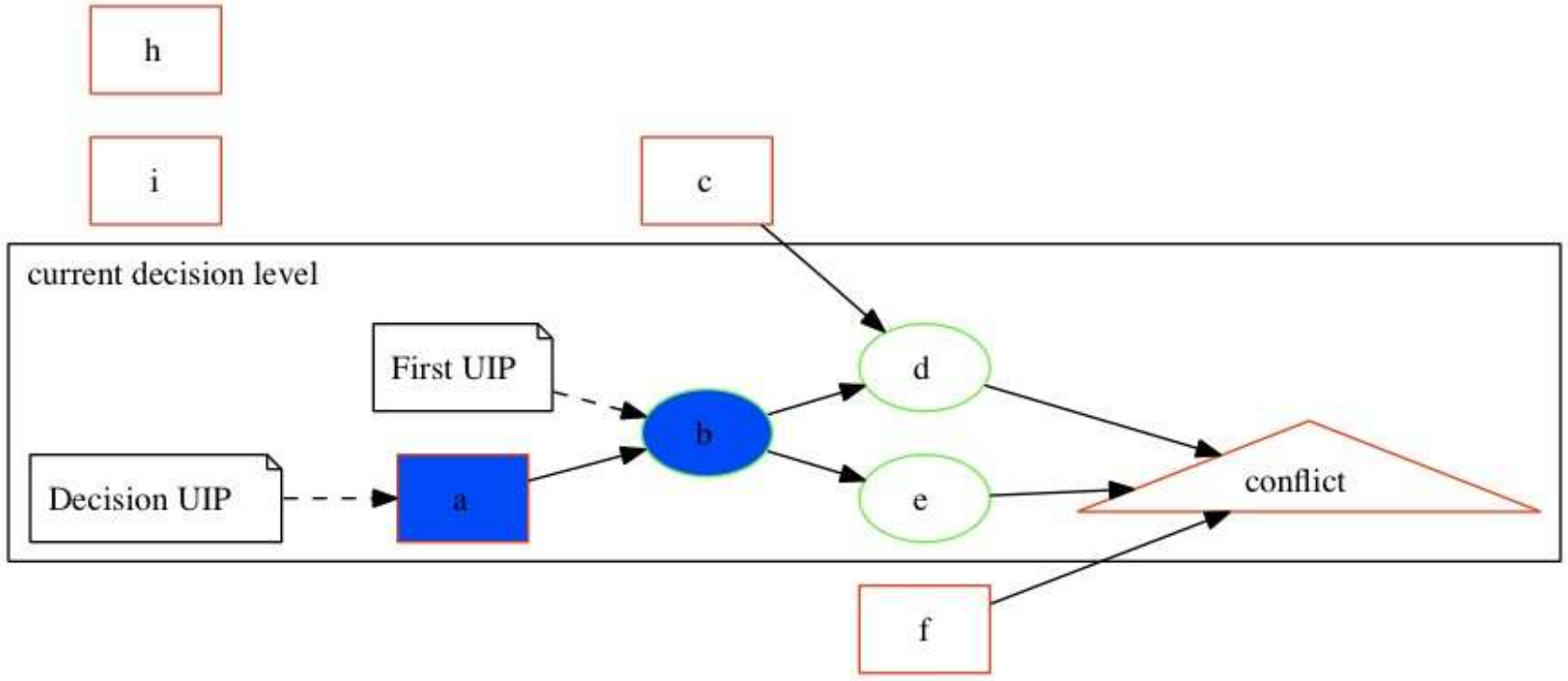
Role of the boolean propagator

- Perform unit propagation on the set of clauses.
- Detect conflicts
- Backtrack according to a specific clause provided by the conflict analyzer

Conflict analyzer

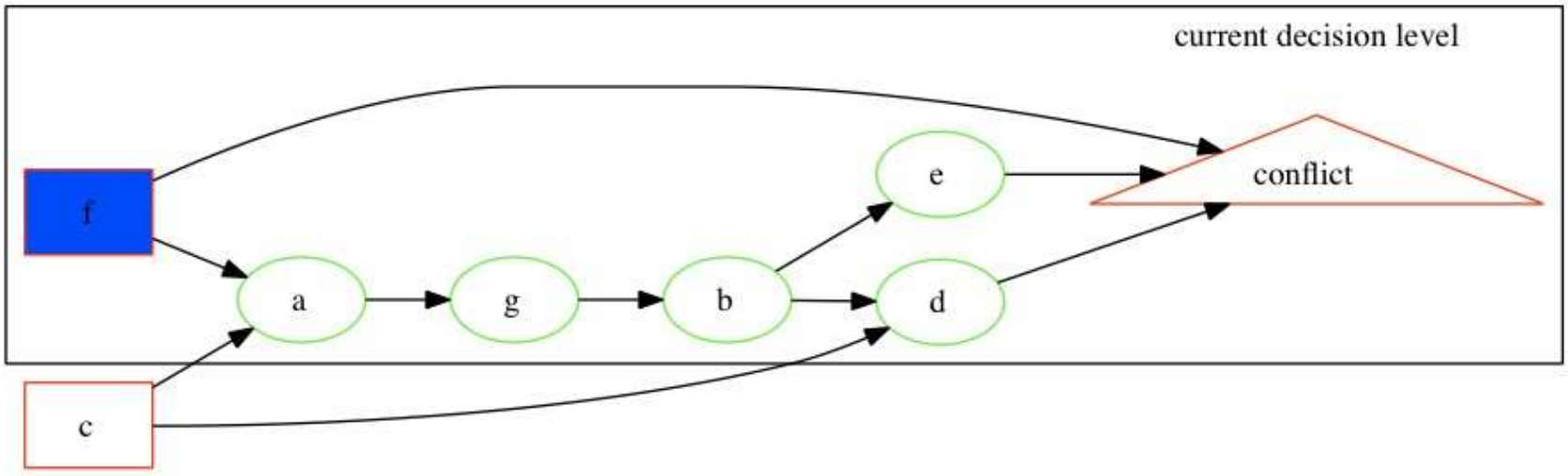
- Must produce a clause that becomes a unit clause after backtracking (asserting clause)
- Introduction of the notion of Unique Implication Point (UIP), as a reference to Unique Sensitization Points in ATPG.
  - Find a literal that need to be propagated before reaching a conflict
  - Based on the notion of decision level, i.e. the number of assumptions made so far.
  - Syntactical : apply resolution until only one literal from current decision level appears in the clause.
  - Decision variables are always UIP : at least one UIP exists for each conflict !
- Backtracking level computed as the lowest decision level of the literals of the clause

