Resolution and Davis-Putnam

Computability and Logic

Logic Recap:
Expressive Completeness

Rewriting Statements

- We can rephrase (rewrite) any occurrence of $P \leftrightarrow Q$ as $(P \rightarrow Q) \wedge (Q \rightarrow P)$.
- And, $P \rightarrow Q$ itself can rewritten as $\neg P \vee Q$
- Therefore, any traditional propositional logic expression (i.e. those using $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$) can be rewritten into one that only uses the Boolean connectives ($\neg$, $\wedge$, $\vee$).

Negation Normal Form

- *Literals*: Atomic Sentences or negations thereof.
- *Negation Normal Form*: An expression built up with '$\wedge$', '$\vee$', and literals.
- Using repeated DeMorgan and Double Negation, we can transform *any* expression built up with '$\wedge$', '$\vee$', and '$\neg$' into an expression that is in Negation Normal Form.
- Example: $\neg((A \vee B) \wedge \neg C) \Leftrightarrow$ (DeMorgan)
  $\neg(A \vee B) \vee \neg\neg C \Leftrightarrow$ (Double Neg, DeM)
  $(\neg A \wedge \neg B) \vee C$

Disjunctive Normal Form

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

$(A \vee B) \wedge (C \vee D) \Leftrightarrow$ (Distribution)

$[(A \vee B) \wedge C] \vee [(A \vee B) \wedge D] \Leftrightarrow$ (Distribution (2x))

$(A \wedge C) \vee (B \wedge C) \vee (A \wedge D) \vee (B \wedge D)$

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 $\wedge$ is like multiplication, and $\vee$ is like addition (where anything > 0 is considered 1)
  - Thus, in computer circuitry design, $(A \wedge C) \vee (B \wedge C) \vee (A \wedge D) \vee (B \wedge D)$ is often written as: AC + BC + AD + BD (‘Sum of Products’)

A	B	AB
1	1	1
1	0	0
0	1	0
0	0	0

A	B	A+B
1	1	1
1	0	1
0	1	1
0	0	0

Conjunctive Normal Form (or: Product of Sums: POS)

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

$(A \wedge B) \vee (C \wedge D) \Leftrightarrow$ (Distribution)
$[(A \wedge B) \vee C] \wedge [(A \wedge B) \vee D] \Leftrightarrow$ (Distribution (2x))
$(A \vee C) \wedge (B \vee C) \wedge (A \vee D) \wedge (B \vee D)$

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. $\neg A \wedge \neg B \wedge C$) 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 $\wedge$, $\vee$, 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 $\wedge$, $\vee$, and $\neg$!
- ‘Proof’: generalize from example on next slide.

Expressive Completeness

P	Q	P*Q
T	T	F
T	F	T
F	T	T
F	F	F

Step 1:
Create term for every ‘T’:

$\Rightarrow P \wedge \neg Q$

$\Rightarrow \neg P \wedge Q$

Step 2:
Disjunct all terms

$\Rightarrow (P \wedge \neg Q) \vee (\neg P \wedge Q)$

(note: expression is in DNF)

Note that this works for any truth-function defined over any number of atomic statements.
We thus say that $\{\wedge, \vee, \neg\}$ is expressively complete!!

CNF and Truth-Tables

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

Step 1:
Create term for every ‘F’:

P	Q	P*Q	
T	T	F	$\Rightarrow P \land Q$
T	F	T	
F	T	T	
F	F	F	$\Rightarrow \neg P \land \neg Q$

Step 2:
Disjunct all terms

$\Rightarrow (P \land Q) \lor (\neg P \land \neg Q)$

Step 3:
Negate!

$\Rightarrow \neg((P \land Q) \lor (\neg P \land \neg Q))$, i.e.
$\Rightarrow \neg(P \land Q) \land \neg(\neg P \land \neg Q)$, i.e.
$\Rightarrow (\neg P \lor \neg Q) \land (P \lor Q)$ (CNF!)

CNF and Truth-Tables II

- More directly:

Step 1:
Create negated 'term' for every 'F':

P	Q	P*Q	
T	T	F	$\Rightarrow \neg P \vee \neg Q$
T	F	T	
F	T	T	
F	F	F	$\Rightarrow P \vee Q$

Step 2:
Conjunct terms

$\Rightarrow (\neg P \vee \neg\, Q) \wedge (P \vee Q)$ (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:

$$\frac{L \in C_1 \qquad L' \in C_2}{C_{NEW} = C_1/L \cup C_2/L'}$$

Putting into CNF

$\neg(P \leftrightarrow Q) \quad \Leftrightarrow$ (Equiv)

$\neg((P \rightarrow Q) \wedge (Q \rightarrow P)) \quad \Leftrightarrow$ (Impl)

$\neg((\neg P \vee Q) \wedge (\neg Q \vee P)) \quad \Leftrightarrow$ (DeM)

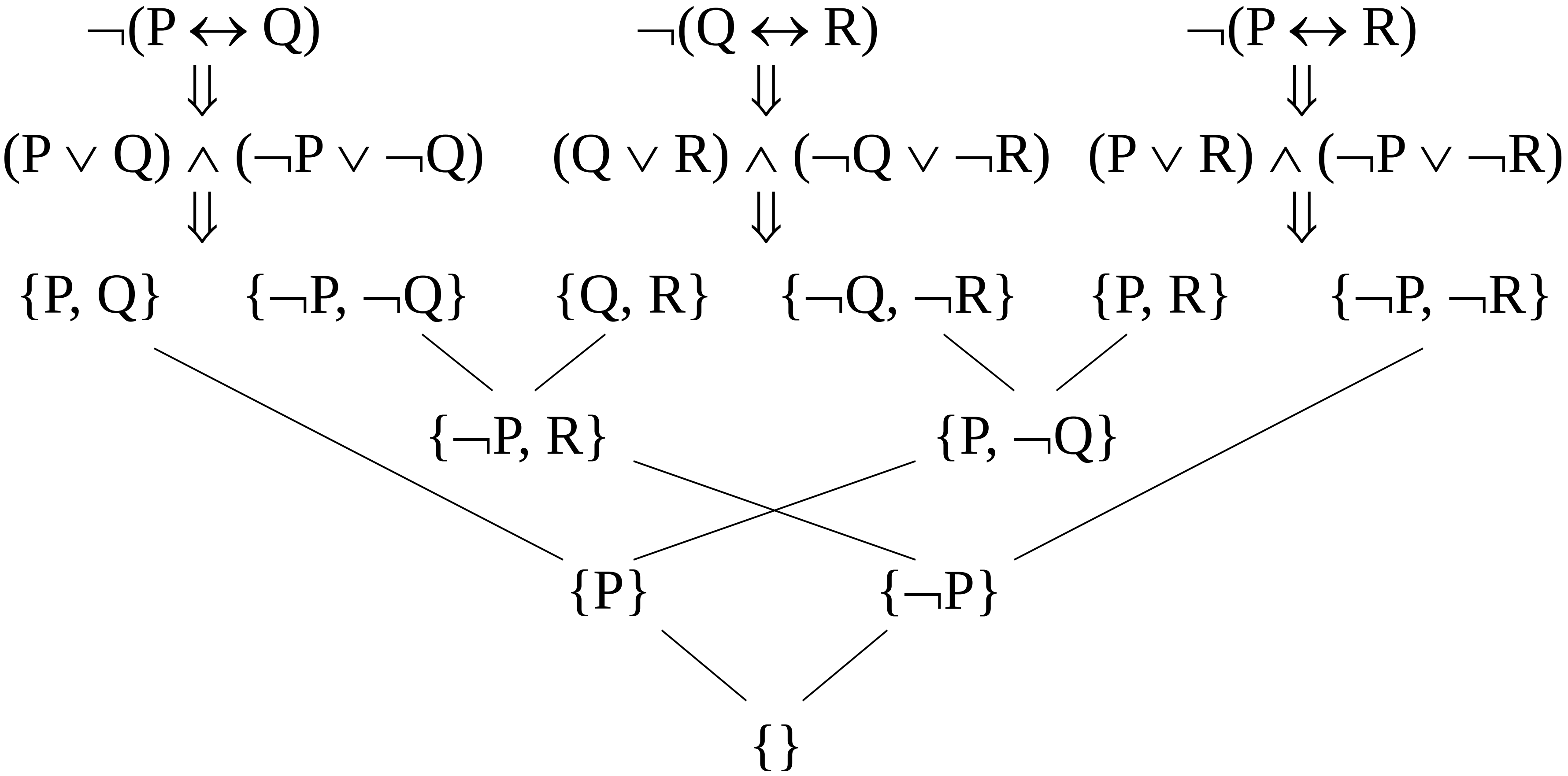
$\neg(\neg P \vee Q) \vee \neg(\neg Q \vee P) \quad \Leftrightarrow$ (DeM, DN)

