Resolution and Davis-Putnam

Computability and Logic
Logic Recap: Expressive Completeness
Rewriting Statements

• We can rephrase (rewrite) any occurrence of \( P \leftrightarrow Q \) as \( (P \rightarrow Q) \land (Q \rightarrow P) \).

• And, \( P \rightarrow Q \) itself can be rewritten as \( \neg P \lor Q \).

• Therefore, any traditional propositional logic expression (i.e. those using \( \neg, \land, \lor, \rightarrow, \leftrightarrow \)) can be rewritten into one that only uses the Boolean connectives (\( \neg, \land, \lor \)).
Negation Normal Form

- **Literals**: Atomic Sentences or negations thereof.
- **Negation Normal Form**: An expression built up with ‘∧’, ‘∨’, and literals.
- Using repeated DeMorgan and Double Negation, we can transform *any* expression built up with ‘∧’, ‘∨’, and ‘¬’ into an expression that is in Negation Normal Form.
- **Example**: ¬((A ∨ B) ∧ ¬C) ⇔ (DeMorgan)
  ¬(A ∨ B) ∨ ¬¬C ⇔ (Double Neg, DeM)
  (¬A ∧ ¬B) ∨ C
Disjunctive Normal Form

- **Disjunctive Normal Form**: A generalized disjunction of generalized conjunctions of literals.
- Using repeated distribution of $\land$ over $\lor$, *any* statement in Negation Normal Form can be written in Disjunctive Normal Form.
- Example:

\[
(A \lor B) \land (C \lor D) \iff (\text{Distribution}) \\
[(A \lor B) \land C] \lor [(A \lor B) \land D] \iff (\text{Distribution (2x)}) \\
(A \land C) \lor (B \land C) \lor (A \land D) \lor (B \land D)
\]
DNF and SOP

• In computer circuitry design, the term Sum Of Products (SOP) is often used, since if you consider T as ‘1’, and F as ‘0’, then $\land$ is like multiplication, and $\lor$ is like addition (where anything > 0 is considered 1)
  
  Thus, in computer circuitry design, $(A \land C) \lor (B \land C) \lor (A \land D) \lor (B \land D)$ is often written as: $AC + BC + AD + BD$ (‘Sum of Products’)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>AB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A+B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Conjunctive Normal Form
(or: Product of Sums: POS)

- **Conjunctive Normal Form**: A generalized conjunction of generalized disjunctions of literals.
- Using repeated distribution of $\lor$ over $\land$, *any* statement in Negation Normal Form can be written in Conjunctive Normal Form.
- Example:

  \[(A \land B) \lor (C \land D) \iff (\text{Distribution}) [\{(A \land B) \lor C\} \land [(A \land B) \lor D] \iff (\text{Distribution (2x)}) (A \lor C) \land (B \lor C) \land (A \lor D) \land (B \lor D)]\]
Special Cases

- Any literal (such as A or \(\neg B\)) is in NNF, DNF (it is a disjunction whose only disjunct is a conjunction whose only conjunct is that literal), and CNF
- A conjunction of literals (e.g. \(\neg A \land \neg B \land C\)) is in NNF, DNF (a disjunction whose only disjunct is that conjunction), and CNF
- A disjunction of literals is in NNF, DNF, and CNF
Summing Up

• Any traditional propositional logic expression can be transformed into a Boolean Logic expression
• Any Boolean logic expression can be put into NNF
• Any NNF expression can be put into CNF
• Any NNF expression can be put into CNF
• So, any traditional propositional logic expression can be put into NNF, CNF, and DNF
Expressing *any* truth-function using ‘and’, ‘or’, and ‘not’

- Even better: no matter what additional truth-functional operators you define (e.g. XOR, ID, “If … Then … Else”, etc.), you can always re-express them in terms of the Boolean connectives $\land$, $\lor$, and $\neg$!
- Indeed, *any* truth-function, no matter how complex, or defined over how many atomic statements, can be expressed in terms of the Boolean connectives $\land$, $\lor$, and $\neg$!
- ‘Proof’: generalize from example on next slide.
Expressive Completeness

