Boolean Satisfiability Solving
Past, Present & Future

Joao Marques-Silva

School of Computer Science & Informatics
University College Dublin

July 2010
The Success of SAT

<table>
<thead>
<tr>
<th>Years</th>
<th># Variables</th>
<th># Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>60s-70s</td>
<td>few tens</td>
<td>hundreds</td>
</tr>
<tr>
<td>80s - early 90s</td>
<td>few hundreds</td>
<td>few thousands</td>
</tr>
<tr>
<td>late 90s</td>
<td>(tens of) thousands</td>
<td>hundreds of thousands</td>
</tr>
<tr>
<td>00s</td>
<td>hundreds of thousands and more</td>
<td>millions</td>
</tr>
</tbody>
</table>
### The Success of SAT

<table>
<thead>
<tr>
<th>Years</th>
<th># Variables</th>
<th># Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>60s-70s</td>
<td>few tens</td>
<td>hundreds</td>
</tr>
<tr>
<td>80s - early 90s</td>
<td>few hundreds</td>
<td>few thousands</td>
</tr>
<tr>
<td>late 90s</td>
<td>(tens of) thousands</td>
<td>hundreds of thousands</td>
</tr>
<tr>
<td>00s</td>
<td>hundreds of thousands and more</td>
<td>millions</td>
</tr>
</tbody>
</table>

- **In this talk:**
  - Organization of modern SAT solvers
    - Why do SAT solvers work in practice?
## The Success of SAT

<table>
<thead>
<tr>
<th>Years</th>
<th># Variables</th>
<th># Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>60s-70s</td>
<td>few tens</td>
<td>hundreds</td>
</tr>
<tr>
<td>80s - early 90s</td>
<td>few hundreds</td>
<td>few thousands</td>
</tr>
<tr>
<td>late 90s</td>
<td>(tens of) thousands</td>
<td>hundreds of thousands</td>
</tr>
<tr>
<td>00s</td>
<td>hundreds of thousands and more</td>
<td>millions</td>
</tr>
</tbody>
</table>

### In this talk:
- Organization of modern SAT solvers
  - Why do SAT solvers work in practice?
- Tentative glimpse of the future
Outline

Preliminaries

The (Recent) Past
   The DPLL Algorithm

The Present
   Conflict-Driven Clause Learning (CDCL)
   Why Does It Work?

The (Near) Future

Conclusions
Outline

Preliminaries

The (Recent) Past
The DPLL Algorithm

The Present
Conflict-Driven Clause Learning (CDCL)
Why Does It Work?

The (Near) Future

Conclusions
Boolean Satisfiability (SAT)

- Boolean formula $\varphi$ is defined over a set of propositional variables $x_1, \ldots, x_n$, using the standard propositional connectives $\neg$, $\land$, $\lor$, $\rightarrow$, $\leftrightarrow$, and parenthesis
  - The domain of propositional variables is $\{0, 1\}$
  - Example: $\varphi(x_1, \ldots, x_3) = ((\neg x_1 \land x_2) \lor x_3) \land (\neg x_2 \lor x_3)$

- A formula $\varphi$ in conjunctive normal form (CNF) is a conjunction of disjunctions (clauses) of literals, where a literal is a variable or its complement
  - Example: $\varphi(x_1, \ldots, x_3) = (\neg x_1 \lor x_2) \land (\neg x_2 \lor x_3)$

- Can encode any Boolean formula into CNF

[Tseitin’68]
Boolean Satisfiability (SAT)

- Boolean formula $\varphi$ is defined over a set of propositional variables $x_1, \ldots, x_n$, using the standard propositional connectives $\neg$, $\land$, $\lor$, $\rightarrow$, $\leftrightarrow$, and parenthesis
  - The domain of propositional variables is $\{0, 1\}$
  - Example: $\varphi(x_1, \ldots, x_3) = ((\neg x_1 \land x_2) \lor x_3) \land (\neg x_2 \lor x_3)$

- A formula $\varphi$ in conjunctive normal form (CNF) is a conjunction of disjunctions (clauses) of literals, where a literal is a variable or its complement
  - Example: $\varphi(x_1, \ldots, x_3) = (\neg x_1 \lor x_2) \land (\neg x_2 \lor x_3)$