$(P \wedge \neg Q) \vee (Q \wedge \neg P) \quad \Leftrightarrow$ (Dist)

$((P \wedge \neg Q) \vee Q) \wedge ((P \wedge \neg Q) \vee \neg P) \quad \Leftrightarrow$ (Dist)

$(P \vee Q) \wedge (\neg Q \vee Q) \wedge (P \vee \neg P) \wedge (\neg Q \vee \neg P)$

Resolution Graph



$\neg(P \leftrightarrow Q)$ $\quad\quad$ $\neg(Q \leftrightarrow R)$ $\quad\quad$ $\neg(P \leftrightarrow R)$

$\Downarrow$ $\quad\quad$ $\Downarrow$ $\quad\quad$ $\Downarrow$

$(P \vee Q) \wedge (\neg P \vee \neg Q)$ $\quad$ $(Q \vee R) \wedge (\neg Q \vee \neg R)$ $\quad$ $(P \vee R) \wedge (\neg P \vee \neg R)$

$\Downarrow$ $\quad\quad$ $\Downarrow$ $\quad\quad$ $\Downarrow$

$\{P, Q\}$ $\quad$ $\{\neg P, \neg Q\}$ $\quad$ $\{Q, R\}$ $\quad$ $\{\neg Q, \neg R\}$ $\quad$ $\{P, R\}$ $\quad$ $\{\neg P, \neg R\}$

$\{\neg P, R\}$ $\quad\quad$ $\{P, \neg Q\}$

$\{P\}$ $\quad\quad$ $\{\neg P\}$

$\{\}$

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

$$A \lor (B \land C) \Rightarrow (A \lor B) \land (A \lor C) \Rightarrow$$

$$\begin{tikzcd}
(A \lor B) \to (D \lor E) \arrow[d, Rightarrow] & (\neg A \lor D \lor E) \land (\neg B \lor D \lor E) \Rightarrow \\
\neg(A \lor B) \lor (D \lor E) \arrow[r, Rightarrow] & (\neg A \land \neg B) \lor (D \lor E) \arrow[u, Rightarrow]
\end{tikzcd}$$

$$\neg E \Rightarrow$$

$$\neg A \Rightarrow$$

$$\neg(C \land D) \Rightarrow \neg C \lor \neg D \Rightarrow$$

1.	$\{A, B\}$	
2.	$\{A, C\}$	
3.	$\{\neg A, D, E\}$	
4.	$\{\neg B, D, E\}$	
5.	$\{\neg E\}$	
6.	$\{\neg A\}$	
7.	$\{\neg C, \neg D\}$	
8.	$\{B\}$	1,6
9.	$\{C\}$	2,6
10.	$\{D, E\}$	4,8
11.	$\{D\}$	5,10
12.	$\{\neg D\}$	7,9
13.	$\{\}$	11,12

17.42 from LPL:
$A \lor (B \land C)$
$\neg E$
$(A \lor B) \to (D \lor E)$
$\neg A$
$\therefore\ C \land D$

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, ¬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, ¬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.

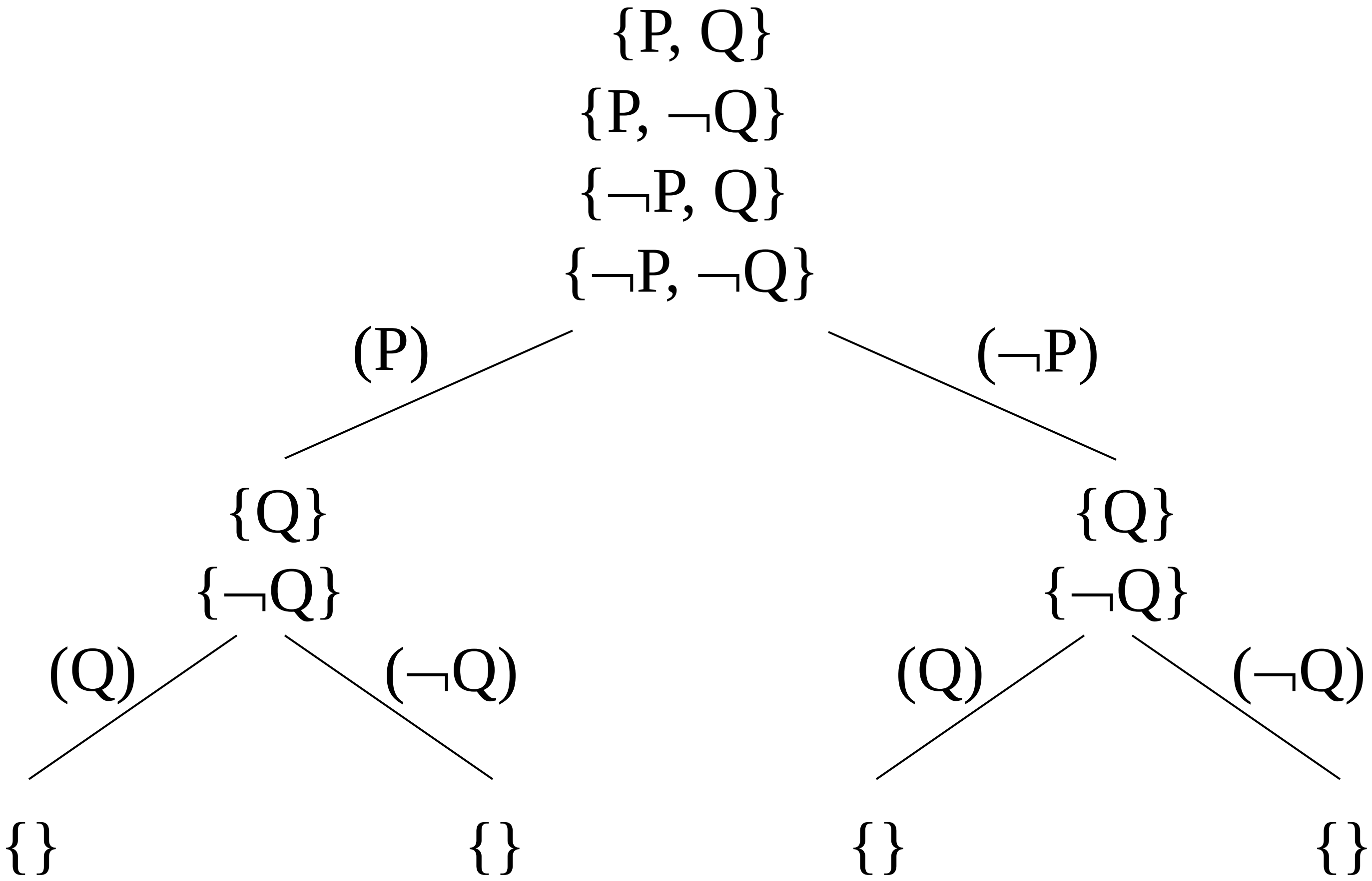
Davis-Putnam

Davis-Putnam

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

```
boolean Satisfiable(S)
begin
    if S = {} return true;
    if S = {{}} return false;
    select $L \in$ lit(S);
    return Satisfiable($S_L$) || Satisfiable($S_{L'}$);
end
```

Davis-Putnam as Trees



{P, Q}
{P, ¬Q}
{¬P, Q}
{¬P, ¬Q}

(P)

(¬P)

{Q}
{¬Q}

{Q}
{¬Q}

(Q)

(¬Q)

(Q)