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P*Q</th>
<th>Step 1: Create term for every ‘T’:</th>
<th>Step 2: Disjunct all terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>⇒ P∧¬Q</td>
<td>⇒ (P∧¬Q) ∨ (¬P∧Q)</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>⇒ P∧¬Q</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>⇒ ¬P∧Q</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that this works for any truth-function defined over any number of atomic statements.
We thus say that \{\land, \lor, \neg\} is expressively complete!!
**CNF and Truth-Tables**

- We can also generate a CNF that captures any truth-function from its truth-table:

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P*Q</th>
<th>Step 1: Create term for every ‘F’:</th>
<th>Step 2: Disjunct all terms</th>
<th>Step 3: Negate!</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>⇒ P ∧ Q</td>
<td>⇒ (P ∧ Q) ∨ (¬P ∧ ¬Q)</td>
<td>⇒ ¬((P ∧ Q) ∨ (¬P ∧ ¬Q)), i.e.</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td>⇒ ¬((P ∧ Q) ∧ (¬P ∧ ¬Q)), i.e.</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td>⇒ (¬P ∨ ¬ Q) ∧ (P ∨ Q) (CNF!)</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>⇒ ¬P ∧ ¬Q</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CNF and Truth-Tables II

• More directly:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Q</td>
<td>P*Q</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>⇒ ¬P ∨ ¬Q</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>⇒ P ∨ Q</td>
</tr>
</tbody>
</table>

Step 1:
Create negated ‘term’ for every ‘F’:

⇒ (¬P ∨ ¬Q) ∧ (P ∨ Q) (CNF!)
Resolution
Resolution

- Resolution is, like the tree method, a method to check for the logical consistency of a set of statements.
- Resolution requires all sentences to be put into CNF.
- A set of sentences in CNF is made into a *clause set S*: a set of clauses, where a *clause C* is a set of literals.
  - Each clause C represents a disjunction of literals
  - The clause set S represents a conjunction of disjunctions of literals
Resolution Rule

- Clauses are resolved using the *resolution rule*, and the resulting clause (the *resolvent*) is added to the clause set:

  \[ L \in C_1 \]
  \[ L' \in C_2 \]
  \[ C_{\text{NEW}} = C_1/L \cup C_2/L' \]
Putting into CNF

\neg (P \leftrightarrow Q) \iff (\text{Equiv})

\neg ((P \rightarrow Q) \land (Q \rightarrow P)) \iff (\text{Impl})

\neg (\neg P \lor Q) \land (\neg Q \lor P) \iff (\text{DeM})

\neg (\neg P \lor Q) \lor \neg (\neg Q \lor P) \iff (\text{DeM, DN})

(P \land \neg Q) \lor (Q \land \neg P) \iff (\text{Dist})

((P \land \neg Q) \lor Q) \land ((P \land \neg Q) \lor \neg P) \iff (\text{Dist})

(P \lor Q) \land (\neg Q \lor Q) \land (P \lor \neg P) \land (\neg Q \lor \neg P)
Resolution Graph

\[ \neg (P \leftrightarrow Q) \quad \neg (Q \leftrightarrow R) \quad \neg (P \leftrightarrow R) \]

\[ (P \lor Q) \land (\neg P \lor \neg Q) \quad (Q \lor R) \land (\neg Q \lor \neg R) \quad (P \lor R) \land (\neg P \lor \neg R) \]

\[ \{P, Q\} \quad \{\neg P, \neg Q\} \quad \{Q, R\} \quad \{\neg Q, \neg R\} \quad \{P, R\} \quad \{\neg P, \neg R\} \]

\[ \{\neg P, R\} \quad \{P, \neg Q\} \]

\[ \{P\} \quad \{\neg P\} \]

\[ \{\} \]
Satisfiability

- A clause is *satisfied* by a truth-value assignment if and only if that assignment makes at least one literal in that clause true.
- A clause set is *satisfiable* if and only if there is a truth-value assignment that satisfies all clauses in that clause set.
- Figuring out whether some clause set is satisfiable is the *satisfiability problem*. This problem is a central problem in computer science, as many problems in computer science can be reduced to a satisfiability problem.
- In our case: a set of sentences is consistent if and only if the corresponding clause set is satisfiable.
Soundness and Completeness of Resolution

- The rule of Resolution is sound, making the method of resolution sound as well (so, if the empty clause (which is a generalized disjunction of 0 disjuncts, which is a contradiction) can be resolved from a clause set, then that means that that clause set is indeed unsatisfiable.