Conflict graph for assumption a=False



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

Conflict graph after learning $a \vee c \vee f$ and backjumping



$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge$$
$$(a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)$$

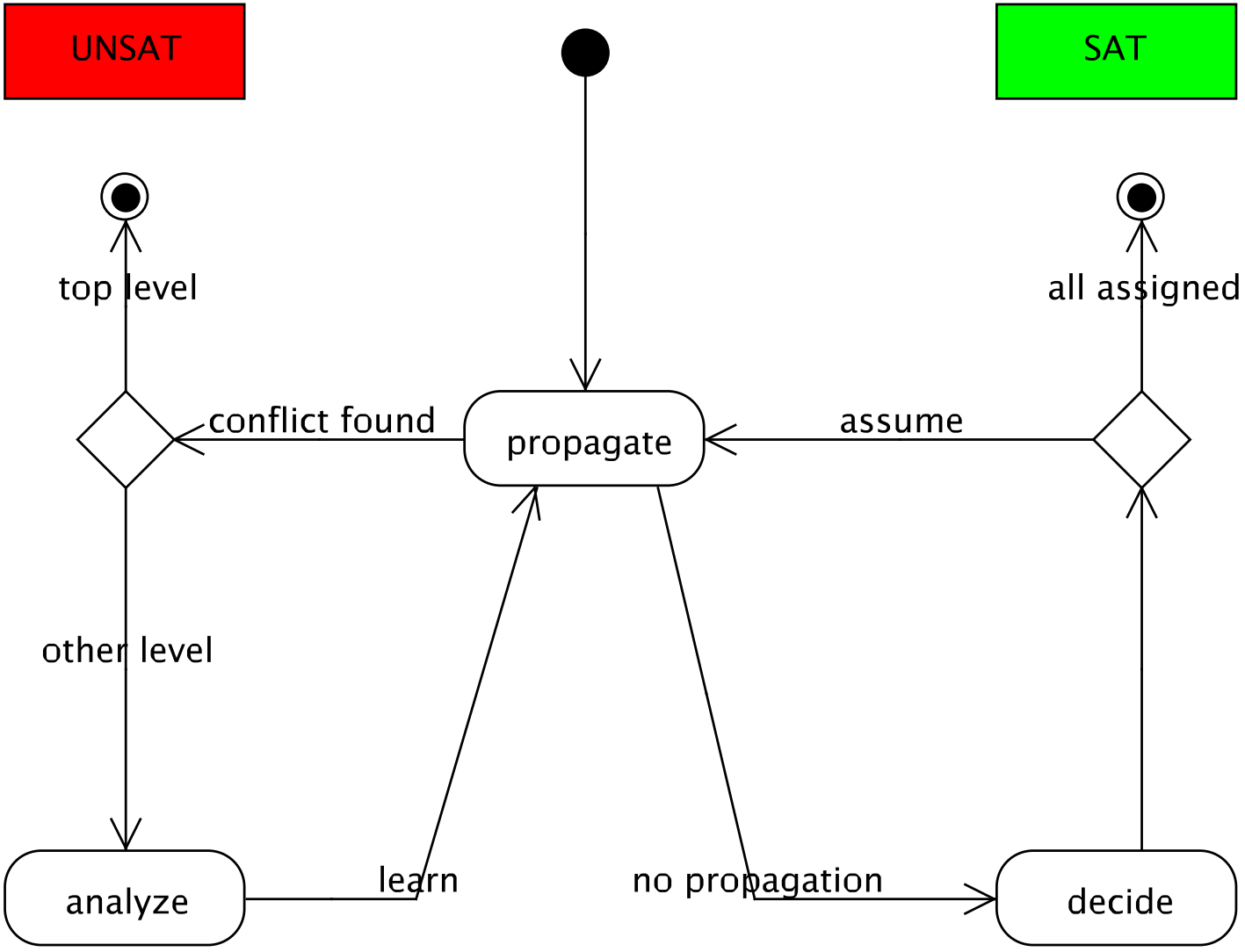
Some remarks about UIPs

- There are many possibilities to derive a clause using UIP
- RELSAT can be seen as applying Decision UIP
- 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 flip The search is thus driven by the conflict analysis.
- Generic name for GRASP approach : Conflict Driven Clause Learning (CDCL) solver [Ryan 2004].

Decision heuristics

- Pick an unassigned variable
- Many sophisticated decision heuristics available in the literature for random formulas (MOMS, JW, etc).
- GRASP uses dynamic largest individual sum (DLIS) : select the literal with the maximum occurrences in unresolved clauses.
- Sophisticated heuristics require an exact representation of the state of the CNF after unit propagation !

