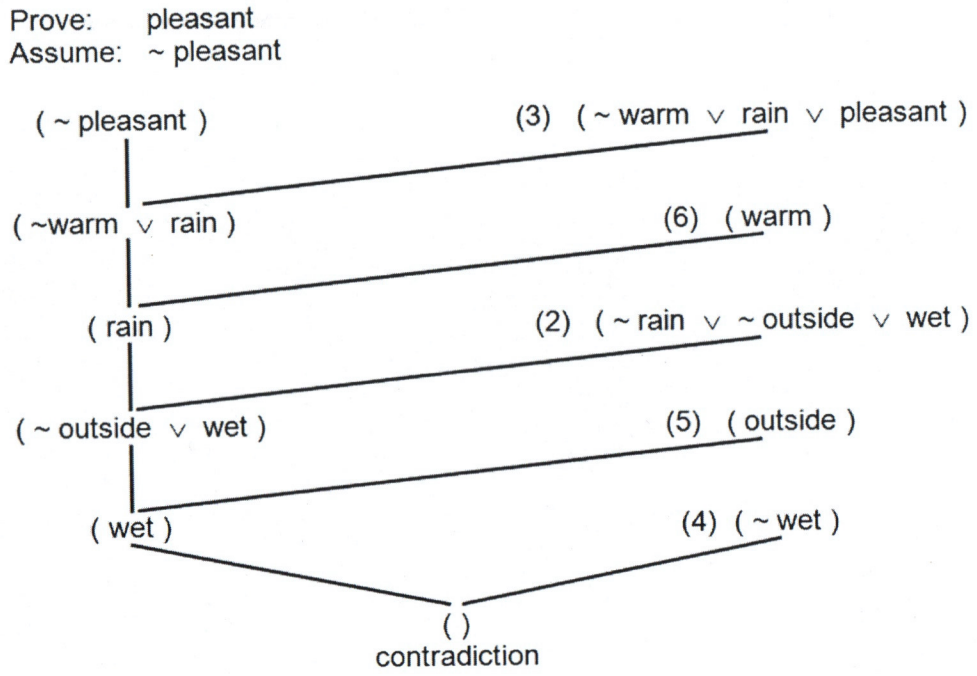
CS 4100: Introduction to AI

Wayne Snyder
Northeastern University

Lecture 3: Refutational Theorem Proving



Propositional Logic: Theorem Proving

In this class we are mostly concerned with how to use logic to prove theorems from an existing set of assertions; we assume that a set KB ("knowledge base") of formulae is true, and then try to prove that a formula Q ("query) is a logical consequence (is *entailed* by) KB.

Our goal is essentially semantic ("model checking"):

$$KB \models Q$$

HUMAN THINKING

but it can be accomplished by entire syntactic means ("deduction"):

$$KB \vdash Q$$

COMPUTER PROCESSING

If we use a sound and complete set of deduction rules, then these are equivalent:

$$KB \models Q \iff KB \vdash Q$$

$\Leftarrow$ SOUNDNESS

$\Rightarrow$ COMPLETENESS

COMPLETENESS IS GENERALLY HARD TO PROVE!

Propositional Logic: Automated Theorem Proving

If we are trying to prove theorems using a computer, there are basically two approaches:

"FORWARD CHAINING"

Forward-Reasoning: Use a proof calculus to reason from KB to Q.

This is probably the most common method people use in mathematics, often called "direct proofs" or "proofs by construction." You reason from what you know toward the statement you are trying to prove.

Example:

Theorem 1. *If a and b are consecutive integers, then the sum $a + b$ is odd.*

Proof. Assume that a and b are consecutive integers. Because a and b are consecutive we know that $b = a + 1$. Thus, the sum $a + b$ may be re-written as $2a + 1$. Thus, there exists a number k such that $a + b = 2k + 1$ so the sum $a + b$ is odd. □

KB

$b = a + 1$

PLUS BASIC MATH FACTS (TAUTOLOGIES) SUCH AS

a EVEN

$\Rightarrow a = 2k$ FOR SOME k

Propositional Logic: Forward Reasoning Proofs

If we are trying to prove theorems using a computer, there are basically two approaches: Direct proofs or proofs by contradiction.