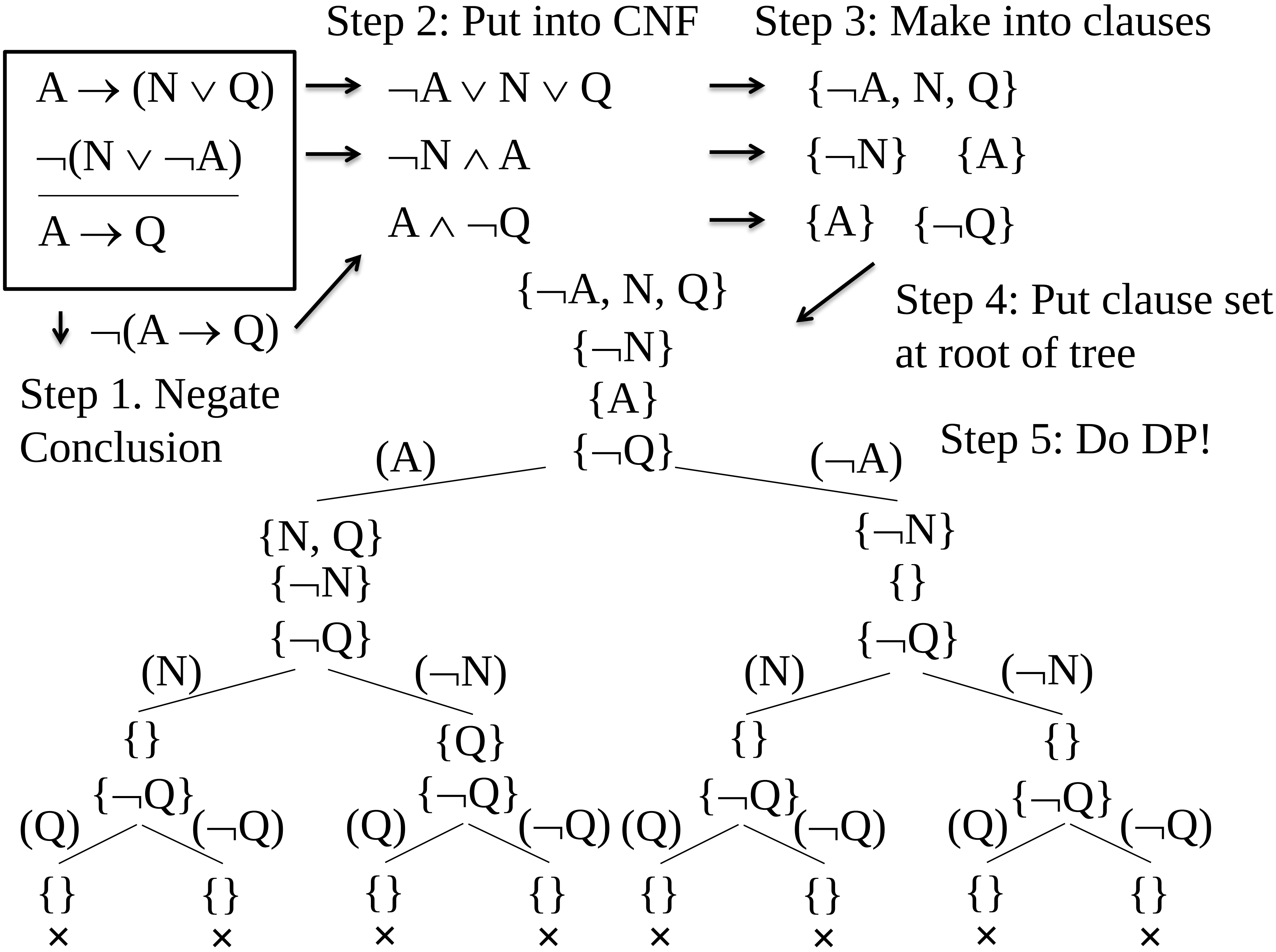
(¬Q)

{}

{}

{}

{}



$A \rightarrow (N \vee Q)$

$\neg(N \vee \neg A)$

$A \rightarrow Q$

↓ $\neg(A \rightarrow Q)$

Step 1. Negate Conclusion

Step 2: Put into CNF

$\neg A \vee N \vee Q$

$\neg N \wedge A$

$A \wedge \neg Q$

Step 3: Make into clauses

$\{\neg A, N, Q\}$

$\{\neg N\}$ $\{A\}$

$\{A\}$ $\{\neg Q\}$

Step 4: Put clause set at root of tree

$\{\neg A, N, Q\}$
$\{\neg N\}$
$\{A\}$
$\{\neg Q\}$

Step 5: Do DP!

(A)

$\{N, Q\}$
$\{\neg N\}$
$\{\neg Q\}$

$(\neg A)$

$\{\neg N\}$
$\{\}$
$\{\neg Q\}$

(N) $\{\}$ $\{\neg Q\}$ (Q) $\{\}$ × $(\neg Q)$ $\{\}$ ×

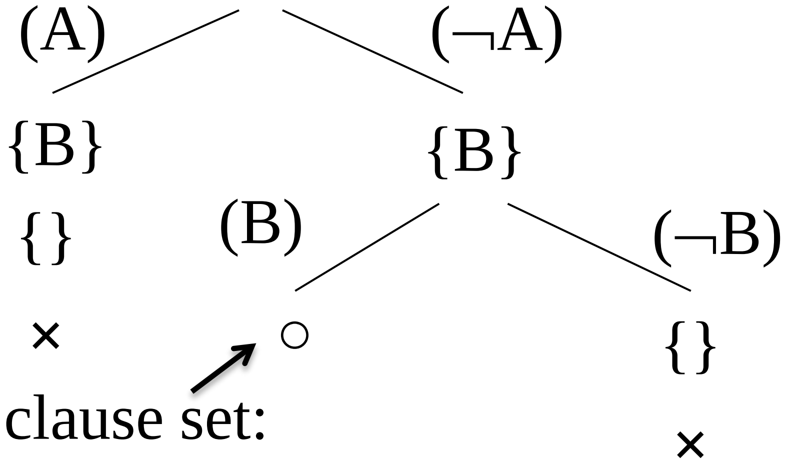
$(\neg N)$ $\{Q\}$ $\{\neg Q\}$ (Q) $\{\}$ × $(\neg Q)$ $\{\}$ ×

(N) $\{\}$ $\{\neg Q\}$ (Q) $\{\}$ × $(\neg Q)$ $\{\}$ ×

$(\neg N)$ $\{\}$ $\{\neg Q\}$ (Q) $\{\}$ × $(\neg Q)$ $\{\}$ ×

Simple Example
Invalid Argument

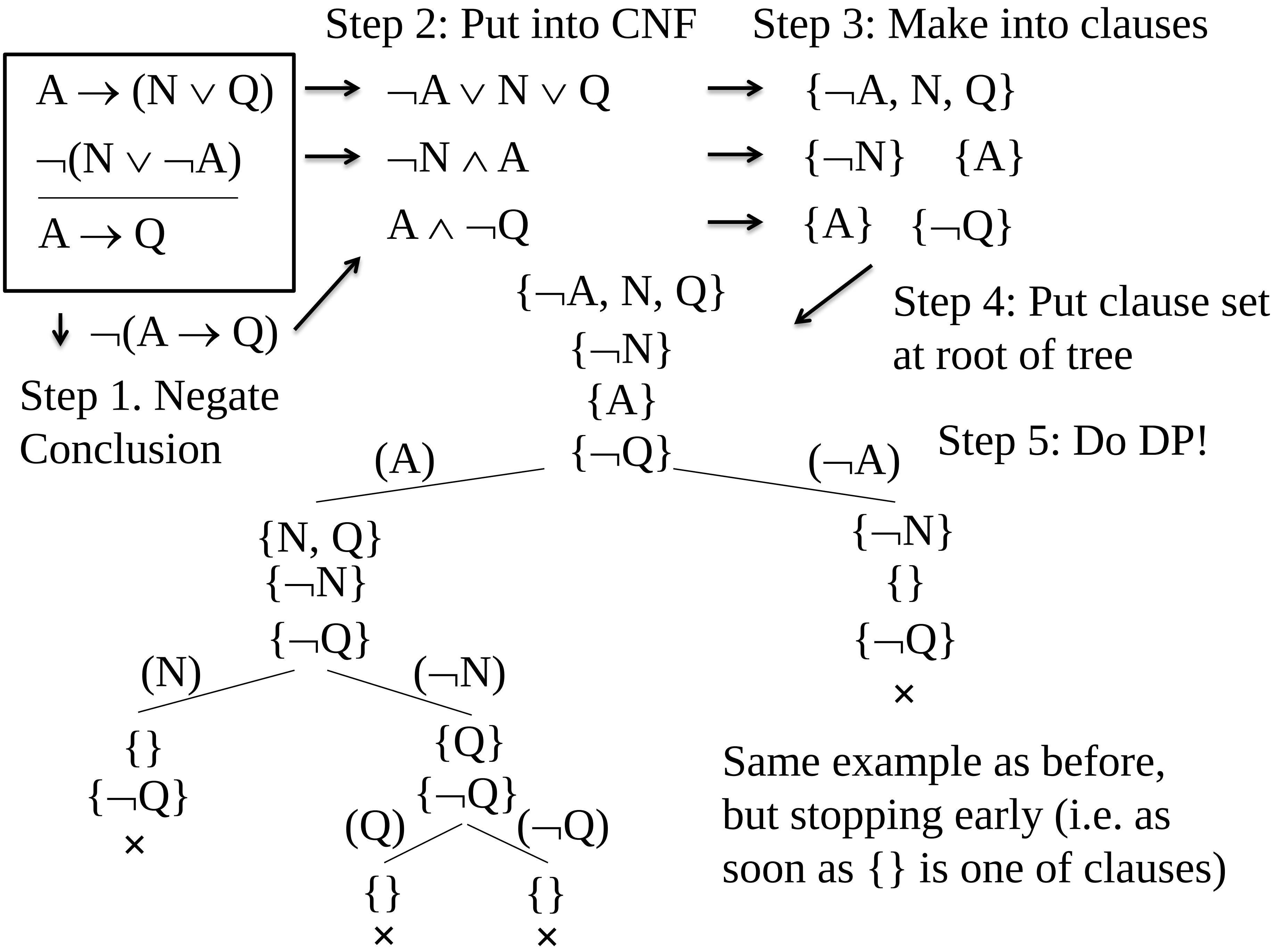
$$\begin{array}{lllllll}
A \to B & & & \longrightarrow & \neg A \vee B & \longrightarrow & \{\neg A , B\} \\
\neg A & & & \longrightarrow & \neg A & \longrightarrow & \{\neg A\} \\ \cline{1-1}
\neg B & \longrightarrow & \neg\neg B & \longrightarrow & B & \longrightarrow & \{B\}
\end{array}$$



