Boolean Satisfiability

Based in part on J. Marques-Silva tutorial, ECAI 2010
Also on Darwiche&Pipatsisawat, handbook of SAT, chapter 3.
Outline

• Review: DPLL, Resolution
• CDCL: Conflict-Directed Clause Learning
  – Implication graphs,
  – asserting clauses,
  – Unique Implication points (UIPs)
• Watch literals
• Empirical evaluation

Reading chapters 3 and 4 in Handbook on SAT)
Outline

• Review: DPLL, Resolution
• CDCL: Conflict-Directed Clause Learning
• Implication graphs, asserting clauses, Unique Implication points (UIPs)
• Watch literals
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
Pure Literals

- A literal is pure if only occurs as a positive literal or as a negative literal in a CNF formula
  - Example:
    \[ \varphi = (\neg x_1 \lor x_2) \land (x_3 \lor \neg x_2) \land (x_4 \lor \neg x_5) \land (x_5 \lor \neg x_4) \]
    - \(x_1 \) and \(x_3\) and pure literals

- Pure literal rule:
  Clauses containing pure literals can be removed from the formula (i.e. just assign pure literals to the values that satisfy the clauses)
  - For the example above, the resulting formula becomes:
    \[ \varphi = (x_4 \lor \neg x_5) \land (x_5 \lor \neg x_4) \]

- A reference technique until the mid 90s; nowadays seldom used
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

\[
\begin{align*}
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
\end{align*}
\]

\[
\begin{align*}
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor \neg x_4)
\end{align*}
\]
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:
    - 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

[Davis&Putnam, JACM'60]
Resolution – An Example

\((x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \vdash\)
Resolution – An Example

\((x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor \neg x_4) \land (x_3 \lor \neg x_4) \vdash\)

\((\neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor \neg x_4) \land (x_3 \lor \neg x_4) \vdash\)
Resolution – An Example

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

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

\[(x_3 \lor \neg x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \vdash\]
Resolution – An Example

\((x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4)\) \vdash

\((\neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4)\) \vdash

\((x_3 \lor \neg x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4)\) \vdash

\((x_3 \lor x_4) \land (x_3 \lor \neg x_4)\) \vdash
Resolution – An Example

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

- Formula is SAT

Do directional resolution in the order: X3,x4,x2,x1
Outline

Preliminaries

Algorithms
   Local Search
   The DPLL Algorithm
   Conflict-Driven Clause Learning (CDCL)

SAT-Based Modelling
Outline

Preliminaries

Algorithms
  Local Search
  The DPLL Algorithm
  Conflict-Driven Clause Learning (CDCL)

SAT-Based Modelling
DPLL – 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
The DPLL Algorithm

- 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
An Example of DPLL

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

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

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

conflict
An Example of DPLL

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

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

Comparing with CSP:
- Sat can be decided before all variables are assigned
Complexity: when is unit propagation complete?....
Think Horn clauses
Outline

• Review: DPLL, Resolution
• Conflict-Directed Clause Learning (CDCL)
  – Implication graphs,
  – asserting clauses,
  – Unique Implication points (UIPs)
• Watch literals
Conflict Analysis: Implication Graphs

- The combination of these techniques makes sure that unit resolution is empowered every time a conflict arises, and that the solver will not repeat any mistake. The identification of conflict-driven clauses is done through a process known as conflict analysis, which analyzes a trace of unit resolution known as the implication graph.

Our vanilla CSP conflict did not take arc-consistency into account. In SAT, conflict-driven analysis does.
Implication graphs

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

Each node in an implication graph has the form \( l/V=v \), which means that variable \( V \) has been set to value \( v \) at level \( l \). Note that a variable is set either by a decision or by an implication. A variable is set by an implication if the setting is due to the application of unit resolution. Otherwise, it is set by a decision.
Deriving conflict clause

- Every cut in the implication graph defines a conflict set as long as that cut separates the decision variables (root nodes) from the contradiction (a leaf node).
- Any node (variable assignment) with an outgoing edge that crosses the cut will be in the conflict set.

leading to conflicts: sets:
{A=true, X=true}, {A=true, Y=true} and {A=true, Y=true, Z=true}.

\( \varphi = \)
1. \{A, B\}
2. \{B, C\}
3. \{¬A, ¬X, Y\}
4. \{¬A, X, Z\}
5. \{¬A, ¬Y, Z\}
6. \{¬A, X, ¬Z\}
7. \{¬A, ¬Y, ¬Z\}
Earliest minimal conflict?

\[ \varphi = \]
1. \{A, B\}
2. \{B, C\}
3. \{\neg A, \neg X, Y\}
4. \{\neg A, X, Z\}
5. \{\neg A, \neg Y, Z\}
6. \{\neg A, X, \neg Z\}
7. \{\neg A, \neg Y, \neg Z\}
8. \{\neg A, \neg X\}

For the graph in Figure 3.7(b), \{A = true\} is a conflict cut.

