Part I

- Reminders -
  - What is Logic
  - Proofs by deduction
  - Proofs by enumeration
  - Decidability, Soundness and Completeness
  - Some notes on Propositional Logic
- Deciding Propositional Logic
  - SAT tools
  - BDDs

Next: Deciding Propositional Formulas

- SAT solvers
- Binary Decision Diagrams

The satisfiability Problem: enumeration

- Given a formula \( \varphi \), is \( \varphi \) satisfiable?

```plaintext
Boolean SAT(\varphi) {
    B := 0
    for all \( \alpha \in 2^{AP(\varphi)} \)
    if Eval(\varphi, \alpha)
        B := 1
        break
    end
    return B
}
```

- NP-Complete (the first-ever! – Cook’s theorem)
- There has got to be a better way …
A Basic SAT algorithm

- Given $\phi$ in CNF: $(x,y,z),(-x,y),(-y,z),(-x,-y,-z)$
  (i.e., $\phi = (x \lor y \lor z) \land (\neg x \lor y) \land (\neg y \lor z) \land (\neg x \lor \neg y \lor \neg z)$)

A Basic SAT algorithm

- Work to find a satisfying assignment affected by:
  - Order in which values (0, 1) are assigned
  - Order in which variables are added to assignment

SAT made some progress…
DPLL Algorithm (intuition)

- Given $\phi$ in CNF: $(x, y, z), (-x, y), (-y, z), (-x, -y, -z)$
  
  (i.e., $\phi = (x \lor y \lor z) \land (\neg x \lor y) \land (\neg y \lor z) \land (\neg x \lor \neg y \lor \neg z)$)

While (true)
{
  if (!Decide()) return (SAT);
  while (!BCP())
    if (!Resolve_Conflict()) return (UNSAT);
}

Decision Procedures
An algorithmic point of view

Basic Backtracking Search

- Organize the search in the form of a decision tree
  
  - Each node corresponds to a decision
  - Definition: Decision Level (DL) is the depth of the node in the decision tree.
  - Notation: $x=v@d$
    
    $x \in \{0, 1\}$ is assigned to $v$ at decision level $d$

- Choose the next variable and value.
  Return False if assignment satisfies formula

- Boolean Constraint Propagation:
  Apply the unit clause rule repeatedly extending assignment.
  Return False if reach a conflict; else return True

- Backtrack over assignments causing conflict.
  Return False if impossible

BCP prunes unsatisfiable sub-trees

- $x_1 = 0@1$
  
  $\omega_1 = (x_2 \lor x_3)$
  $\omega_2 = (\neg x_2 \lor \neg x_3)$

- $x_2 = 0@2$
  
  $x_3 = 1@1$

- No backtrack in this example, regardless of the decision!
Backtracking with BCP

\[ \omega_1 = (x_2 \lor x_3) \]
\[ \omega_2 = (\neg x_1 \lor \neg x_4) \]
\[ \omega_3 = (\neg x_2 \lor x_4) \]
\[ \omega_4 = (\neg x_1 \lor x_2 \lor \neg x_3) \]

\[ \Rightarrow x_4 = 0 \]
\[ \Rightarrow x_2 = 0 \]
\[ \Rightarrow x_3 = 1 \]

\{ (x_1, 0), (x_2, 0), (x_3, 1) \}

Status of a clause w/r to an assignment

- For a (partial) assignment, a clause can be
  - Satisfied: at least one literal is satisfied
  - Conflicting: all literals are assigned but none are satisfied
  - Unit: all but one literal are assigned but none are satisfied
  - Unresolved: all other cases

Example: \( C = (x_1 \lor x_2 \lor x_3) \)

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>SATISFIED</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>CONFLICTING</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>UNIT</td>
</tr>
</tbody>
</table>

Decision heuristics - DLIS

DLIS (Dynamic Largest Individual Sum) – choose the assignment that increases the most the number of satisfied clauses

- For a given variable \( x \):
  - \( P_x \) - # unresolved clauses in which \( x \) appears positively
  - \( N_x \) - # unresolved clauses in which \( x \) appears negatively
  - Let \( x \) be the literal for which \( P_x \) is maximal
  - Let \( y \) be the literal for which \( N_x \) is maximal
  - If \( P_x > N_x \), choose \( x \) and assign it 1
  - Otherwise choose \( y \) and assign it 0

Decision heuristics - JW

Jeroslow-Wang method

In each phase, for every literal \( l \), compute:

\[ J(l) := \sum_{\omega \in \omega} 2^{-|\omega|} \quad \text{(for clause } \omega) \]

- Choose a variable \( l \) that maximizes \( J(l) \).
- This gives an exponentially higher weight to literals in shorter clauses.
Pause... →

- We will see other (more advanced) decision heuristics soon.
- These heuristics are integrated with a mechanism called Learning with Conflict Clauses, which we will learn next.