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 $\{\} \in S$
  - add the unit rule: if $\{L\} \in S$ 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.



$A \rightarrow (N \vee Q)$

$\neg(N \vee \neg A)$

$A \rightarrow Q$

↓ $\neg(A \rightarrow Q)$

Step 1. Negate Conclusion

Step 2: Put into CNF

$\neg A \vee N \vee Q$

$\neg N \wedge A$

$A \wedge \neg Q$

Step 3: Make into clauses

$\{\neg A, N, Q\}$

$\{\neg N\}$ $\{A\}$

$\{A\}$ $\{\neg Q\}$

Step 4: Put clause set at root of tree

$\{\neg A, N, Q\}$
$\{\neg N\}$
$\{A\}$
$\{\neg Q\}$

Step 5: Do DP!

(A)

$\{N, Q\}$
$\{\neg N\}$
$\{\neg Q\}$

(N)

$\{\}$
$\{\neg Q\}$
×

$(\neg N)$

$\{Q\}$
$\{\neg Q\}$

(Q)

$\{\}$
×

$(\neg Q)$

$\{\}$
×

$(\neg A)$

$\{\neg N\}$
$\{\}$
$\{\neg Q\}$
×

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

Step 2: Put into CNF

Step 3: Make into clauses

$A \rightarrow (N \vee Q)$ ⟶ $\neg A \vee N \vee Q$ ⟶ $\{\neg A, N, Q\}$

$\neg(N \vee \neg A)$ ⟶ $\neg N \wedge A$ ⟶ $\{\neg N\}$ $\{A\}$

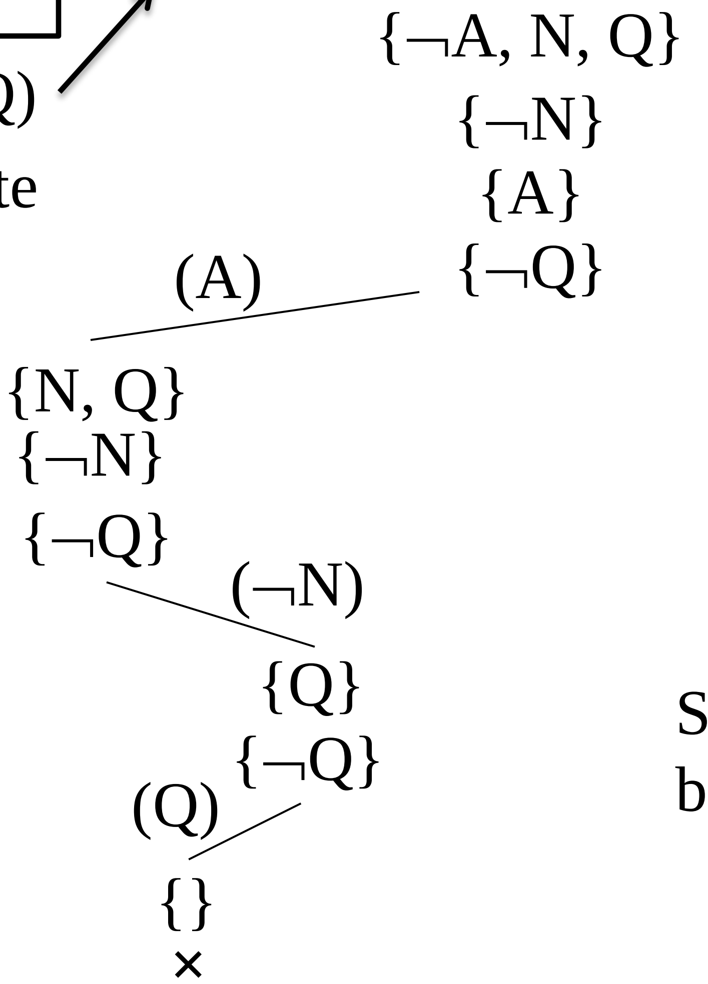
$A \rightarrow Q$

↓ $\neg(A \rightarrow Q)$ ⟶ $A \wedge \neg Q$ ⟶ $\{A\}$ $\{\neg Q\}$

Step 1. Negate Conclusion

Step 4: Put clause set at root of tree

Step 5: Do DP!



$\{\neg A, N, Q\}$
$\{\neg N\}$
$\{A\}$
$\{\neg Q\}$

(A)

$\{N, Q\}$
$\{\neg N\}$
$\{\neg Q\}$

$(\neg N)$

$\{Q\}$
$\{\neg Q\}$

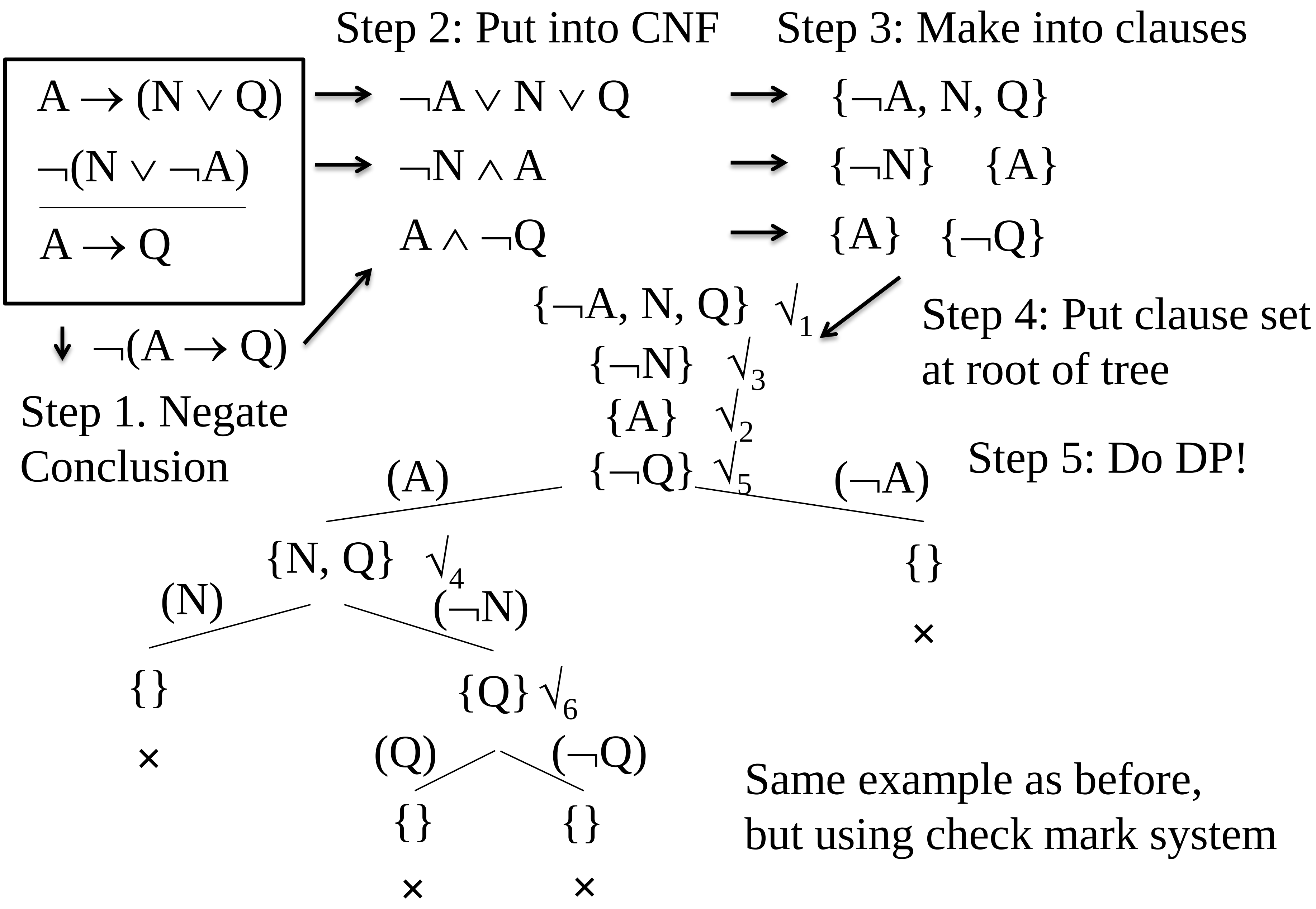
(Q)

$\{\}$
×

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.