- Can encode any Boolean formula into CNF

- The Boolean satisfiability (SAT) problem:
  - Find an assignment to the variables $x_1, \ldots, x_n$ such that $\varphi(x_1, \ldots, x_n) = 1$, or prove that no such assignment exists
Boolean Satisfiability (SAT)

- In theory: NP-complete  

[Cook’71]
Boolean Satisfiability (SAT)

- In theory: NP-complete
- In practice: success story of Computer Science

[Cook’71]
Boolean Satisfiability (SAT)

- In theory: NP-complete
- In practice: success story of Computer Science
  - Remarkable improvements since the mid 90s: Clause learning; UIPs; Search restarts; Lazy data structures; Adaptive branching heuristics; Clause minimization; Preprocessing; etc.
Boolean Satisfiability (SAT)

- In theory: NP-complete
- In practice: success story of Computer Science
  - Remarkable improvements since the mid 90s: Clause learning; UIPs; Search restarts; Lazy data structures; Adaptive branching heuristics; Clause minimization; Preprocessing; etc.
  - Hundreds (thousands?) of practical applications: Hardware model checking; Software model checking; Termination analysis of term-rewrite systems; Test pattern generation (testing of software & hardware); Model finding; Symbolic trajectory evaluation; Planning; Knowledge representation; Games (n-queens, sudoku, etc.); Haplotype inference; Pedigree checking; Equivalence checking; Delay computation; Fault diagnosis; Digital filter design; Noise analysis; Cryptanalysis; Inversion attacks on hash functions; Graph coloring; Traveling salesperson; van der Waerden numbers; (your favourite SAT application here!); etc.; etc.
Boolean Satisfiability (SAT)

- In theory: NP-complete
- In practice: success story of Computer Science
  - Remarkable improvements since the mid 90s: Clause learning; UIPs; Search restarts; Lazy data structures; Adaptive branching heuristics; Clause minimization; Preprocessing; etc.
  - Hundreds (thousands?) of practical applications: Hardware model checking; Software model checking; Termination analysis of term-rewrite systems; Test pattern generation (testing of software & hardware); Model finding; Symbolic trajectory evaluation; Planning; Knowledge representation; Games (n-queens, sudoku, etc.); Haplotype inference; Pedigree checking; Equivalence checking; Delay computation; Fault diagnosis; Digital filter design; Noise analysis; Cryptanalysis; Inversion attacks on hash functions; Graph coloring; Traveling salesperson; van der Waerden numbers; (your favourite SAT application here!); etc.; etc.
  - Core engine for other solvers: Pseudo-Boolean; MaxSAT, QBF; #SAT; ASP; SMT; MVL; CSP; etc.
  - Integrated into theorem provers: HOL; Isabelle; etc.
Basic Definitions

- Propositional variables can be assigned value 0 or 1
  - In some contexts variables may be unassigned

- A clause is satisfied if at least one of its literals is assigned value 1
  \(( x_1 \lor \neg x_2 \lor \neg x_3 )\)

- A clause is unsatisfied if all of its literals are assigned value 0
  \(( x_1 \lor \neg x_2 \lor \neg x_3 )\)

- A clause is unit if it contains one single unassigned literal and all other literals are assigned value 0
  \(( x_1 \lor \neg x_2 \lor \neg x_3 )\)

- A formula is satisfied if all of its clauses are satisfied
- A formula is unsatisfied if at least one of its clauses is unsatisfied
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal **must be assigned value 1** for the clause to be satisfied
  
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) **must** be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule
  
  \((x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)\)
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule
  \[
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
  \]
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
\]
Unit Propagation

• **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  
  – Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

• **Unit propagation**
  Iterated application of the unit clause rule

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
\]

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor \neg x_4)
\]
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied

  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
\]

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor \neg x_4)
\]
Unit Propagation

- **Unit clause rule**: Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied.
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0.

- **Unit propagation**: Iterated application of the unit clause rule.