Implication graphs and learning: option #1

Current truth assignment: \( x_1 = 0 @ 1, x_2 = 0 @ 3, x_9 = 0 @ 3, x_{11} = 1 @ 2, x_{12} = 1 @ 2 \)

Current decision assignment: \( x_1 = 1 @ 6 \)

\[
\begin{align*}
&\bar{x}_1 = (\neg x_2 \lor x_3) \\
&\bar{x}_2 = (\neg x_1 \lor x_2 \lor x_3) \\
&\bar{x}_3 = (\neg x_1 \lor x_2 \lor x_3) \\
&\bar{x}_4 = (\neg x_2 \lor x_3 \lor x_4) \\
&\bar{x}_5 = (\neg x_2 \lor x_3 \lor x_4) \\
&\bar{x}_6 = (\neg x_2 \lor x_3 \lor x_4) \\
&\bar{x}_7 = (x_2 \lor x_3 \lor x_4) \\
&\bar{x}_8 = (\neg x_1 \lor x_5 \lor x_6 \lor x_7) \\
&\bar{x}_9 = (x_1 \lor x_8) \\
&\bar{x}_{10} = (x_1 \lor x_8) \\
&\bar{x}_{11} = (x_1 \lor x_8) \\
&\bar{x}_{12} = (x_1 \lor x_8) \\
\end{align*}
\]

We learn the conflict clause \( a_{10} = (\neg x_1 \lor x_5 \lor x_6 \lor x_7) \)

Non-chronological backtracking

Which assignments caused the conflicts?

- \( x_1 = 0 @ 1 \)
- \( x_9 = 0 @ 3 \)
- \( x_{11} = 1 @ 2 \)
- \( x_{12} = 1 @ 2 \)

These assignments are sufficient for causing a conflict.

Backtrack to DL = 3
Non-chronological Backtracking

- So the rule is: backtrack to the largest decision level in the conflict clause.
- This works for both the initial conflict and the conflict after the flip.

More Conflict Clauses

- **Def**: A *Conflict Clause* is any clause implied by the formula.
- Let \( L \) be a set of literals labeling source nodes of a set of edges \( E_{cut} \) that “form a cut” in the implication graph, separating the conflict node from the roots.
- **Claim**: \( \forall_{EL} \neg l \) is a Conflict Clause.

More Conflict Clauses

- **Claim**: Let \( L \) be the set of literals labeling source nodes of a set of edges \( E_{cut} \) that “cut” the implication graph, separating the conflict node from the roots. Then \( \forall_{EL} \neg l \) is a Conflict Clause.

Conflict: \( \neg x_9 \lor x_4 \lor \neg x_{11} \)
More Conflict Clauses

- Claim: Let $L$ be the set of literals labeling source nodes of a set of edges $E_{cut}$ that “cut” the implication graph, separating the conflict node from the roots. Then $\lor_{l \in L} \neg l$ is a Conflict Clause.

  By construction, we know: 
  
  \[
  (\lor_{l \in L} l) \land (\land_{\omega \in W} \neg \omega) \rightarrow \text{False}
  \]

  Hence:
  
  \[
  \neg (\lor_{l \in L} l) \lor (\land_{\omega \in W} \omega)
  \]

  Or, equivalently:
  
  \[
  (\land_{\omega \in W} \omega) \rightarrow (\lor_{l \in L} \neg l)
  \]

  Hence, the formula implies $(\lor_{l \in L} \neg l)$. 

Pf: Let $W$ be the set of labels of edges reachable from a literal in $L$. 

Conflict clauses

- Def: An Asserting Clause is a Conflict Clause with a single literal from the current decision level.
- Backtracking (to the right level) makes it a Unit clause.
- Asserting clauses are those that force an immediate change in the search path.
- Modern solvers only consider Asserting Clauses.

Example

- Conflict, but not asserting: $(\neg x_6 \lor \neg x_7 \lor x_8 \lor x_9 \lor x_10 \lor x_11)$
- Conflict and asserting: $(\neg x_6 \lor \neg x_7 \lor x_8 \lor x_9)$
### Unique Implication Points (UIP’s)

- Definition: A Unique Implication Point (UIP) is any node other than the conflict node that is on all paths from the decision to the conflict node.
- The First-UIP is the closest UIP to the conflict.

![Diagram showing unique implication points and decision node]

### Conflict-driven backtracking (option #2)

- Conflict clause: \( (x_{10} \lor \neg x_4 \lor x_{11}) \)
- With standard Non-Chronological Backtracking we backtracked to DL = 6.
- Conflict-driven Backtrack: backtrack to the second highest decision level in the clause (without erasing it).
- In this case, to DL = 3.
- Q: why?