Step 2: Put into CNF

Step 3: Make into clauses

$A \rightarrow (N \vee Q)$ → $\neg A \vee N \vee Q$ → $\{\neg A, N, Q\}$

$\neg(N \vee \neg A)$ → $\neg N \wedge A$ → $\{\neg N\}$ $\{A\}$

$A \rightarrow Q$

↓ $\neg(A \rightarrow Q)$ → $A \wedge \neg Q$ → $\{A\}$ $\{\neg Q\}$

Step 1. Negate Conclusion

Step 4: Put clause set at root of tree

$\{\neg A, N, Q\}$ $\surd_1$

$\{\neg N\}$ $\surd_3$

$\{A\}$ $\surd_2$

$\{\neg Q\}$ $\surd_5$

Step 5: Do DP!

(A)

$\{N, Q\}$ $\surd_4$

(N)

$\{\}$

×

$(\neg N)$

$\{Q\}$ $\surd_6$

(Q)

$\{\}$

×

$(\neg Q)$

$\{\}$

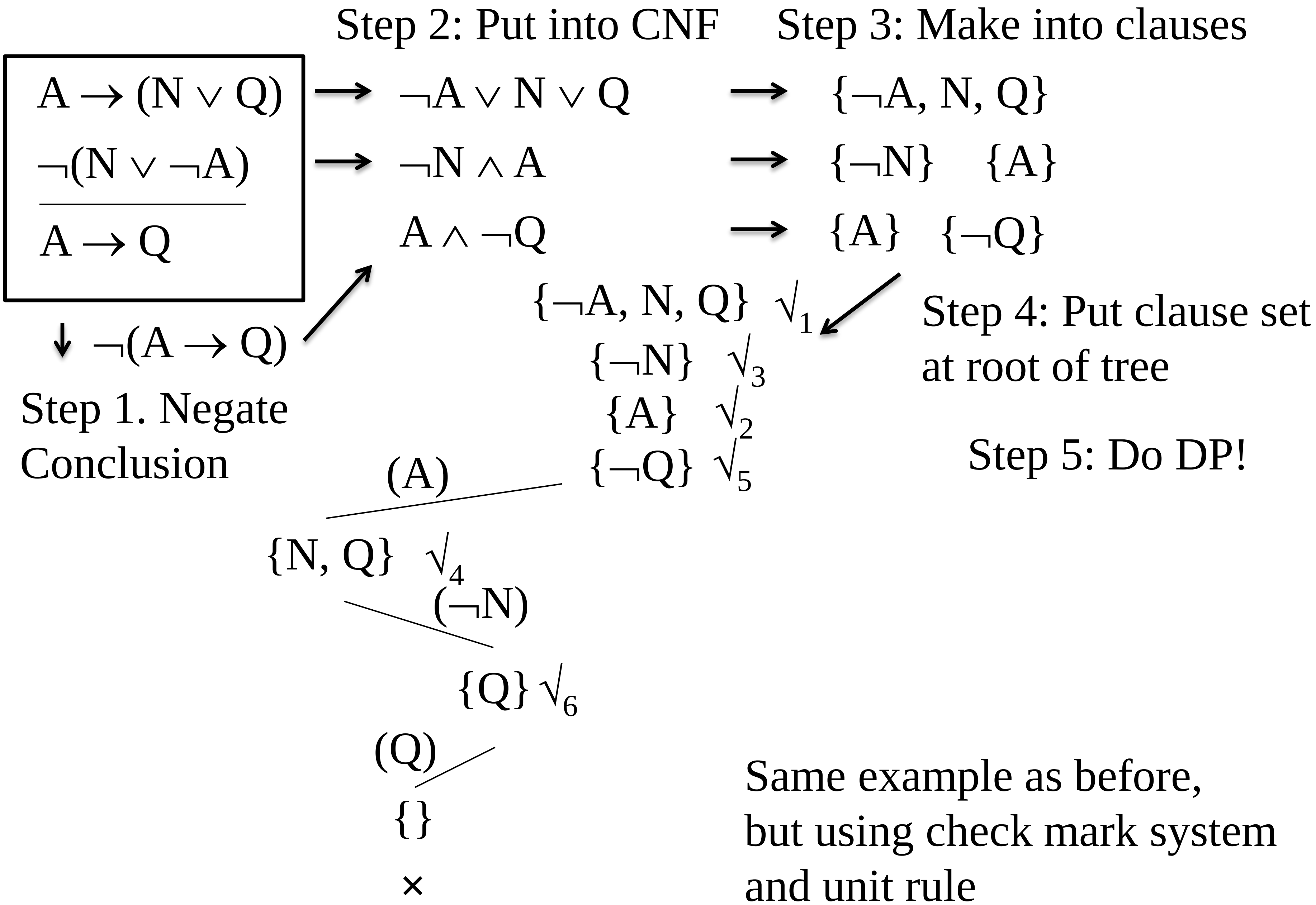
×

$(\neg A)$

$\{\}$

×

Same example as before, but using check mark system



Step 2: Put into CNF

Step 3: Make into clauses

$A \rightarrow (N \vee Q)$ → $\neg A \vee N \vee Q$ → $\{\neg A, N, Q\}$

$\neg(N \vee \neg A)$ → $\neg N \wedge A$ → $\{\neg N\}$ $\{A\}$

$A \rightarrow Q$

↓ $\neg(A \rightarrow Q)$ → $A \wedge \neg Q$ → $\{A\}$ $\{\neg Q\}$

Step 1. Negate Conclusion

Step 4: Put clause set at root of tree

$\{\neg A, N, Q\}$ $\surd_1$

$\{\neg N\}$ $\surd_3$

$\{A\}$ $\surd_2$

$\{\neg Q\}$ $\surd_5$

(A)

$\{N, Q\}$ $\surd_4$

$(\neg N)$

$\{Q\}$ $\surd_6$

(Q)

$\{\}$

×

Step 5: Do DP!

Same example as before,
but using check mark system
and unit rule

Step 2: Put into CNF Step 3: Make into clauses

$A \rightarrow (N \vee Q)$ → $\neg A \vee N \vee Q$ → $\{\neg A, N, Q\}$

$\neg(N \vee \neg A)$ → $\neg N \wedge A$ → $\{\neg N\}$ $\{A\}$

$\overline{}$

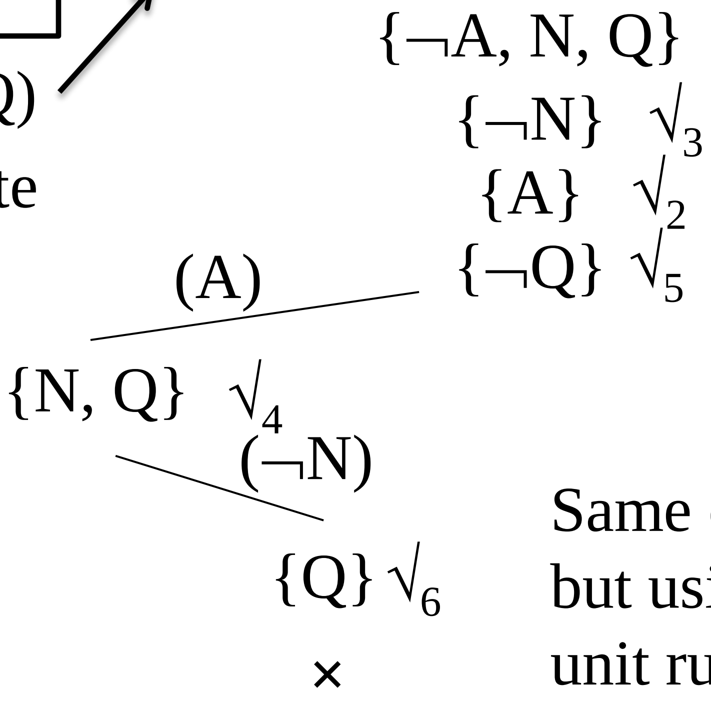
$A \rightarrow Q$ $A \wedge \neg Q$ → $\{A\}$ $\{\neg Q\}$

↓ $\neg(A \rightarrow Q)$ ↗

Step 1. Negate Conclusion

Step 4: Put clause set at root of tree

Step 5: Do DP!