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
\]

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor \neg x_4)
\]

[Davis & Putnam, JACM'60]
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \( x_1 \lor \neg x_2 \lor \neg x_3 \), \( x_3 \) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule
  \[
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
  \]
  \[
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor \neg x_4)
  \]

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

• Resolution rule:
  – If a formula $\varphi$ contains clauses $(x \lor \alpha)$ and $(\neg x \lor \beta)$, then infer $(\alpha \lor \beta)$

$$\text{RES}(x \lor \alpha, \neg x \lor \beta) = (\alpha \lor \beta)$$

• Resolution forms the basis of a complete algorithm for SAT
  – Iteratively apply the following steps: [Davis&Putnam, JACM’60]
    ▶ Select variable $x$
    ▶ Apply resolution rule between every pair of clauses of the form $(x \lor \alpha)$ and $(\neg x \lor \beta)$
    ▶ Remove all clauses containing either $x$ or $\neg x$
    ▶ Apply the pure literal rule and unit propagation
  – Terminate when either the empty clause or the empty formula is derived
Outline

Preliminaries

The (Recent) Past
   The DPLL Algorithm

The Present
   Conflict-Driven Clause Learning (CDCL)
   Why Does It Work?

The (Near) Future

Conclusions
Historical Perspective

- In 1960, M. Davis and H. Putnam proposed the DP algorithm:
  - Resolution used to eliminate 1 variable at each step
  - Applied the pure literal rule and unit propagation

- Original algorithm was inefficient

- In 1962, M. Davis, G. Logemann and D. Loveland proposed an alternative algorithm:
  - Instead of eliminating variables, the algorithm would split on a given variable at each step
  - Also applied the pure literal rule and unit propagation

- The 1962 algorithm is actually an implementation of backtrack search

- Over the years the 1962 algorithm became known as the DPLL (sometimes DLL) algorithm
Basic Algorithm for SAT – DPLL

• Standard **backtrack search**
• At each step:
  – **[DECIDE]** Select decision assignment
  – **[DEDUCE]** Apply unit propagation and (optionally) the pure literal rule
  – **[DIAGNOSE]** If conflict identified, then backtrack
    ▶ If cannot backtrack further, return **UNSAT**
    ▶ Otherwise, proceed with unit propagation
  – If formula satisfied, return **SAT**
  – Otherwise, proceed with another decision
\( \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \)
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \land \\
\text{conflict} \]
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land (\neg b \lor \neg d \lor \neg e) \land (a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land (a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land (\neg b \lor \neg d \lor \neg e) \land (a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land (a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
An Example of DPLL

\[
\phi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e)
\]
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land (\neg b \lor \neg d \lor \neg e) \land (a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land (a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
Outline

Preliminaries

The (Recent) Past
   The DPLL Algorithm

The Present
   Conflict-Driven Clause Learning (CDCL)
   Why Does It Work?

The (Near) Future

Conclusions
CDCL SAT Solvers – Basic Techniques

- Based on DPLL
  - Must be able to prove unsatisfiability

- New clauses are **learned** from conflicts
  - Backtracking can be **non-chronological**

- Structure of conflicts is exploited (**UIPs**)

- Backtrack search is periodically **restart**

- Lazy data structures are used
  - Compact with low maintenance overhead

- Branching is guided by conflicts
  - E.g. VSIDS, etc.
CDCL SAT Solvers – Additional Techniques

• (Currently) **effective** techniques:
  – Unused learned clauses are discarded
  – Use formula preprocessing I
  – Minimize learned clauses
  – Use literal progress saving
  – Use dynamic restart policies
  – Exploit extended implication graphs
  – Identify glue clauses

• (Currently) **ineffective** techniques:
  – Identify pure literals
  – Implement variable lookahead
  – Use formula preprocessing II

[Goldberg & Novikov, DATE’02]
[Goldberg & Novikov, DATE’02]
[Een & Biere, SAT’05]
[Sorensson & Biere, SAT’09]
[Pipatsrisawat & Darwiche, SAT’07]
[Biere, SAT’08]
[Audemard et al., SAT’08]
[Audemard & Simon, IJCAI’09]
[Davis & Putnam, JACM’60]
[Anbulagan & Li, IJCAI’97]
[Brafman, IJCAI’01]
Clause Learning