- It can be shown that resolution is complete, i.e. that the empty clause can be resolved from any unsatisfiable clause set.
Resolutions as Derivations

\[ A \lor (B \land C) \implies (A \lor B) \land (A \lor C) \implies \]
\[ (A \lor B) \implies (D \lor E) \quad (\neg A \lor D \lor E) \land (\neg B \lor D \lor E) \implies \]
\[ \neg (A \lor B) \lor (D \lor E) \implies (\neg A \land \neg B) \lor (D \lor E) \quad \neg E \implies \]
\[ \neg A \implies \]
\[ \neg (C \land D) \implies \neg C \lor \neg D \implies \]

17.42 from LPL:

\[ A \lor (B \land C) \]
\[ \neg E \]
\[ (A \lor B) \rightarrow (D \lor E) \]
\[ \neg A \]
\[ \therefore C \land D \]
Resolutions as Decision Procedures

• Resolution can be made into a decision procedure by systematically exhausting all possible resolvents (of which there are finitely many).

• This will not be very efficient unless we add some resolution strategies.
Resolution Strategies

• Clause Elimination Strategies
  – Tautology Elimination
  – Subsumption Elimination
  – Pure Literal Elimination

• Resolving Strategies
  – Unit Preference Resolution
  – Linear Resolution
  – Ordered Resolution
  – Etc.
Tautology Elimination

- A *tautologous clause* is a clause that contains an atomic statement as well as the negation of that atomic statement. E.g. \{A, B, \neg A\} is tautologous.

- Obviously, for any tautologous clause C, any truth-value assignment is going to satisfy C.

- Hence, with S any clause set, and with S’ the clause set S with all tautologous clauses removed: S is satisfiable if and only if S’ is satisfiable.
Subsumption Elimination

• A clause $C_1$ subsumes a clause $C_2$ if and only if every literal contained in $C_1$ is contained in $C_2$, i.e. $C_1 \subseteq C_2$. E.g. $\{A, B\}$ subsumes $\{A, B, \neg C\}$
• Obviously, if $C_1$ subsumes $C_2$, then any truth-value assignment that satisfies $C_1$ will satisfy $C_2$.
• Hence, with $S$ any clause set, and $S'$ the clause set $S$ with all subsumed clauses removed: $S$ is satisfiable if and only if $S'$ is satisfiable.
Pure Literal Elimination

- A literal $L$ is *pure* with regard to a clause set $S$ if and only if $L$ is contained in at least one clause in $S$, but $L'$ is not.
- A clause is *pure* with regard to a clause set $S$ if and only if it contains a pure literal.
- Obviously, with $S$ any clause set, and with $S'$ the clause set $S$ with all pure clauses removed: $S$ is satisfiable if and only if $S'$ is satisfiable.
Unit Preference Resolution

- A *unit clause* is a clause that contains one literal.
- Unit preference resolution tries to resolve using unit clauses first.
Unit Literal Deletion and Splitting

• For any clause set $S$, $S_L$ is the clause set that is generated from $S$ as follows:
  – Remove all clauses from $S$ that contain $L$.
  – Remove all instances of $L'$ from all other clauses.

• Obviously, with $C = \{L\} \in S$, $S$ is satisfiable if and only if $S_L$ is satisfiable.

• It is also easy to see that for any clause set $S$, and any literal $L$: $S$ is satisfiable if and only if $S_L$ is satisfiable or $S_{L'}$ is satisfiable.

• The last observation suggests a splitting strategy that forms the basis of Davis-Putnam.
Davis-Putnam
Davis-Putnam

- Recursive routine Satisfiable(S) returns true iff S is satisfiable:

```java
boolean Satisfiable(S)
begin
    if S = {} return true;
    if S = {{}} return false;
    select L ∈ lit(S);
    return Satisfiable(S_L) || Satisfiable(S_L');
end
```
Davis-Putnam as Trees

\[
\begin{array}{c}
\{P, Q\} \\
\{P, \neg Q\} \\
\{\neg P, Q\} \\
\{\neg P, \neg Q\} \\
\{Q\} \\
\{\neg Q\} \\
\{\neg Q\} \\
\{\neg Q\} \\
\{\emptyset\} \\
\{\emptyset\}
\end{array}
\]
Step 1. Negate Conclusion

\[ A \rightarrow (N \lor Q) \]
\[ \neg(N \lor \neg A) \]
\[ A \rightarrow Q \]

\[ \neg(A \rightarrow Q) \]