$\{\neg A, N, Q\}$ $\surd_1$

$\{\neg N\}$ $\surd_3$

$\{A\}$ $\surd_2$

$\{\neg Q\}$ $\surd_5$

(A)

$\{N, Q\}$ $\surd_4$

$(\neg N)$

$\{Q\}$ $\surd_6$

×

Same example as before,
but using check mark system,
unit rule, and TT rule that any
branch with atomic P and ¬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:
  - $(A \lor B) \to (D \lor E)$ becomes
  - $(\text{True} \lor B) \to (D \lor E)$ becomes
  - $\text{True} \to (D \lor E)$ becomes
  - $D \lor E$

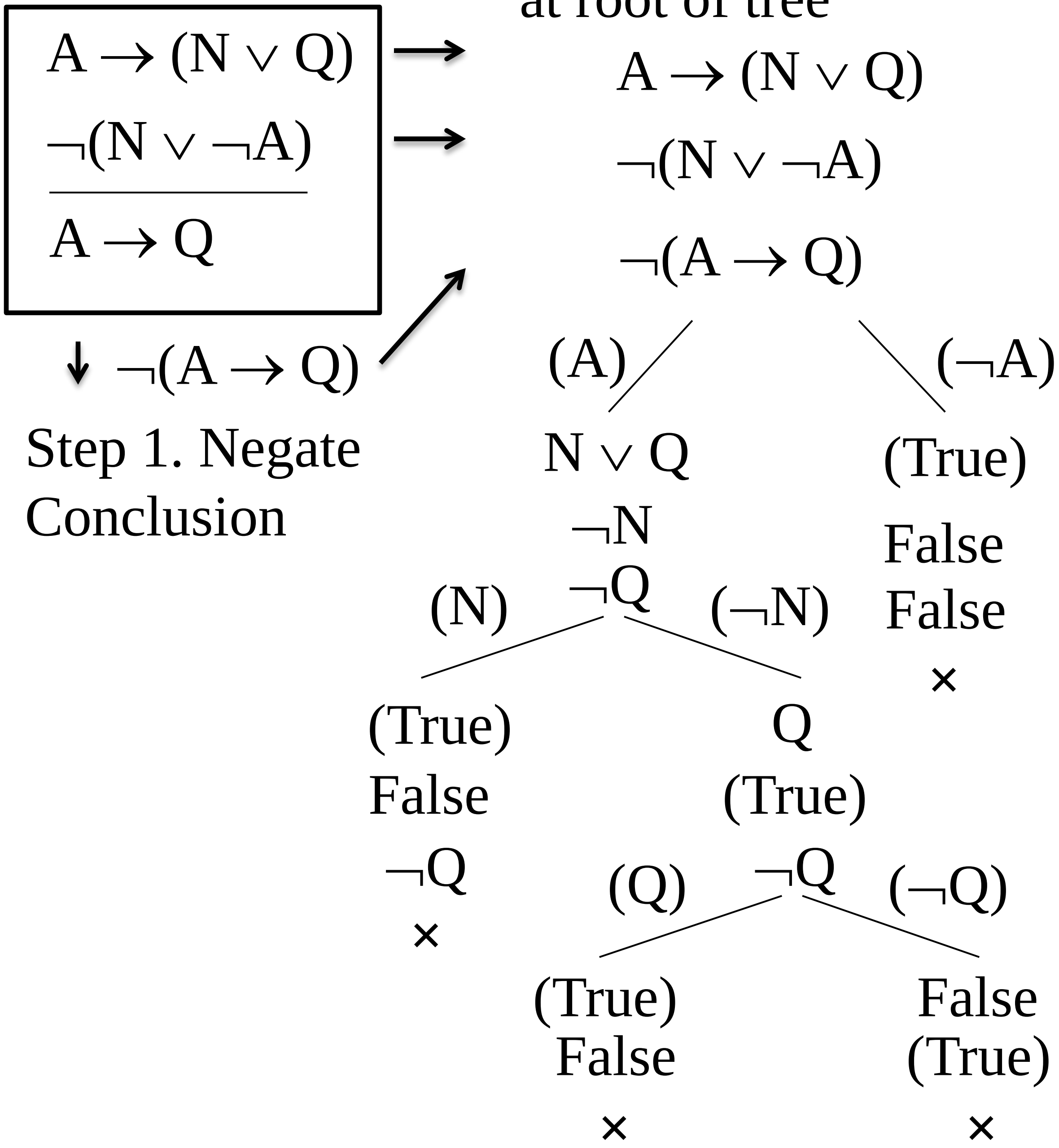
Rules for DP without CNF

$\neg$ True $\Rightarrow$ False	True $\wedge$ P $\Rightarrow$ P	True $\vee$ P $\Rightarrow$ True	True $\rightarrow$ P $\Rightarrow$ P	True $\leftrightarrow$ P $\Rightarrow$ P
$\neg$ False $\Rightarrow$ True	False $\wedge$ P $\Rightarrow$ False	False $\vee$ P $\Rightarrow$ P	False $\rightarrow$ P $\Rightarrow$ True	False $\leftrightarrow$ P $\Rightarrow$ $\neg$P
	P $\wedge$ True $\Rightarrow$ P	P $\vee$ True $\Rightarrow$ True	P $\rightarrow$ True $\Rightarrow$ True	P $\leftrightarrow$ True $\Rightarrow$ P
	P $\wedge$ False $\Rightarrow$ False	P $\vee$ False $\Rightarrow$ P	P $\rightarrow$ False $\Rightarrow$ $\neg$P	P $\leftrightarrow$ False $\Rightarrow$ $\neg$P

Step 1. Negate Conclusion

Step 2: Put statements at root of tree

Step 3: Do DP!



$A \rightarrow (N \vee Q)$

$\neg(N \vee \neg A)$

$A \rightarrow Q$

$\downarrow \neg(A \rightarrow Q)$

$A \rightarrow (N \vee Q)$

$\neg(N \vee \neg A)$

$\neg(A \rightarrow Q)$

(A)

$N \vee Q$

$\neg N$

$\neg Q$

(N)

(True)

False

$\neg Q$

×

$(\neg N)$

Q

(True)

$\neg Q$

(Q)

(True)

False

×

$(\neg Q)$

False

(True)