Direct Proof: Use a proof calculus to reason from KB to Q.

EXAMPLE OF FORWARD-REASONING

Example:

Proof Calculus

Modus Ponens: $\dfrac{A \quad A \Rightarrow B}{B}$

(CONTRAPOSITION) Modus Tollens: $\dfrac{A \Rightarrow B \quad \neg B}{\neg A}$

Hypothetical Syllogism: $\dfrac{A \Rightarrow B \quad B \Rightarrow C}{A \Rightarrow C}$

Disjunctive Syllogism: $\dfrac{A \vee B \quad \neg B}{A}$

Conjunction: $\dfrac{A \quad B}{A \vee B}$

Simplification: $\dfrac{A \wedge B}{A}$

... etc. ....

KB

Problem: John is a programmer. Programmers are either lazy or hard-working. If a person is hard-working then they are successful and happy. John is not happy.

Q: **Prove:** John is lazy.

See here for a complete exposition of a sound and complete calculus for propositional logic.

Propositional Logic: Automated Theorem Proving

Problem: John is a programmer. Programmers are either lazy or hard-working. If a person is hard-working then they are successful and happy. John is not happy.

Prove: John is lazy.

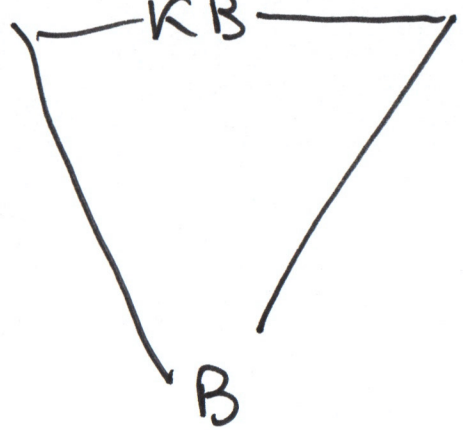
Represent problem in propositional logic:

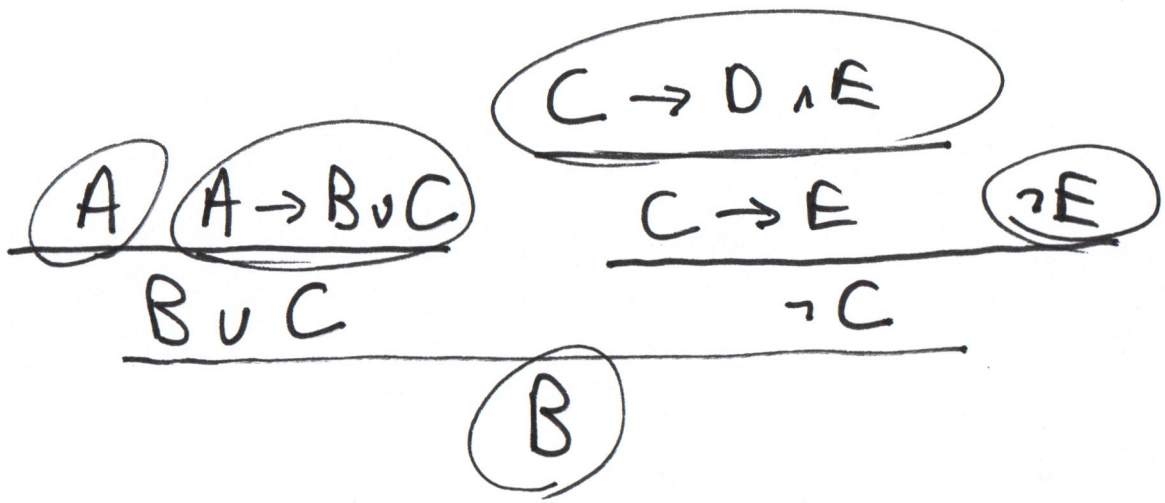
A = Programmer
B = Lazy
C = Hard-Working
D = Successful
E = Happy