Step 2: Put into CNF

\[ \neg A \lor N \lor Q \]
\[ \neg N \land A \]
\[ A \land \neg Q \]

Step 3: Make into clauses

\[ \{\neg A, N, Q\} \]
\[ \{\neg N\} \quad \{A\} \]
\[ \{A\} \quad \{\neg Q\} \]

Step 4: Put clause set at root of tree

Step 5: Do DP!
Simple Example
Invalid Argument

A → B
¬A
¬B → ¬¬B → B

→ ¬A ∨ B → \{¬A, B\}

¬¬B → ¬¬¬B → B

(A) (¬A)

Reached empty clause set:
So set of statements in root are consistent
So original argument is invalid
Model is given by branches: A False and B True
Making Davis-Putnam Efficient: Adding Bells and Whistles

• The routine on the previous slide is not very efficient. However, we can easily make it more efficient:
  – return false as soon as ∅ ∈ S
  – add the unit rule: if {L} ∈ S return Satisfiable(S_L)
  – strategically add clause deletion strategies (e.g. subsumption, pure literal)
  – strategically choose the literal on which to split

• As far as I have gathered from the ATP literature, such efficient Davis-Putnam routines are credited to do well in comparison to other ATP routines.
Step 1. Negate Conclusion

Step 2: Put into CNF

Step 3: Make into clauses

Step 4: Put clause set at root of tree

Step 5: Do DP!

Same example as before, but stopping early (i.e. as soon as {} is one of clauses)
Step 1. Negate Conclusion

\[ \neg (A \rightarrow Q) \]

Step 2: Put into CNF

\[ \neg A \lor N \lor Q \]

\[ \neg N \land A \]

\[ A \land \neg Q \]

Step 3: Make into clauses

\{ \neg A, N, Q \}

\{ \neg N \}

\{ A \}

\{ \neg Q \}

Step 4: Put clause set at root of tree

Step 5: Do DP!

Same example as before, but using unit rule
Davis-Putnam and Truth-Trees

• Observation: Davis-Putnam looks a bit like Truth-Tree method. In fact, on the next slides, we’ll see:
  – Like TT, ‘check marks’ can be used in representation of DP
  – Like TT, whole statements can be used (i.e. no need for clauses)

• How does Davis-Putnam differ from Truth-Trees?
  – Davis-Putnam is an ‘inside-out’ approach: it assigns a truth-value to atomic statements and determines the consequences of that assignment for the more complex statements composed of those atomic statements.
  – Truth-Trees is an ‘outside-in’ approach: it assigns truth-values to complex statements and determines the consequences of that assignment for the smaller statements it is composed of.
A → (N ∨ Q)
¬(N ∨ ¬A)
\[ \frac{\downarrow ¬(A → Q)}{A → Q} \]

Step 1. Negate Conclusion

Step 2: Put into CNF

\[ \neg A \lor N \lor Q \]
\[ \neg N \land A \]
A ∧ ¬Q

Step 3: Make into clauses

\[ \{\neg A, N, Q\} \]
\[ \{\neg N\} \quad \{A\} \]
\[ \{A\} \quad \{\neg Q\} \]

Step 4: Put clause set at root of tree

Step 5: Do DP!

Same example as before, but using check mark system
Step 1. Negate Conclusion

\[ \neg(A \rightarrow Q) \]

Step 2: Put into CNF

\[ \neg A \lor N \lor Q \rightarrow \neg N \land A \rightarrow A \land \neg Q \]

Step 3: Make into clauses

\[ \{\neg A, N, Q\} \rightarrow \{\neg N\} \rightarrow \{A\} \rightarrow \{A\} \rightarrow \{\neg Q\} \]

Step 4: Put clause set at root of tree

\[ \{\neg A, N, Q\} \rightarrow \{\neg N\} \rightarrow \{A\} \rightarrow \{\neg Q\} \]

Step 5: Do DP!

\[ \{Q\} \rightarrow \{\\} \rightarrow \times \]

Same example as before, but using check mark system and unit rule
\[ \begin{align*}
A & \rightarrow (N \lor Q) \\
\neg(N \lor \neg A) & \\
\hline
A & \rightarrow Q \\
\hline
\neg (A \rightarrow Q)
\end{align*} \]