Conflict–driven clauses generated from cuts that contain exactly one variable assigned at the level of conflict are said to be asserting [ZMMM01]. Modern SAT solvers insist on learning only asserting clauses.
A UIP of a decision level in an implication graph is a variable setting at that decision level which lies on every path from the decision variable of that level to the contradiction. Intuitively, a UIP of a level is an assignment at the level that, by itself, is sufficient for implying the contradiction. In Figure 3.9, the variable setting 3/Y=true and 3/X=true would be UIPs as they lie on every path from the decision 3/X=true to the contradiction 3/{}. 

\[ \varphi = \]
1. \{A,B\}
2. \{B,C\}
3. \{¬A, ¬X, Y\}
4. \{¬A,X,Z\}
5. \{¬A, ¬Y,Z\}
6. \{¬A,X, ¬Z\}
7. \{¬A, ¬Y, ¬Z\}
8. \{¬A, ¬X\}

Deriving asserting clauses that contain the first UIP is popular: \{¬A ∨ ¬Y\} will be learnt.
Last slide
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 restarted

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

- Branching is guided by conflicts
  - E.g. VSIDS, etc.

[Davis et al., JACM’60, CACM’62]
[Marques-Silva&Sakallah, ICCAD’96]
[Marques-Silva&Sakallah, ICCAD’96]
[Gomes et al., AAAI’98]
[Moskewicz et al, DAC’01]
[Moskewicz et al, DAC’01]
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

  [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]

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

  [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

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

- 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
- \( (a = 0) \land (c = 0) \land (f = 0) \Rightarrow (\varphi = 0) \)
- \( (\varphi = 1) \Rightarrow (a = 1) \lor (c = 1) \lor (f = 1) \)
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 \( \Rightarrow \) learnt clause \( (a \lor c \lor f) \) implies \( a = 1 \)
- A conflict is again reached: \( (\neg d \lor \neg e \lor f) \) is unsatisfied
- \( (c = 0) \land (f = 0) \Rightarrow (\varphi = 0) \)
- \( (\varphi = 1) \Rightarrow (c = 1) \lor (f = 1) \)
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

\[(a + c + f) \quad \text{and} \quad (c + f)\]
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
Clause deletion policies

- Keep only the **small clauses**
  - For each conflict record one clause
  - Keep clauses of size $\leq K$
  - Large clauses get deleted when become unresolved

- Keep only the **relevant clauses**
  - Delete unresolved clauses with $\leq M$ free literals

- Keep only the clauses **that are used**
  - Keep track of clauses **activity**
Outline

• Review: DPLL, Resolution

• Conflict-Directed Clause Learning (CDCL)
  – Implication graphs,
  – asserting clauses,
  – Unique Implication points (UIPs)

• Watch literals
Data Structures

- **Key point:** only unit and unsatisfied clauses *must* be detected during search
  - Formula is **unsatisfied** when at least one clause is unsatisfied
  - Formula is **satisfied** when all the variables are assigned and there are no unsatisfied clauses
- **In practice:** unit and unsatisfied clauses may be identified using only two references

- **Standard data structures (adjacency lists):**
  - Each variable $x$ keeps a reference to all clauses containing a literal in $x$
- **Lazy data structures (watched literals):**
  - For each clause, only two variables keep a reference to the clause, i.e. only 2 literals are watched
Lazy Data Structures (watched literals)

- For each clause, only two variables keep a reference to the clause, i.e. only 2 literals are watched
  - If variable $x$ is assigned, only the clauses where literals in $x$ are watched need to be evaluated
  - If search backtracks, then nothing needs to be done

- Total number of references is $2 \times C$, where $C$ is the number of clauses
  - In general $L \gg 2 \times C$, in particular if clauses are learnt
BCP Algorithm (1.1/8)

- Big Invariants
  - Each clause has two watched literals.
  - If a clause can become unit via any sequence of assignments, then this sequence will include an assignment of one of the watched literals to F.
    - Example again: \((v1 + v2 + v3 + v4 + v5)\)
    - \((v1=X + v2=X + v3=? + v4=? + v5=? )\)
  - BCP consists of identifying unit (and conflict) clauses (and the associated implications) while maintaining the “Big Invariants”
BCP Algorithm (2/8)

- Let’s illustrate this with an example:

  \[ v_2 + v_3 + v_1 + v_4 + v_5 \]
  \[ v_1 + v_2 + v_3' \]
  \[ v_1 + v_2' \]
  \[ v_1' + v_4 \]
  \[ v_1' \]
BCP Algorithm (2.1/8)