Putting everything together : the CDCL approach



Agenda

From GRASP to CHAFF

- Some key insights in the design of SAT solvers were discovered when trying to solve real problems by translation into SAT.
- Huge interest on SAT after the introduction of Bounded Model Checking [Biere et al 99] from the EDA community.
- The design of SAT solver becomes more pragmatic

Application 1 : Planning as satisfiability

Henry A. Kautz, Bart Selman : Planning as Satisfiability. ECAI 1992 : 359-363

- Input : a set of actions, an initial state and a goal state
- Output : a sequence of actions to reach the goal state from the initial state
- One of the first application of SAT in Artificial Intelligence
- A key application for the adoption of SAT in EDA later on
- The instances are supposed to be SAT
- Those instances are too big for complete solvers based on DPLL

1992 - Planning As Satisfiability

$$PAS(S, I, T, G, k) = I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} G(s_i)$$

où :

- S the set of possible states s_i
- I the initial state
- T transitions between states
- G goal state
- k bound

If the formula is satisfiable, then there is a plan of length at most k.

Greedy SAT (Local Search Scheme for SAT)

```
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;
}
```

Lessons learned from GSAT

- The decision procedure is very simple to implement and very fast !
- Efficiency depends on which literal to flip, and the values of the parameters.
- Problem with local minima : use of Random Walks !
- Main drawback : incomplete, cannot answer UNSAT !
- Lesson 1 : An agile (fast) SAT solver sometimes better than a clever one !

Application 2 : Quasigroup (Latin Square) open problems

- S a set and * a binary operator. $|S|$ is the order of the group.
- a*b=c has a unique solution when fixing any pair of variables.
- equivalent to fill in a $|S| \times |S|$ square with elements of S unique in each row and column.
- Looking for the existence of QG of a given order with additional constraints, e.g. :

$$\text{QG1}\quad x*y=u, z*w=u, v*y=x, v*w=z \Rightarrow x=z, y=w$$

$$\text{QG2}\quad x*y=u, z*w=u, y*v=x, w*v=z \Rightarrow x=z, y=w$$

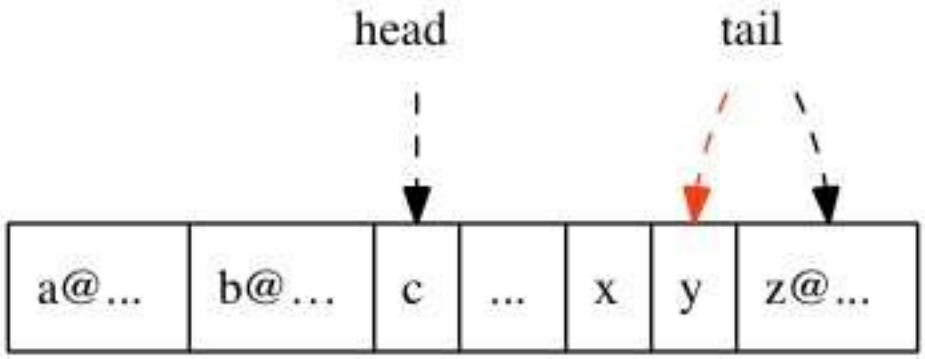
- First open QG problems solved by MGTP (Fujita, Slaney, Benett 93)
- QG2(12) solved by DDPP in 1993.
- QG1(12), QG2(14),QG2(15) solved by SATO in 1996.

SATO head/tail lazy data structure

Zhang, H., Stickel, M. : Implementing Davis-Putnam's method . It appeared as a Technical Report, The University of Iowa, 1994

- CNF resulting for QG problems have a huge amount of clauses : 10K to 150K !
- Encoding of real problems into SAT can lead to very large clauses
- Truth value propagation cost in eager data structure depends on the number of propagation to perform, thus on the size of the clauses
- How to limit the cost of numerous and long clauses during propagation ?
- Answer : use a lazy data structure to detect only unit propagation and falsified literals.