Step 1: Negate Conclusion

Step 2: Put into CNF

\[ \begin{align*}
A & \rightarrow (N \lor Q) \\
\neg (N \lor \neg A) & \\
\iff & \\
\neg A \lor N \lor Q & \\
\neg N \land A & \\
A \land \neg Q & \\
\hline
\end{align*} \]

Step 3: Make into clauses

\[ \begin{align*}
\neg A, N, Q & \\
\neg N & \\
A & \\
\neg Q & \\
\hline
\end{align*} \]

Step 4: Put clause set at root of tree

Step 5: Do DP!

Same example as before, but using check mark system, unit rule, and TT rule that any branch with atomic P and \( \neg P \) can be closed
Can we do DP without CNF?

- Sure, simply consider a set of statements, and see what happens to each of the statements when some atomic claim is set to true or false, respectively.

- For example, when we set A to True:
  - \((A \lor B) \rightarrow (D \lor E)\) becomes
  - \((\text{True} \lor B) \rightarrow (D \lor E)\) becomes
  - \(\text{True} \rightarrow (D \lor E)\) becomes
  - \(D \lor E\)
## Rules for DP without CNF

<table>
<thead>
<tr>
<th>¬ True</th>
<th>True ∧ P</th>
<th>True ∨ P</th>
<th>True → P</th>
<th>True ↔ P</th>
</tr>
</thead>
<tbody>
<tr>
<td>⇒ False</td>
<td>⇒ P</td>
<td>⇒ True</td>
<td>⇒ P</td>
<td>⇒ P</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>¬ False</th>
<th>False ∧ P</th>
<th>False ∨ P</th>
<th>False → P</th>
<th>False ↔ P</th>
</tr>
</thead>
<tbody>
<tr>
<td>⇒ True</td>
<td>⇒ False</td>
<td>⇒ P</td>
<td>⇒ True</td>
<td>⇒ ¬P</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P ∧ True</th>
<th>P ∨ True</th>
<th>P → True</th>
<th>P ↔ True</th>
</tr>
</thead>
<tbody>
<tr>
<td>⇒ P</td>
<td>⇒ True</td>
<td>⇒ True</td>
<td>⇒ P</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P ∧ False</th>
<th>P ∨ False</th>
<th>P → False</th>
<th>P ↔ False</th>
</tr>
</thead>
<tbody>
<tr>
<td>⇒ False</td>
<td>⇒ P</td>
<td>⇒ ¬P</td>
<td>⇒ ¬P</td>
</tr>
</tbody>
</table>
Step 1. Negate Conclusion

\[ A \rightarrow (N \lor Q) \]

\[ \neg (N \lor \neg A) \]

\[ A \rightarrow Q \]

\[ \neg (A \rightarrow Q) \]

Step 2: Put statements at root of tree

\[ A \rightarrow (N \lor Q) \]

\[ \neg (N \lor \neg A) \]

\[ \neg (A \rightarrow Q) \]

Step 3: Do DP!

\[ (A) / \neg (\neg A) \]

\[ N \lor Q \quad \text{(True)} \]

\[ \neg N \quad \text{False} \]

\[ \neg Q \quad \text{False} \]

\[ (\neg N) \quad \times \]

\[ (\text{True}) \quad Q \quad \text{(True)} \]

\[ \neg Q \quad \text{False} \]

\[ (Q) / \neg (\neg Q) \]

\[ \neg Q \quad \text{False} \]

\[ (\neg Q) \quad \times \]

\[ (\text{True}) \quad \times \]

\[ \times \quad \times \]
Step 1. Negate Conclusion

\[ A \rightarrow (N \lor Q) \]
\[ \neg(N \lor \neg A) \]
\[ A \rightarrow Q \]

\[ \neg(A \rightarrow Q) \]

Step 2: Put statements at root of tree

\[ A \rightarrow (N \lor Q) \]
\[ \neg(N \lor \neg A) \]
\[ A \rightarrow Q \]

\[ \neg(A \rightarrow Q) \]

Step 3: Do DP!

\[ (A) / \quad \neg A \]
\[ N \lor Q \quad \text{False} \]
\[ \neg N \quad \text{False} \]

\[ \neg Q \quad \text{False} \]

\[ (N) \quad \neg Q \quad (\neg N) \quad \times \]

\[ \text{Same example, but leaving out the True’s} \]

\[ \neg Q \quad (Q) \quad \neg Q \quad (\neg Q) \]

\[ \times \quad \times \quad \times \quad \times \]

\[ \text{False} \quad \text{False} \quad \text{False} \]
Step 1. Negate Conclusion

\[ \neg (A \to Q) \]

Step 2: Put statements at root of tree

\[ A \to (N \lor Q) \]
\[ \neg (N \lor \neg A) \]
\[ A \to Q \]

\[ \neg (A \to Q) \]

Step 3: Do DP!