- Let’s illustrate this with an example:

  \[
  \begin{align*}
  &v_2 + v_3 + v_1 + v_4 + v_5 \\
  &v_1 + v_2 + v_3' \\
  &v_1 + v_2' \\
  &v_1' + v_4 \\
  &v_1'
  \end{align*}
  \]

- Initially, we identify any two literals in each clause as the watched ones.
- Clauses of size one are a special case.
- One literal clause breaks invariants: handled as a special case (ignored hereafter).
BCP Algorithm (3/8)

- We begin by processing the assignment $v_1 = F$ (which is implied by the size one clause)

State: \((v_1=F)\)
Pending:

\[
\begin{align*}
  v_2 + v_3 + v_1 + v_4 + v_5 \\
  v_1 + v_2 + v_3' \\
  v_1 + v_2' \\
  v_1' + v_4
\end{align*}
\]
We begin by processing the assignment $v_1 = F$ (which is implied by the size one clause)

<table>
<thead>
<tr>
<th>State: ($v_1 = F$)</th>
<th>$v_2 + v_3 + v_1 + v_4 + v_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pending:</td>
<td>$v_1 + v_2 + v_3'$</td>
</tr>
<tr>
<td></td>
<td>$v_1 + v_2'$</td>
</tr>
<tr>
<td></td>
<td>$v_1' + v_4$</td>
</tr>
</tbody>
</table>

To maintain our invariants, we must examine each clause where the assignment being processed has set a watched literal to $F$. 
BCP Algorithm (3.2/8)

- We begin by processing the assignment $v_1 = F$ (which is implied by the size one clause)

  \[ v_2 + v_3 + v_1 + v_4 + v_5 \]

  **State:** $(v_1=F)$

  **Pending:**

  \[ v_1 + v_2 + v_3' \]

  \[ v_1 + v_2' \]

  \[ v_1' + v_4 \]

  $\rightarrow$

- To maintain our invariants, we must examine each clause where the assignment being processed has set a watched literal to $F$.

- We need not process clauses where a watched literal has been set to $T$, because the clause is now satisfied and so can not become unit.
BCP Algorithm (3.3/8)

- We begin by processing the assignment $v1 = F$ (which is implied by the size one clause)

  \[ v2 + v3 + v1 + v4 + v5 \]

  State: \((v1=F)\)

  Pending:

- To maintain our invariants, we must examine each clause where the assignment being processed has set a watched literal to $F$.
- We need not process clauses where a watched literal has been set to $T$, because the clause is now satisfied and so can not become unit.
- We certainly need not process any clauses where neither watched literal changes state (in this example, where $v1$ is not watched).
BCP Algorithm (4/8)

- Now let’s actually process the second and third clauses:

\[
\begin{align*}
\textcolor{red}{v2} & + \textcolor{green}{v3} + \textcolor{red}{v1} + v4 + v5 \\
\textcolor{red}{v1} & + \textcolor{green}{v2} + v3' \\
\textcolor{red}{v1} & + v2' \\
\textcolor{red}{v1'} & + v4
\end{align*}
\]

State: (v1=F)
Pending:
BCP Algorithm (4.1/8)

- Now let's actually process the second and third clauses:

  \[ v_2 + v_3 + v_1 + v_4 + v_5 \]

  \[ v_1 + v_2 + v_3' \]

  \[ v_1 + v_2' \]

  \[ v_1' + v_4 \]

  \[ v_2 + v_3 + v_1 + v_4 + v_5 \]

  \[ v_1 + v_2 + v_3' \]

  \[ v_1 + v_2' \]

  \[ v_1' + v_4 \]

  \[ \text{State: } (v_1=F) \]

  \[ \text{Pending:} \]

- For the second clause, we replace \( v_1 \) with \( v_3' \) as a new watched literal. Since \( v_3' \) is not assigned to F, this maintains our invariants.
BCP Algorithm (4.2/8)

- Now let’s actually process the second and third clauses:

  \[ v_2 + v_3 + v_1 + v_4 + v_5 \]
  \[ v_1 + v_2 + v_3' \]
  \[ v_1 + v_2' \]
  \[ v_1' + v_4 \]

  \[ v_2 + v_3 + v_1 + v_4 + v_5 \]
  \[ v_1 + v_2 + v_3' \]
  \[ v_1 + v_2' \]
  \[ v_1' + v_4 \]

  State: (v1=F)
  Pending:

  State: (v1=F)
  Pending: (v2=F)

- For the second clause, we replace \( v_1 \) with \( v_3' \) as a new watched literal. Since \( v_3' \) is not assigned to \( F \), this maintains our invariants.

- The third clause is unit. We record the new implication of \( v_2' \), and add it to the queue of assignments to process. Since the clause cannot again become unit, our invariants are maintained.
BCP Algorithm (5/8)

- Next, we process $v2'$. We only examine the first 2 clauses.