![Diagram showing conflict-driven backtracking]

### Conflict-driven Non-chronological Backtracking

- Solution:
  
  \[ \begin{align*}
  x_1 &= 0 \\
  x_2 &= 0 \\
  x_3 &= 1 \\
  x_4 &= 0 \\
  x_5 &= 0 \\
  x_6 &= 0 \\
  x_7 &= 1 \\
  x_9 &= 0 \\
  x_9 &= 1 \\
  x_6 &= 0 \\
  \end{align*} \]

- Progress of a SAT solver

  - Work invested in refuting \( x=1 \)
  - (some of it seems wasted)

  ![Diagram showing progress of a SAT solver]

- Decision:
  - BCP
- Conflict:
  - Refutation of \( x=1 \)
Conflict-Driven Backtracking

- So the rule is: backtrack to the second highest decision level $dl$ of lits in the conflict clause (undo assignments made at all higher levels).
- This way the literal with the currently highest decision level will be implied at level $dl$.
- Q: what if the conflict clause has a single literal? For example, from $(x \lor \neg y) \land (x \lor y)$ and decision $x=0$, we learn the conflict clause $(x)$.

Conflict clauses and Resolution

- Binary-resolution is a sound inference rule:
  \[ \frac{(a_1 \lor \ldots \lor a_n \lor \beta) \land (b_1 \lor \ldots \lor b_m \lor \neg \beta)}{(a_1 \lor \ldots \lor a_n \lor \neg \beta) \lor (b_1 \lor \ldots \lor b_m)} \]  
  (Binary Resolution)

- Example:
  \[ \frac{(x_1 \lor x_2) \land (\neg x_1 \lor x_3 \lor x_4)}{(x_1 \lor x_3 \lor x_4)} \]

Conflict clauses and resolution

- Consider the following example:

- Conflict clause: \(c_5: (x_2 \lor \neg x_4 \lor x_{10})\)
  \[ c_1 = (\neg x_4 \lor x_2 \lor x_9) \]
  \[ c_2 = (\neg x_4 \lor x_{10} \lor x_9) \]
  \[ c_3 = (\neg x_5 \lor \neg x_6 \lor \neg x_7) \]
  \[ c_4 = (\neg x_9 \lor \neg x_7) \]

- Resolution order: \(x_5, x_6, x_7\)
  \[ T_1 = \text{Res}(c_1, c_2) = (\neg x_5 \lor \neg x_9) \]
  \[ T_2 = \text{Res}(T_1, c_3) = (\neg x_7 \lor x_5 \lor x_9) \]
  \[ T_3 = \text{Res}(T_2, c_4) = (x_1 \lor \neg x_4 \lor x_{10}) \]
Finding the conflict clause:

1: procedure Analyze-Conflict
2: if current-decision-level = 0 then return -1
3: cl := current-conflicting-clause;
4: while (¬Stop-criterion-met(cl)) do
5: list := Last-assigned-literal(cl);
6: var := Variable-of-literal(list);
7: ante := Antecedent(var);
8: cl := Resolve(cl, ante,var);
9: add-clause-to-database(cl);
10: return clause-asserting-level(cl);

Applied to our example:

<table>
<thead>
<tr>
<th>name</th>
<th>cl</th>
<th>list var ante</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>(¬x1 ∨ x5)</td>
<td>x7 x7 c7</td>
</tr>
<tr>
<td></td>
<td>(¬x5 ∨ ¬x6)</td>
<td>¬x6 x6 c2</td>
</tr>
<tr>
<td></td>
<td>(¬x4 ∨ x10 ∨ ¬x3)</td>
<td>¬x5 x3 c1</td>
</tr>
<tr>
<td>c2</td>
<td>(¬x2 ∨ x3 ∨ x10)</td>
<td></td>
</tr>
</tbody>
</table>

d is asserting the first UIP
2nd highest decision level

The resolution graph

What is it good for?
Example: for computing an Unsatisfiable core

Resolution graph: example
Decision heuristics - VSIDS

VSIDS (Variable State Independent Decaying Sum)

1. Each variable in each polarity has a counter initialized to 0.
2. When a clause is added, the counters are updated.
3. The unassigned variable with the highest counter is chosen.
4. Periodically, all the counters are divided by a constant.

(Implemented in Chaff)

Decision heuristics – VSIDS (cont’d)

- Chaff holds a list of unassigned variables sorted by the counter value.
- Updates are needed only when adding conflict clauses.

Decision Heuristics - Berkmin

- Keep conflict clauses in a stack
- Choose the first unresolved clause in the stack
  - If there is no such clause, use VSIDS
- Choose from this clause a variable + value according to some scoring (e.g. VSIDS)

This gives absolute priority to conflicts.

VSIDS is a ‘quasi-static’ strategy:

- static because it doesn’t depend on current assignment
- dynamic because it gradually changes. Variables that appear in recent conflicts have higher priority.

This strategy is a conflict-driven decision strategy.

“...employing this strategy dramatically (i.e. an order of magnitude) improved performance ... “
- End of SAT (for now)

- Beginning of Binary Decision Diagrams