×

$(\neg A)$

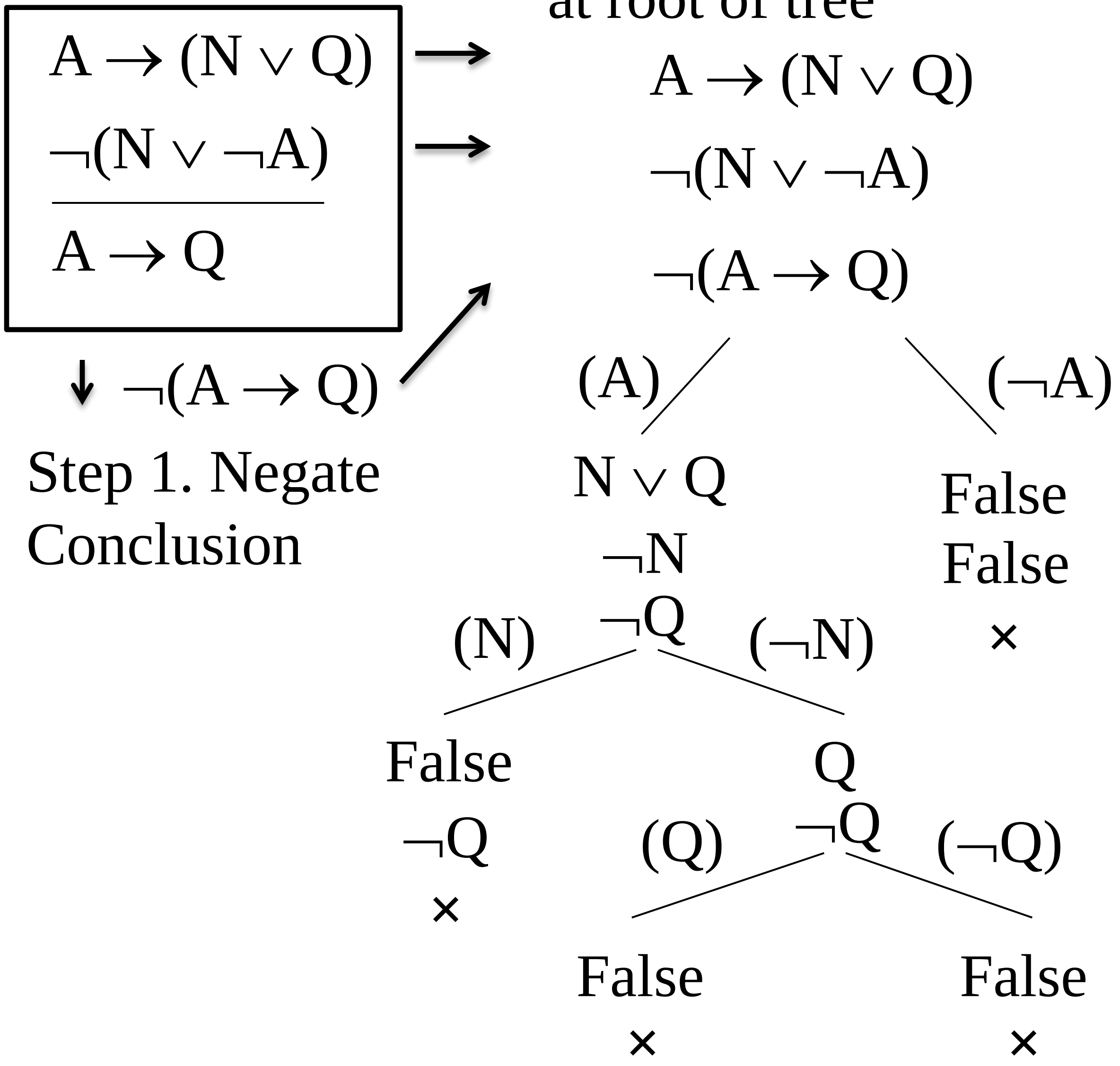
(True)

False

False

×

Step 2: Put statements at root of tree



$A \rightarrow (N \vee Q)$

$\neg(N \vee \neg A)$

$A \rightarrow Q$

$\neg(A \rightarrow Q)$

Step 1. Negate Conclusion

$A \rightarrow (N \vee Q)$

$\neg(N \vee \neg A)$

$\neg(A \rightarrow Q)$

(A)

$N \vee Q$

$\neg N$

$\neg Q$

(N)

False

$\neg Q$

×

$(\neg N)$

Q

$\neg Q$

(Q)

False

×

$(\neg Q)$

False

×

$(\neg A)$

False

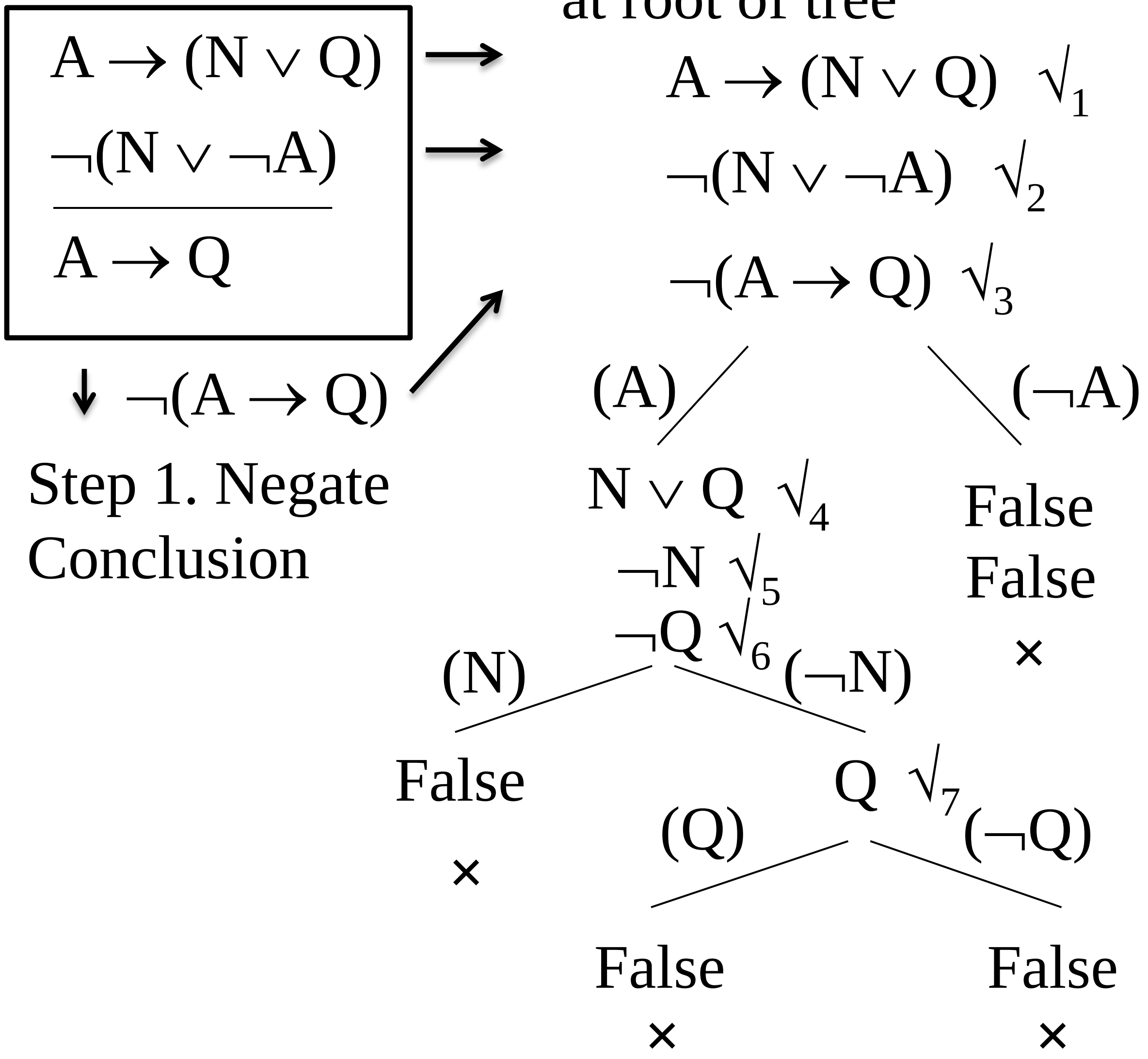
False

×

Step 3: Do DP!

Same example, but leaving out the True’s

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 1. Negate Conclusion

$A \to (N \lor Q)$ $\surd_1$

$\neg(N \lor \neg A)$ $\surd_2$

$\neg(A \to Q)$ $\surd_3$

(A)

$N \lor Q$ $\surd_4$

$\neg N$ $\surd_5$

$\neg Q$ $\surd_6$

(N)

False

×

($\neg$N)

Q $\surd_7$

(Q)

False

×

($\neg$Q)

False

×

($\neg$A)