The Head/Tail data structure

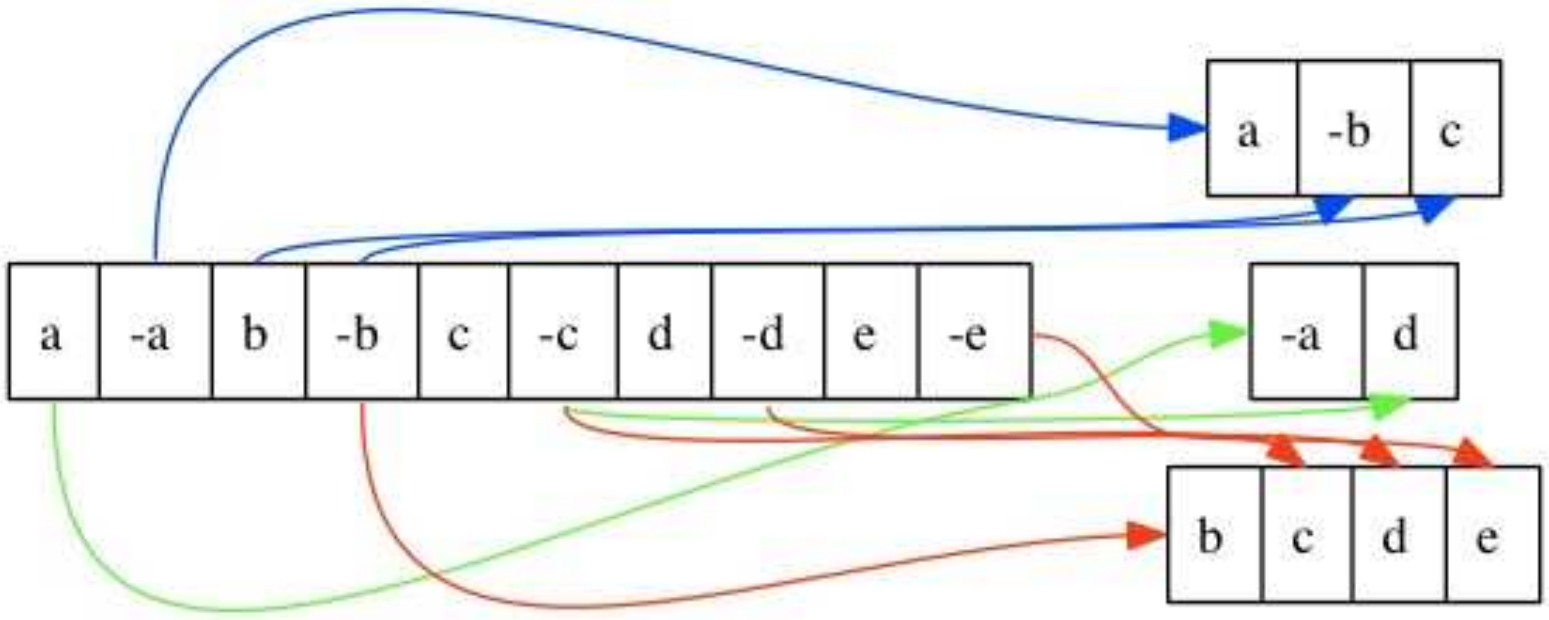


initially put a head (resp. tail) pointer to the first (resp. last) element of the clause

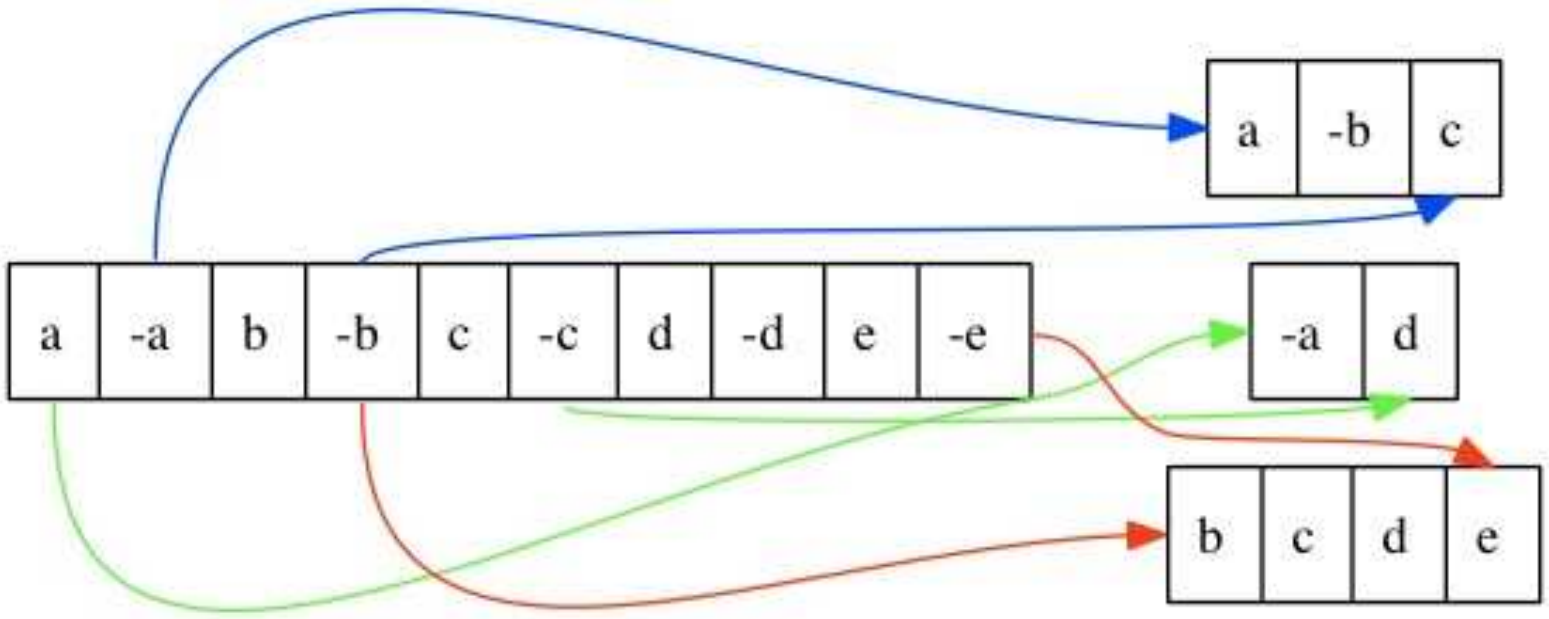
during propagation move heads or tails pointing to the negation of the propagated literal. Easy identification of unit and falsified clauses.

during backtracking move back the pointers to their previous location

Unit propagation with Adjacency lists



Unit propagation with Head /Tail



Pro and Cons of the H/T data structure

advantage reduces the cost of unit propagation

drawback the solver has no longer a complete picture of the reduced CNF !