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots \]
Clause Learning

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots \]

- Assume decisions \( c = 0 \) and \( f = 0 \)
Clause Learning

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots \]

- Assume decisions \( c = 0 \) and \( f = 0 \)
- Assign \( a = 0 \) and imply assignments
Clause Learning

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots \]

- Assume decisions \( c = 0 \) and \( f = 0 \)
- Assign \( a = 0 \) and imply assignments
Clause Learning

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots \]

- Assume decisions \( c = 0 \) and \( f = 0 \)
- Assign \( a = 0 \) and imply assignments
- A conflict is reached: \( (\neg d \lor \neg e \lor f) \) is unsatisfied
Clause Learning

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots \]

- Assume decisions \( c = 0 \) and \( f = 0 \)
- Assign \( a = 0 \) and imply assignments
- A conflict is reached: \( (\neg d \lor \neg e \lor f) \) is unsatisfied
- \( (a = 0) \land (c = 0) \land (f = 0) \Rightarrow (\varphi = 0) \)
Clause Learning

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

\[ \varphi = (a \lor b) \land (\neg b \lor \text{c} \lor \text{d}) \land (\neg b \lor \text{e}) \land (\neg \text{d} \lor \neg \text{e} \lor \text{f}) \ldots \]

- Assume decisions \( c = 0 \) and \( f = 0 \)
- Assign \( a = 0 \) and imply assignments
- A conflict is reached: \( (\neg \text{d} \lor \neg \text{e} \lor \text{f}) \) is unsatisfied
- \( (a = 0) \land (c = 0) \land (f = 0) \Rightarrow (\varphi = 0) \)
- \( (\varphi = 1) \Rightarrow (a = 1) \lor (c = 1) \lor (f = 1) \)
Clause Learning

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots \]

- Assume decisions \( c = 0 \) and \( f = 0 \)
- Assign \( a = 0 \) and imply assignments
- A conflict is reached: \( (\neg d \lor \neg e \lor f) \) is unsatisfied
- \( (a = 0) \land (c = 0) \land (f = 0) \Rightarrow (\varphi = 0) \)
- \( (\varphi = 1) \Rightarrow (a = 1) \lor (c = 1) \lor (f = 1) \)
- Learn new clause \( (a \lor c \lor f) \)
Non-Chronological Backtracking

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]
Non-Chronological Backtracking

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

\[
\varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k)
\]

  - Assume decisions \( c = 0 \), \( f = 0 \), \( h = 0 \) and \( i = 0 \)
Non-Chronological Backtracking

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]

- Assume decisions \( c = 0 \), \( f = 0 \), \( h = 0 \) and \( i = 0 \)
- Assignment \( a = 0 \) caused conflict \( \Rightarrow \) learnt clause \((a \lor c \lor f)\) implies \( a = 1 \)
Non-Chronological Backtracking

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]

- Assume decisions \( c = 0, f = 0, h = 0 \) and \( i = 0 \)
- Assignment \( a = 0 \) caused conflict \( \Rightarrow \) learnt clause \((a \lor c \lor f)\) implies \( a = 1 \)
Non-Chronological Backtracking

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]

- Assume decisions \( c = 0, f = 0, h = 0 \) and \( i = 0 \)
- Assignment \( a = 0 \) caused conflict \( \Rightarrow \) learnt clause \((a \lor c \lor f)\) implies \( a = 1 \)
Non-Chronological Backtracking

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]

- Assume decisions \( c = 0, f = 0, h = 0 \) and \( i = 0 \)
- Assignment \( a = 0 \) caused conflict \( \Rightarrow \) learnt clause \((a \lor c \lor f)\) implies \( a = 1 \)
- A conflict is again reached: \((\neg d \lor \neg e \lor f)\) is unsatisfied
Non-Chronological Backtracking

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

\[
\varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k)
\]