Same example, but using check mark system

\[ \sqrt{1} \]
\[ \sqrt{2} \]
\[ \sqrt{3} \]
\[ \sqrt{4} \]
\[ \sqrt{5} \]
\[ \sqrt{6} \]
\[ \sqrt{7} \]
17.42 from LPL:
A ∨ (B ∧ C)
¬E
(A ∨ B) → (D ∨ E)
¬A
∴ C ∧ D
17.42 from LPL:
A ∨ (B ∧ C)
¬E
(A ∨ B) → (D ∨ E)
¬A
∴ C ∧ D

Using check marks:

A ∨ (B ∧ C) $\sqrt{1}$
¬E $\sqrt{4}$
(A ∨ B) → (D ∨ E) $\sqrt{2}$
¬A $\sqrt{3}$
¬(C ∧ D) $\sqrt{8}$

A

¬A

B ∧ C $\sqrt{6}$
B → (D ∨ E) $\sqrt{5}$

B

¬B

C $\sqrt{9}$

D $\sqrt{40}$
¬C

D
¬D $\sqrt{11}$
D
False
×

False
×

False
×

False
×
Can DP and TT be combined?

- OK, Davis-Putnam now really starts to look like the truth tree method…
- Can these two methods be combined into one method?
- Sure!
- Project: Investigate efficiency of this method
Example: DP and TT Combo

\[\neg A \rightarrow B \quad \sqrt{4}\]
\[C \rightarrow (D \lor E) \quad \sqrt{2}\]
\[D \rightarrow \neg C \quad \sqrt{3}\]
\[A \rightarrow \neg E \quad \sqrt{5}\]
\[\neg (C \rightarrow B) \quad \sqrt{1}\]
\[C \quad \sqrt{6}\]
\[\neg B \quad \sqrt{7}\]
\[D \lor E \quad \sqrt{8}\]
\[\neg D \quad \sqrt{9}\]
\[\neg \neg A \quad \sqrt{10}\]
\[A \quad \sqrt{11}\]
\[E \quad \sqrt{12}\]
\[\neg E \quad \sqrt{13}\]
\[\times_7\]

17.43 from LPL:
\[\neg A \rightarrow B \quad \sqrt{4}\]
\[C \rightarrow (D \lor E) \quad \sqrt{2}\]
\[D \rightarrow \neg C \quad \sqrt{3}\]
\[A \rightarrow \neg E \quad \sqrt{5}\]
\[\neg (C \rightarrow B) \quad \sqrt{1}\]
\[C \quad \sqrt{6}\]
\[\neg B \quad \sqrt{7}\]

1. TT rule: decompose \(\neg (C \rightarrow B)\)
2. Unit rule: reduce with regard to C
3. Unit rule: reduce with regard to C
4. Unit rule: reduce with regard to \(\neg B\)
5. TT rule: decompose \(\neg \neg A\)
6. Unit rule: reduce with regard to \(\neg B\)
7. Close between \(E\) and \(\neg E\)
Exercise

• Show the argument below to be valid using:
  – 1. Resolution
  – 2. Davis-Putnam (on clauses)
  – 3. Davis-Putnam (on original statements)
  – 4. Davis-Putnam and Truth-Tree combo

\[
\begin{align*}
Q \lor \neg S \\
(P \land Q) & \iff R \\
\neg S & \rightarrow R \\
& \quad \ldots \ldots \\
\neg P & \rightarrow (Q \leftrightarrow S)
\end{align*}
\]
Projects

• Compare and contrast efficiency of different methods
  – How is efficiency effected by
    • Using Clause elimination strategies
    • Using Unit rule
    • Not putting into CNF
    • Etc.
  – What about combinations of different methods?