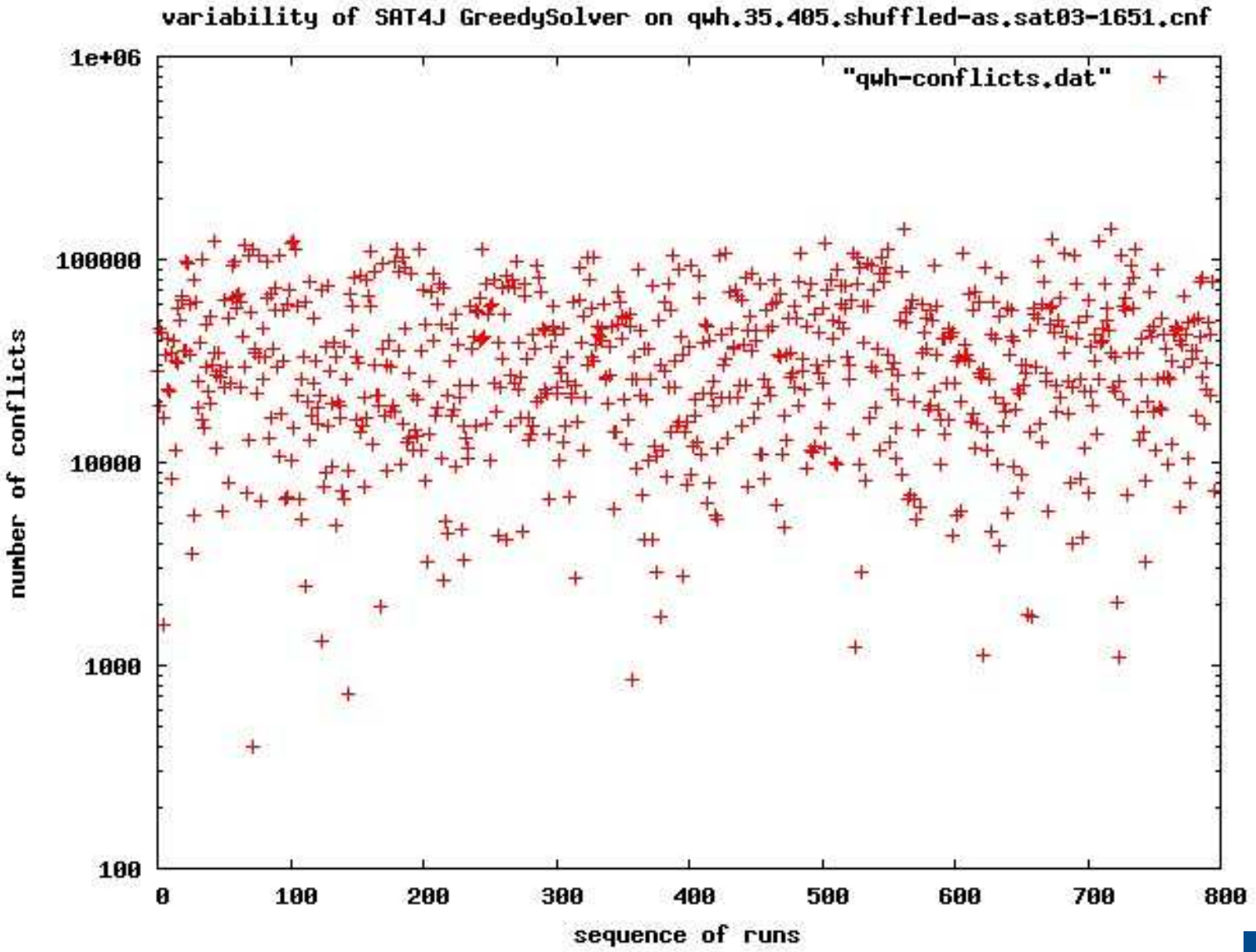
Lesson 2 : data structure matters !

High variability of SAT solvers runtime !

Heavy-tailed Distributions in Combinatorial Search. Carla Gomes, Bart Selman, and Nuno Crato. In Principles and Practices of Constraint Programming, (CP-97) Lecture Notes in Computer Science 1330, pp 121-135, Linz, Austria., 1997. Springer-Verlag

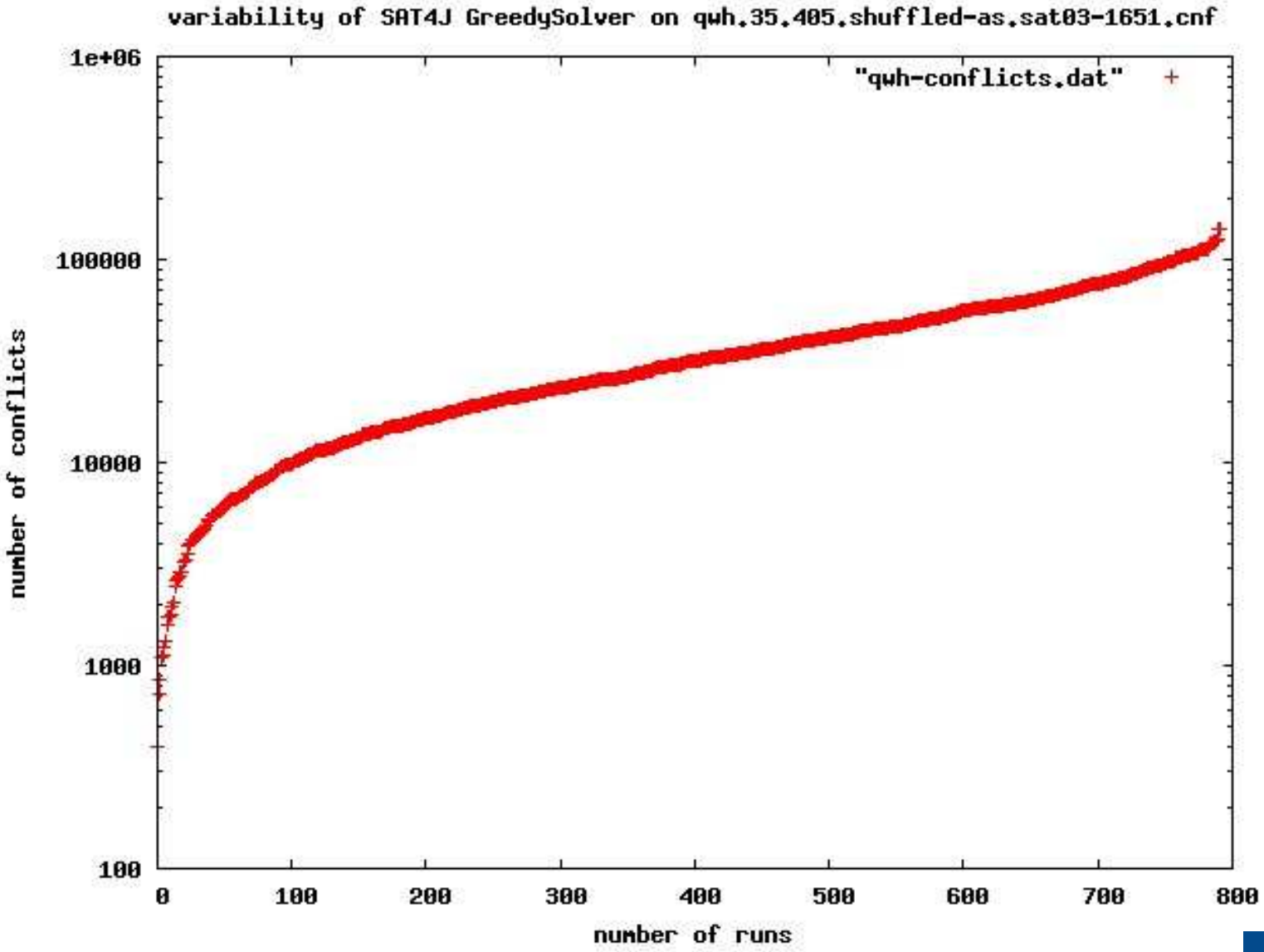
- SAT solvers exhibits on some problems a high runtime variability
- Decision heuristics need to break ties, often randomly
- The solver are sensible to syntactical input changes :
  - Shuffled industrial benchmarks harder than original ones for most solvers
  - The "lisa syndrome" during the SAT 2003 competition
- An explanation : Heavy tailed distribution