$$KB = \{ A,\ A \Rightarrow B \vee C,\ C \Rightarrow D \wedge E,\ \neg E \}$$

$$Q = B$$

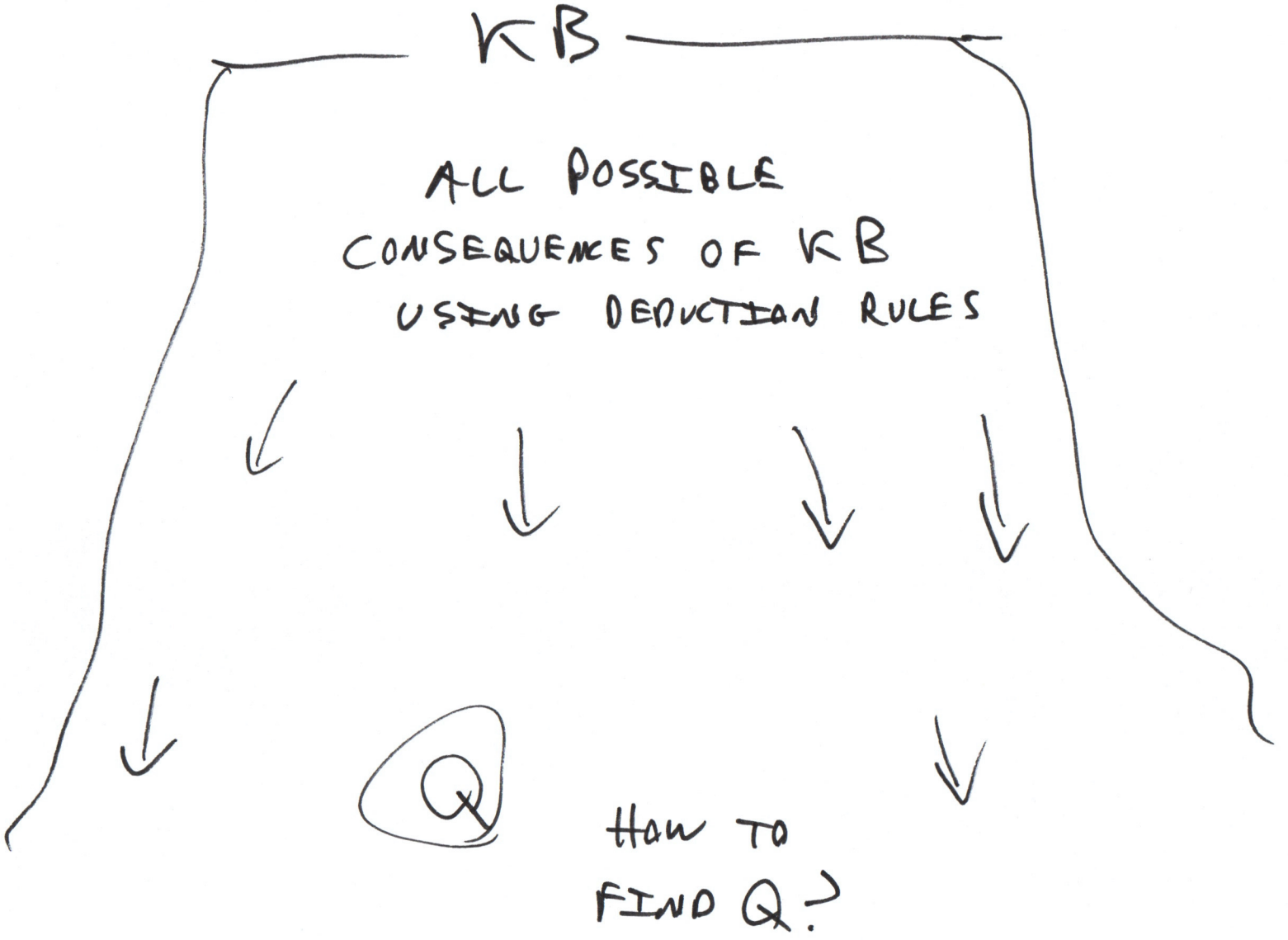
Reason from assertions to theorem:





Propositional Logic: Automated Theorem Proving

Theorem proving as search:



Propositional Logic: Refutational Theorem Proving

The second (and usual) way to prove theorems by computer is based on the following theorem:

BOTH FALSE EXACTLY WHEN I SAT. KB BUT NOT Q

Theorem 2.3 (Proof by contradiction) *$KB \models Q$ if and only if $KB \wedge \neg Q$ is unsatisfiable.*

This should be quite familiar to you as a standard proof technique in math: To prove some assertion, assume it is not true, the derive a contradiction, e.g., the classic proof that the square root of 2 is irrational:

Suppose $\sqrt{2}$ is **rational.** That means it can be written as the ratio of two integers p and q

$$\sqrt{2} = \frac{p}{q} \tag{1}$$

where we may assume that p **and** q **have no common factors.** (If there are any common factors we cancel them in the numerator and denominator.) Squaring in (1) on both sides gives

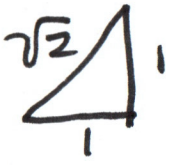
$$2 = \frac{p^2}{q^2} \tag{2}$$

which implies

$$p^2 = 2q^2 \tag{3}$$

Thus p^2 is even. The only way this can be true is that p itself is even. But then p^2 is actually divisible by 4. Hence q^2 and therefore q must be even. So p and q are both even which is a contradiction to our assumption that they have no common factors. The square root of 2 cannot be rational!

DUE TO ANCIENT GREEKS. THEY NOTICED:



Propositional Logic: Refutational Theorem Proving

This is typically called Refutational Theorem Proving. In contrast with the direct method, which reasons forward from KB to Q, we reason backward from ¬ Q to show a contradiction with the assumption that KB is satisfiable.

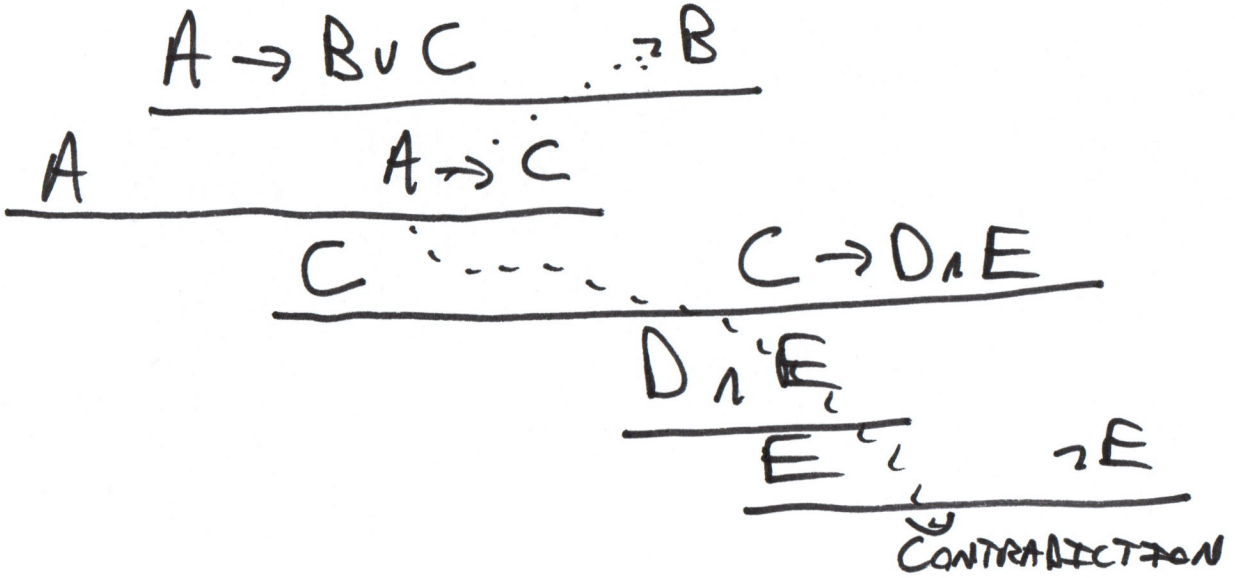
(Compare Modus Tollens: $\dfrac{A \Rightarrow B \quad \neg B}{\neg A}$.)