<table>
<thead>
<tr>
<th>$v2 + v3 + v1 + v4 + v5$</th>
<th>$v2 + v3 + v1 + v4 + v5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v1 + v2 + v3'$</td>
<td>$v1 + v2 + v3'$</td>
</tr>
<tr>
<td>$v1 + v2'$</td>
<td>$v1 + v2'$</td>
</tr>
<tr>
<td>$v1' + v4$</td>
<td>$v1' + v4$</td>
</tr>
</tbody>
</table>

State: $(v1=F, v2=F)$  
Pending: 

State: $(v1=F, v2=F)$  
Pending: $(v3=F)$

- For the first clause, we replace $v2$ with $v4$ as a new watched literal. Since $v4$ is not assigned to $F$, this maintains our invariants.

- The second clause is unit. We record the new implication of $v3'$, and add it to the queue of assignments to process. Since the clause cannot again become unit, our invariants are maintained.
BCP Algorithm (6/8)

- Next, we process $v_3'$. We only examine the first clause.

\[ v_2 + v_3 + v_1 + v_4 + v_5 \]
\[ v_1 + v_2 + v_3' \]
\[ v_1' + v_4 \]

\[ v_2 + v_3 + v_1 + v_4 + v_5 \]
\[ v_1 + v_2 + v_3' \]
\[ v_1' + v_4 \]

State: ($v_1=F$, $v_2=F$, $v_3=F$)
Pending:

State: ($v_1=F$, $v_2=F$, $v_3=F$)
Pending:

- For the first clause, we replace $v_3$ with $v_5$ as a new watched literal. Since $v_5$ is not assigned to $F$, this maintains our invariants.

- Since there are no pending assignments, and no conflict, BCP terminates and we make a decision. Both $v_4$ and $v_5$ are unassigned. Let’s say we decide to assign $v_4=T$ and proceed.
BCP Algorithm (7/8)

- Next, we process v4. We do nothing at all.

\[
\begin{align*}
v2 & + v3 + v1 + \textcolor{red}{v4} + v5 \\
v1 & + \textcolor{red}{v2} + v3' \\
v1 & + v2' \\
v1' & + v4
\end{align*}
\]

State: (v1=F, v2=F, v3=F, v4=T)

\[
\begin{align*}
v2 & + v3 + v1 + v4 + v5 \\
v1 & + \textcolor{red}{v2} + v3' \\
v1 & + v2' \\
v1' & + v4
\end{align*}
\]

State: (v1=F, v2=F, v3=F, v4=T)

- Since there are no pending assignments, and no conflict, BCP terminates and we make a decision. Only v5 is unassigned. Let’s say we decide to assign v5=F and proceed.
BCP Algorithm (8/8)

- Next, we process $v_5=F$. We examine the first clause.

\[
\begin{align*}
&v_2 + v_3 + v_1 + v_4 + v_5 \\
v_1 + v_2 + v_3' \\
v_1 + v_2' \\
v_1' + v_4 & \quad \rightarrow \quad \rightarrow \\
&v_2 + v_3 + v_1 + v_4 + v_5 \\
v_1 + v_2 + v_3' \\
v_1 + v_2' \\
v_1' + v_4
\end{align*}
\]

State: \((v_1=F, v_2=F, v_3=F, v_4=T, v_5=F)\)

State: \((v_1=F, v_2=F, v_3=F, v_4=T, v_5=F)\)

- The first clause is already satisfied by $v_4$ so we ignore it.
- Since there are no pending assignments, and no conflict, BCP terminates and we make a decision. No variables are unassigned, so the instance is SAT, and we are done.
Outline

• Review: DPLL, Resolution
• Conflict-Directed Clause Learning (CDCL)
  – Implication graphs,
  – asserting clauses,
  – Unique Implication points (UIPs)
• Watch literals
• Restarts, Empirical evaluation
• Plot for processor verification instance with branching randomization and 10000 runs
  - More than 50% of the runs require less than 1000 backtracks
  - A small percentage requires more than 10000 backtracks

• Run times of backtrack search SAT solvers characterized by heavy-tail distributions
  [Gomes et al.'98]
Restart

- Abandon the current search tree and reconstruct a new one.
- Helps reduce variance - adds to robustness in the solver.
- The clauses learned prior to the restart are still there after the restart and can help pruning the search space.

Conflic clause: $x_1' + x_3 + x_5'$
## 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.
Benchmarks

- Random
- Crafted
- Industrial
Outline

• Review: DPLL, Resolution
• Conflict-Directed Clause Learning (CDCL)
  – Implication graphs,
  – asserting clauses,
  – Unique Implication points (UIPs)
• Watch literals
• Restarts, Empirical evaluation