False

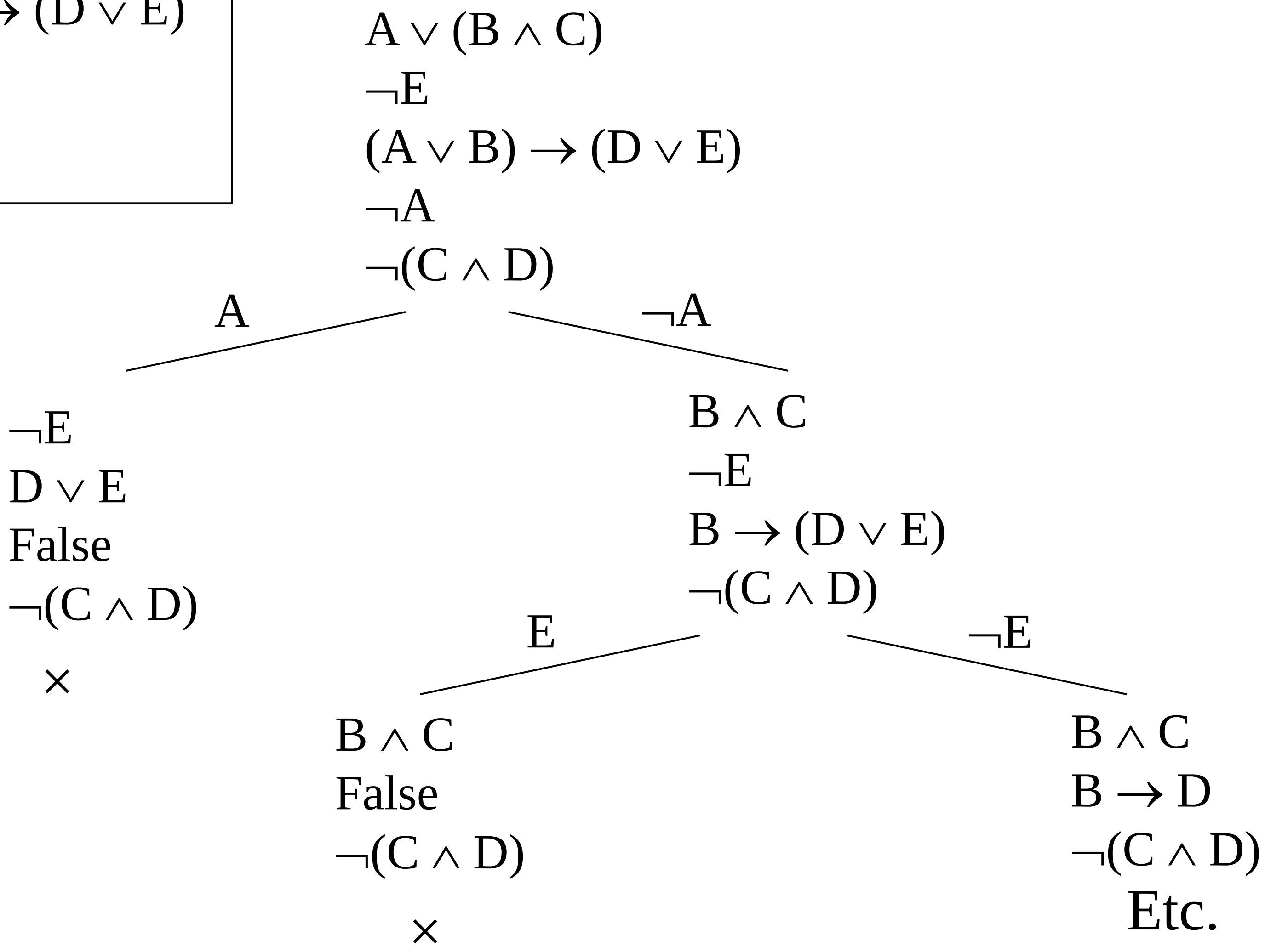
False

×

Step 3: Do DP!

Same example, but using check mark system

17.42 from LPL:
$A \vee (B \wedge C)$
$\neg E$
$(A \vee B) \rightarrow (D \vee E)$
$\neg A$
$\therefore\ C \wedge D$



$A \vee (B \wedge C)$
$\neg E$
$(A \vee B) \rightarrow (D \vee E)$
$\neg A$
$\neg(C \wedge D)$

A

$\neg E$
$D \vee E$
False
$\neg(C \wedge D)$
×

$\neg A$

$B \wedge C$
$\neg E$
$B \rightarrow (D \vee E)$
$\neg(C \wedge D)$

E

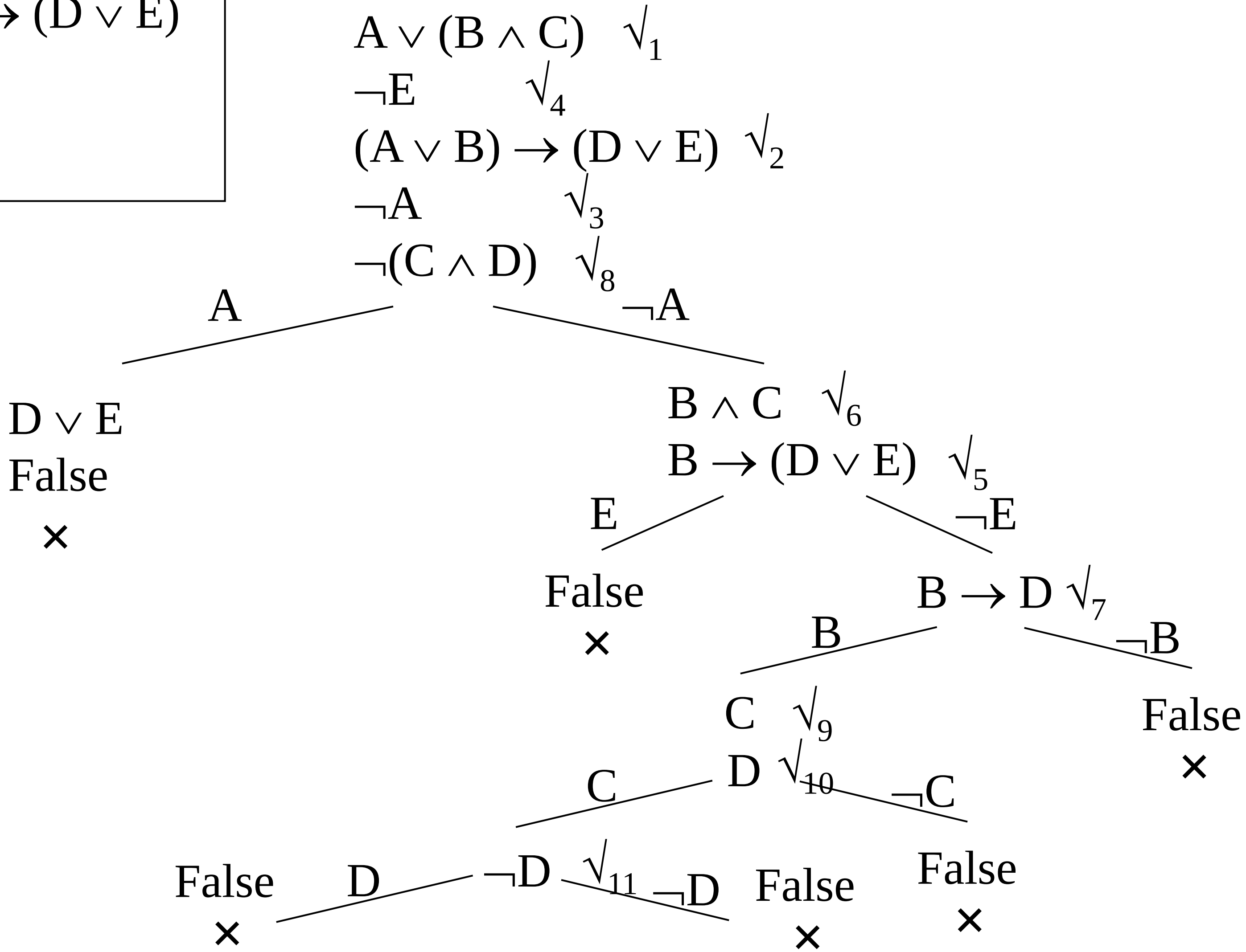
$B \wedge C$
False
$\neg(C \wedge D)$
×

$\neg E$

$B \wedge C$
$B \rightarrow D$
$\neg(C \wedge D)$
Etc.

17.42 from LPL:
$A \vee (B \wedge C)$
$\neg E$
$(A \vee B) \rightarrow (D \vee E)$
$\neg A$
$\therefore\ C \wedge D$

Using check marks:



$A \vee (B \wedge C)$ $\surd_1$
$\neg E$ $\surd_4$
$(A \vee B) \rightarrow (D \vee E)$ $\surd_2$
$\neg A$ $\surd_3$
$\neg(C \wedge D)$ $\surd_8$

A
$D \vee E$
False
×

$\neg A$
$B \wedge C$ $\surd_6$
$B \rightarrow (D \vee E)$ $\surd_5$

E
False
×

$\neg E$
$B \rightarrow D$ $\surd_7$

B
C $\surd_9$
D $\surd_{10}$

C
$\neg D$ $\surd_{11}$

D
False
×

$\neg D$
False
×

$\neg C$
False
×

$\neg B$
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

$\neg A \to B \quad \surd_4$

$C \to (D \lor E) \quad \surd_2$

$D \to \neg C \quad \surd_3$

$A \to \neg E$

$\neg(C \to B) \quad \surd_1$

C

$\neg B$

$D \lor E \quad \surd_6$

$\neg D$

$\neg\neg A \quad \surd_5$

A

E

$\neg E$

$\times_7$

17.43 from LPL:

$\neg A \to B$

$C \to (D \lor E)$

$D \to \neg C$

$A \to \neg E$

$\therefore C \to B$

1. TT rule: decompose $\neg(C \to B)$
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

$Q \lor \neg S$
$(P \land Q) \leftrightarrow R$
$\neg S \rightarrow R$

$\neg P \rightarrow (Q \leftrightarrow S)$

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?