Example of variability : SAT4J GreedySolver on QGH

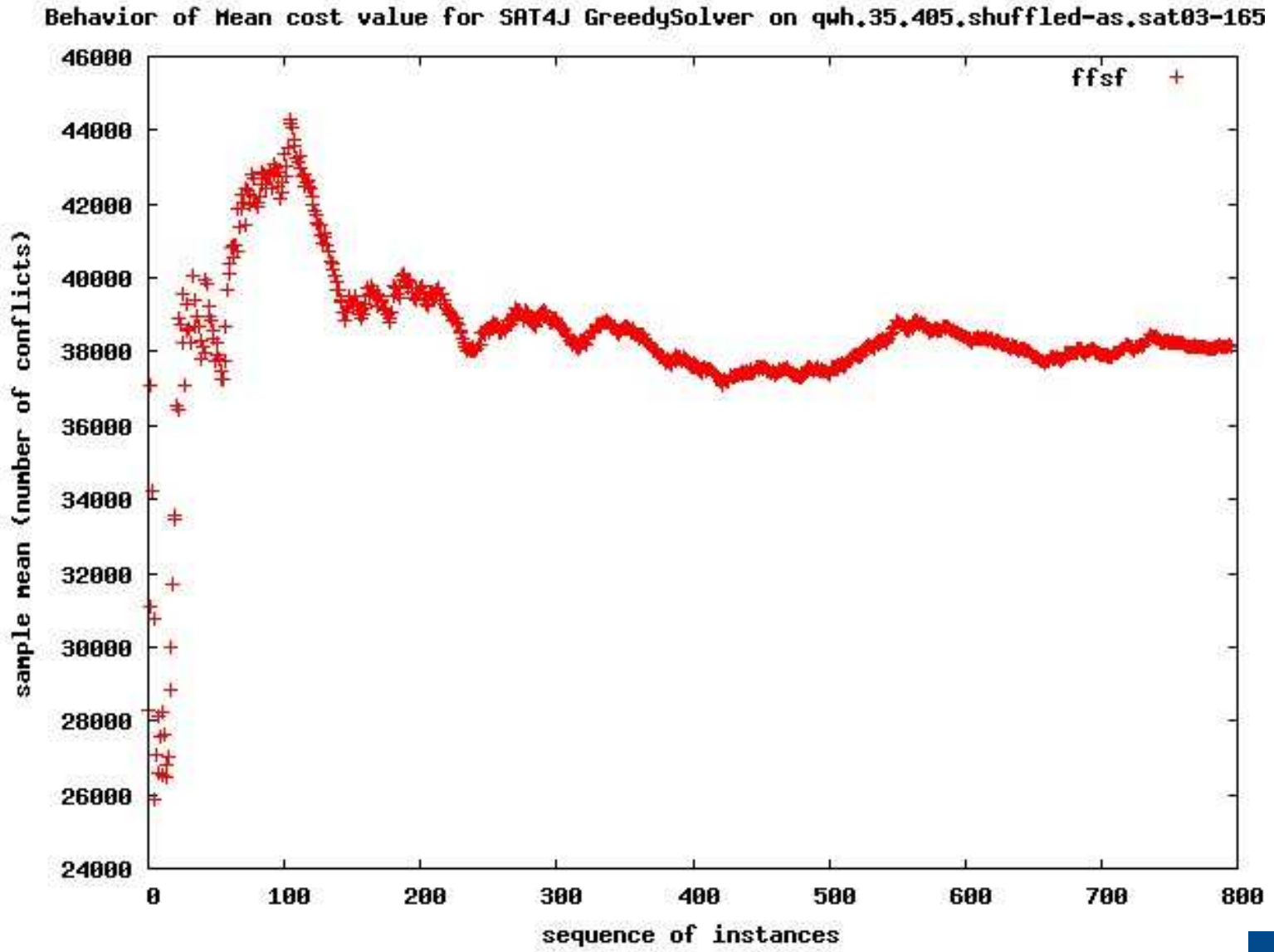


Verification Technology
Systems
Applications

Example of variability : SAT4J GreedySolver on QGH



Example of variability : SAT4J GreedySolver on QGH



Heavy Tailed distribution

- Introduced by the economist Pareto in the context of income distribution
- Widely used in many areas : stock market analysis, weather forecast, earthquake prediction, time delays on the WWW.
- Those distributions have infinite mean and infinite variance
- Some SAT solvers exhibit an Heavy Tailed distribution on Quasigroup Completion with Holes problems.
- What does it mean in practice ?
  - In rare occasion, the solver can get trapped on a very long run
  - while most of the time the run could be short
- the solution : restarts !