- Assume decisions \(c = 0\), \(f = 0\), \(h = 0\) and \(i = 0\)
- Assignment \(a = 0\) caused conflict \(\Rightarrow\) learnt clause \((a \lor c \lor f)\) implies \(a = 1\)
- A conflict is again reached: \((\neg d \lor \neg e \lor f)\) is unsatisfied
- \((c = 0) \land (f = 0) \Rightarrow (\varphi = 0)\)
Non-Chronological Backtracking

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]

- Assume decisions \( c = 0 \), \( f = 0 \), \( h = 0 \) and \( i = 0 \)
- Assignment \( a = 0 \) caused conflict ⇒ learnt clause \( (a \lor c \lor f) \)
  implies \( a = 1 \)
- A conflict is again reached: \( (\neg d \lor \neg e \lor f) \) is unsatisfied
- \( (c = 0) \land (f = 0) \Rightarrow (\varphi = 0) \)
- \( (\varphi = 1) \Rightarrow (c = 1) \lor (f = 1) \)
Non-Chronological Backtracking

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

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]

- Assume decisions \( c = 0 \), \( f = 0 \), \( h = 0 \) and \( i = 0 \)
- Assignment \( a = 0 \) caused conflict \( \Rightarrow \) learnt clause \((a \lor c \lor f)\)
  implies \( a = 1 \)
- A conflict is again reached: \( (\neg d \lor \neg e \lor f) \) is unsatisfied
- \( (c = 0) \land (f = 0) \Rightarrow (\varphi = 0) \)
- \( (\varphi = 1) \Rightarrow (c = 1) \lor (f = 1) \)

- Learn new clause \((c \lor f)\)
Non-Chronological Backtracking

\[(a + c + f) \rightarrow (c + f)\]
Non-Chronological Backtracking

- Learnt clause: \((c \lor f)\)
- Need to backtrack, given new clause
- Backtrack to most recent decision: \(f = 0\)
- Clause learning and non-chronological backtracking are hallmarks of modern SAT solvers
Most Recent Backtracking Scheme

\[(a \lor c \lor f)\]
Most Recent Backtracking Scheme

\[(a \lor c \lor f)\]
Most Recent Backtracking Scheme

- Learnt clause: \((a \lor c \lor f)\)
- No need to assign \(a = 1\): backtrack to most recent decision: \(f = 0\)
- Search algorithm is no longer a traditional backtracking scheme
- Akin to dynamic backtracking
Evolution of SAT Solvers

<table>
<thead>
<tr>
<th>Instance</th>
<th>Posit'94</th>
<th>Grasp'96</th>
<th>Chaff'03</th>
<th>Minisat'03</th>
<th>Picosat'08</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssa2670-136</td>
<td>13.57</td>
<td>0.22</td>
<td>0.02</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>bf1355-638</td>
<td>310.93</td>
<td>0.02</td>
<td>0.02</td>
<td>0.00</td>
<td>0.03</td>
</tr>
<tr>
<td>design_3</td>
<td>&gt; 1800</td>
<td>3.93</td>
<td>0.18</td>
<td>0.17</td>
<td>0.93</td>
</tr>
<tr>
<td>design_1</td>
<td>&gt; 1800</td>
<td>34.55</td>
<td>0.35</td>
<td>0.11</td>
<td>0.68</td>
</tr>
<tr>
<td>4pipe_4_ooo</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>17.47</td>
<td>110.97</td>
<td>44.95</td>
</tr>
<tr>
<td>fifo8_300</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>348.50</td>
<td>53.66</td>
<td>39.31</td>
</tr>
<tr>
<td>w08_15</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>99.10</td>
<td>71.89</td>
</tr>
<tr>
<td>9pipe_9_ooo</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
</tr>
<tr>
<td>c6288</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
<td>&gt; 1800</td>
</tr>
</tbody>
</table>

- Modern SAT algorithms can solve instances with hundreds of thousands of variables and tens of millions of clauses.
Proof Complexity Characterizations

- Formulate CDCL as a proof system
  - CL: clause learning with restarts

- CL as powerful as general resolution (RES)