Restarting in SAT solvers

- Stop the search after a given number of conflicts/decisions/propagation is achieved (cutoff).
- Start again the search [with increased cutoff to be complete]
- Requires some variability in the solver behavior between two runs
- Problem : how to choose the cutoff value ?
- In theory, an optimal strategy exists [Luby 93].
- Lesson 3 : introduce restarts to make the solver more robusts

The killer app : Bounded Model Checking

A. Biere, A. Cimatti, E. Clarke, M. Fujita, Y. Zhu. Symbolic Model Checking using SAT procedures instead of BDDs. In Proc. ACM Design Automation Conf. (DAC'99), ACM 1999.

$$BMC(S, I, T, p, k) = I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} \neg p(s_i)$$

where :

- S the set of possible states s_i
- I the initial state
- T transitions between states
- p is an invariant property
- k a bound

If the formula is satisfiable, then there is a counter-example reachable within k steps.

SAT vs BDD model checking

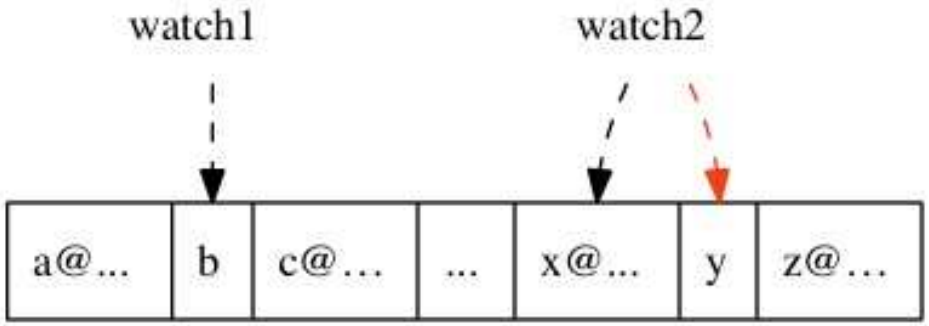
- Some model checking problems out of reach of BDD checkers can be solved thanks to a reduction to SAT
- The behavior of SAT solvers is less dependent of the form of the input than BDD solvers
- But the SAT solvers are not powerful enough yet for industrial use...

The breakthrough : Chaff

Chaff : Engineering an Efficient SAT Solver by M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik, 39th Design Automation Conference (DAC 2001), Las Vegas, June 2001.

- 2 order of magnitude speedup on unsat instances compared to existing approaches on BMC (Velev) benchmarks.
- Immediate speedup for SAT based tools : BlackBox "Supercharged with Chaff"
- Based on careful analysis of GRASP internals
- 3 key features :
  - New lazy data structure : Watched literals
  - New adaptative heuristic : Variable State Independent Decaying Sum
  - New conflict analysis approach : First UIP
- Taking into account randomization

The watched literals data structure



initially watch two arbitrary literals in the clause

during propagation move watchers pointers in clauses containing the negation of the propagated literal.

during backtracking do nothing !

advantage cost free data structure when backtracking

issue pointers can move in both directions.

Variable State Independent Decaying Sum

- compatible with Lazy Data Structures
- each literal has a score
- score based on the number of occurrences of the literals in the formula
- score updated whenever a new clause is learned
- pick the unassigned literal with the highest score, tie broken randomly
- regularly (every 256 conflicts), divided the scores by a constant (2)

New Learning Scheme : First UIP

Efficient Conflict Driven Learning in a Boolean Satisfiability Solver by L. Zhang, C. Madigan, M. Moskewicz, S. Malik, Proceedings of ICCAD 2001, San Jose, CA, Nov. 2001

- The idea is to quickly compute a reason for the conflict
- Stop the resolution process as soon as an UIP is detected
- First UIP Shown to be optimal in terms of backtrack level compared to the other possible UIPs [Audemard et al 08].

Chaff : a highly coupled set of features

- Learning does not degrade solver performance because the use of the watched literals
- The VSIDS heuristics does not need a complete picture of the reduced formula, i.e. is compatible with the lazy data structure.
- VSIDS take advantage of the conflict analysis to spot important literals.
- VSIDS provides different orders of literals at each restart
- VSIDS adapt itself to the instance !

The reason of the success ?

- Better engineering (level 2 cache awareness) ?

The reason of the success ?

- Better engineering (level 2 cache awareness) ?
- Better tradeoff between speed and intelligence ?

The reason of the success ?

- Better engineering (level 2 cache awareness) ?
- Better tradeoff between speed and intelligence ?
- Instance-based auto adaptation ?

The reason of the success ?

- Better engineering (level 2 cache awareness) ?
- Better tradeoff between speed and intelligence ?
- Instance-based auto adaptation ?
- ...

The reason of the success ?

- Better engineering (level 2 cache awareness) ?
- Better tradeoff between speed and intelligence ?
- Instance-based auto adaptation ?
- ...

All those reasons are correct. There is a more fundamental reason too ...

CDCL has a better proof system than DPLL !