- In practice:
  - RES impractical in practice
  - CL very effective in practice

- So, why does CL work in practice?
  - Clause learning explained by sequence of (trivial) resolution operations
  - Clause learning (somehow) identifies the right resolution operations to perform
    - From the analysis of conflicts resulting from unit propagation
  - “Hard problems can be solved by exploiting structure” [J. Hooker, ITW’10]
Properties of CNF Formulas

• **Scale-free graphs**: nodes’ arity follows power law  
  [Li et al., IM’05]

• CNF formulas *can* exhibit properties of scale-free graphs
  
  – Experimental data is preliminary, and outliers do exist
  – Many possible uses
    
    Run SAT solver that performs best given node arity distribution  
  [Ansotegui et al., CP’09]
Outline

Preliminaries

The (Recent) Past
  The DPLL Algorithm

The Present
  Conflict-Driven Clause Learning (CDCL)
  Why Does It Work?

The (Near) Future

Conclusions
Domain Extensions

- Use richer modelling languages & allow for optimization
  - Pseudo-Boolean Constraints (PB)
    - Cardinality constraints, general PB constraints, etc.
    - Simple non-linear constraints
  - (Partial) (Weighted) MaxSAT
    - Soft clauses/constraints
  - Answer Set Programming (ASP)
  - Quantified Boolean Formulas (QBF)
    - Variable quantification
  - Satisfiability Modulo Theories (SMT)
    - Decidable fragments of FOL
  - Constraint Programming (?)
  - First Order Logic (?)
SAT-Based Problem Solving

- More Applications
  - Hundreds (thousands?) of documented applications of SAT and extensions of SAT
  - (Many?) more to be expected

- Better Modelling
  - Many SAT applications; some with naive SAT modelling
    - Sophisticated SAT-based modelling

- SAT solvers as oracles, but:
  - White-box SAT solver integration
    - Also for SAT extensions

- Access to computed models
- Access to unsatisfiable subformulas
- Access to learned clauses
- Access to variable/clause activity information
- ...
Problem-Specific Solvers

• Embed dedicated solvers within SAT solver
  – Adapt DPLL(T) framework used in SMT

Concrete examples:
  ▶ (Some) PBO solvers
  ▶ Handling parity constraints

[Manquinho&Marques-Silva, AMAI’04]
[Laitinen et al., ECAI’10]
Algorithmic Improvements

- Emulate extended resolution
  - Add/learn new variables/definitions, in addition to clause learning

- Integrate Stalmarck's Dilemma rule
  - Probe $UP(a)$ and $UP(\neg a)$
  - Compute intersection

- Integrate recursive learning's rule
  - Probe $UP(l)$, for $l \in \omega$
  - Compute intersection

[Audemard et al., AAAI’10]
[Sheeran & Stalmarck, FMCAD’98]
[Kunz & Pradhan, TCAD’94]
Exploit Tractability

- Enumerate models of polynomially-solvable subproblems
  - E.g. enumerate 2CNFSAT models
  - Impractical in general: number of models can be exponential
Exploit Tractability

- Enumerate models of polynomially-solvable subproblems
  - E.g. enumerate 2CNFSAT models
  - Impractical in general: number of models can be exponential

- Solve subproblems to find lower bounds in optimization problems
  - Applicable to PBO, MaxSAT, MaxSMT, etc.
  - Solve polynomial subclasses
    - E.g. 2CNFSAT, Horn-SAT, etc.
    - Does not capture hard part of problem instances
  - Recent advances in fixed-parameter algorithms
    - E.g. MaxSAT
    - Compute lower bounds by solving (extended) subproblems
Outline

Preliminaries

The (Recent) Past
   The DPLL Algorithm

The Present
   Conflict-Driven Clause Learning (CDCL)
   Why Does It Work?

The (Near) Future

Conclusions
Conclusions

- SAT is a success story in Computer Science

- Remarkable performance improvements since the mid 90s
  - Best solvers follow accepted recipe:
    Clause learning; Adaptive branching; Lazy data structures; Search restarts; etc.
  - Benchmark-driven development of solvers
  - Reasons for performance breakthroughs still unclear

- Many research directions
  - Extensions of SAT; Advanced modelling solutions; Problem-specific solvers; New algorithmic techniques; etc.
Thank you