Proof theory strikes back !

- ... thanks to many others before ...
- Bonet, M. L., & Galesi, N. (2001). Optimality of size-width tradeoffs for resolution. Computational Complexity, 10(4), 261-276.
- Beame, P., Kautz, H., and Sabharwal, A. Towards understanding and harnessing the potential of clause learning. JAIR 22 (2004), 319-351.
- Van Gelder, A. Pool resolution and its relation to regular resolution and dpll with clause learning. In LPAR'05 (2005), pp. 580-594.
- Hertel, P., Bacchus, F., Pitassi, T., and Van Gelder, A. Clause learning can effectively p-simulate general propositional resolution. In Proc. of AAAI-08 (2008), pp. 283-290.
- Knot Pipatsrisawat, Adnan Darwiche : On the Power of Clause-Learning SAT Solvers with Restarts. CP 2009 : 654-668

CDCL has a better proof system than DPLL !

Proof theory strikes back !

Definition

p-simulation Proof system S p-simulates proof system T , if, for every unsatisfiable formula φ, the shortest refutation proof of φ in S is at most polynomially longer than the shortest refutation proof of φ in T.

CDCL has a better proof system than DPLL !

Proof theory strikes back !

Definition

p-simulation Proof system S p-simulates proof system T , if, for every unsatisfiable formula φ, the shortest refutation proof of φ in S is at most polynomially longer than the shortest refutation proof of φ in T.

Theorem 1 [Pipatsrisawat, Darwiche 09]. CLR with any asserting learning scheme p-simulates general resolution.

Since Chaff ...

- The international SAT competition/SAT race is organized every year
- A huge number of CDCL solvers have been developed, and made available to the community
- SAT has integrated the engineer toolbox to solve combinatorial problems
- Many papers published on the design of efficient SAT solvers

Since Chaff ...

- The international SAT competition/SAT race is organized every year
- A huge number of CDCL solvers have been developed, and made available to the community
- SAT has integrated the engineer toolbox to solve combinatorial problems
- Many papers published on the design of efficient SAT solvers
- ... but a big part of the knowledge still lies in source code !

Agenda

Minisat : the minimalist CDCL SAT solver

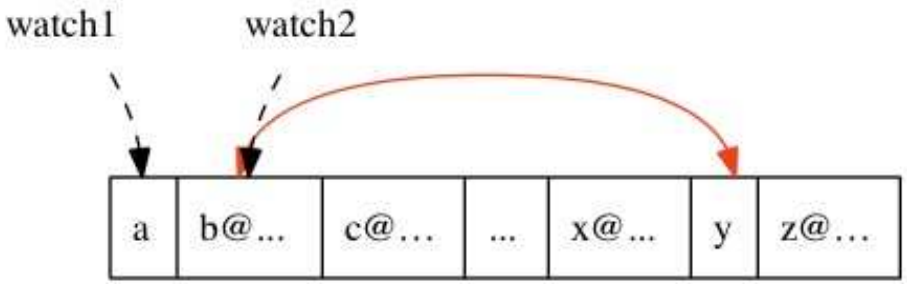
`http://www.minisat.se`, Niklas Eén, Niklas Sörensson : An Extensible SAT-solver. SAT 2003 : 502-518

- very simple implementation of a Chaff-like solver
- resulting from the lessons learned from designing Satzoo (SAT 2003 Winner) and SATnick
- with implementation improvements (Watched Literals, Heuristics, Priority Queue (2005), etc.)
- ready for generic constraints (cardinality, linear pseudo boolean, etc.).
- published description of the design

Reduced the entry level required to experiment with CDCL SAT solvers

The watched literals data structure improved

[mChaff,vanGelder02,Minisat]



initially watch the two first literals in the clause

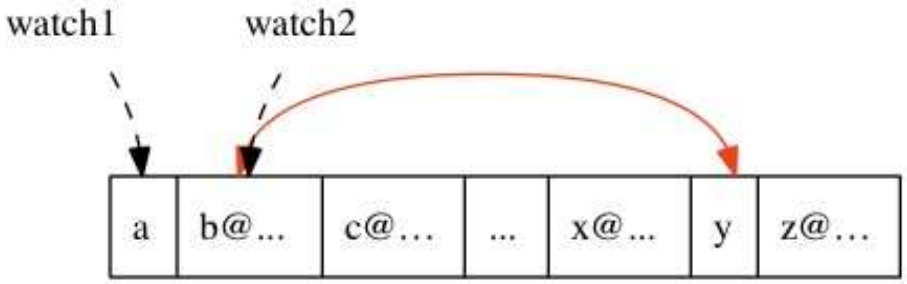
during propagation move falsified literal in second position. Exchange it with an unassigned literal is any. Easy identification of unit and falsified clauses.

during backtracking do nothing !

advantage cost free data structure when backtracking

The watched literals data structure improved

[mChaff,vanGelder02,Minisat]



initially watch the two first literals in the clause

during propagation move falsified literal in second position. Exchange it with an unassigned literal is any. Easy identification of unit and falsified clauses.

during backtracking do nothing !

advantage cost free data structure when backtracking

Moving literals instead of pointers in HT data structure also provides cost free backtracking !

Berkmin style heuristic

Evguenii I. Goldberg, Yakov Novikov : BerkMin : A Fast and Robust Sat-Solver. DATE 2002 : 142-149

Ideas :

- force the heuristic to satisfy recently learned clauses to be more reactive than VSIDS
- sophisticated phase selection strategy based on an estimate of the unit propagations to result from the selection (a la SATZ [Li Anbulagan 97]).
- take into account literals met during the conflict analysis

Berkmin performed quite well during SAT 2002 (despite a stupid bug) and it's successor Forklift won in 2003.