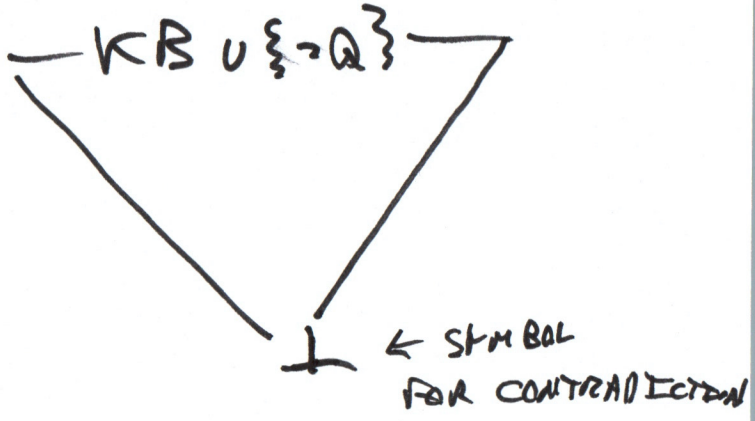
We can adapt a forward-reasoning calculus to reason backwards:

$\neg Q = \neg B$

$KB = \{ A,\ A \Rightarrow B \vee C,\ C \Rightarrow D \wedge E,\ \neg E \}$

$Q = B$





Propositional Logic: Resolution

But there is a very simple calculus involving only 2 rules that works well, based on a variation of Modus Tollens called Disjunctive Syllogism:

Disjunctive Syllogism: $\dfrac{A \vee B \quad \neg B}{A}$

Resolution:

"COMPLEMENTARY LITERALS"

$$\frac{(A_1 \vee \cdots \vee A_m \vee B), \quad (\neg B \vee C_1 \vee \cdots \vee C_n)}{(A_1 \vee \cdots \vee A_m \vee C_1 \vee \cdots \vee C_n)}.$$

(GENERALIZATION)

SIMILAR TO OTHER RULES FOR $\Rightarrow$

RESOLUTION IS SOUND

2 CASES

I MAKES B TRUE:
THEN I MAKES ONE OF C_j TRUE

I MAKES B FALSE
THEN I MAKES ONE OF A_j TRUE

$$\frac{A \quad A \to B}{B} \equiv \frac{A \quad \neg A \vee B}{B}$$

$$\frac{A \to B \quad B \to C}{A \to C} \equiv \frac{\neg A \vee B \quad \neg B \vee C}{\neg A \vee C}$$

Propositional Logic: Resolution

In order to use Resolution, we need to convert all formulae to Conjunctive Normal Form (as in HW 01).

CONNECTIVES ARE STRATIFIED:

Some terminology:

A Literal is a symbol or its negation: $A_i \quad \neg A_i$

A Clause is a disjunction of literals: $(L_1 \vee L_2 \vee \cdots L_m)$

A CNF is a conjunction of clauses: $(C_1 \wedge C_2 \wedge \cdots C_n)$

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

Propositional Logic: Resolution

Technically, we would need to add two rules to Resolution to make it complete:

Tautology: $\dfrac{A \vee A}{A}$

Contradiction: $\dfrac{A \neg A}{\bot}$] LAST STEP

CAN'T HAVE $(A \vee A)$

$\dfrac{(A \vee A) \quad (\neg A \vee \neg A)}{(A \vee \neg A)}$

CAN NEVER PRODUCE ANYTHING BUT CLAUSES WITH 2 LITERALS.

However, since we are going to implement resolution using a computer, it is convenient to use sets to represent clauses and CNFs:

Clause: $\{L_1, L_2, \cdots, L_m\} \quad \equiv \quad (L_1 \vee L_2 \vee \cdots L_m)$

CNF: $\{ \{L_1, L_2, \cdots, L_{m_1}\}, \{L_1, L_2, \cdots, L_{m_2}\}, \cdots, \{L_1, L_2, \cdots, L_{m_n}\} \}$

$\equiv$

$\big((L_1 \vee L_2 \vee \cdots \vee L_{m_1}) \wedge (L_1 \vee L_2 \vee \cdots \vee L_{m_2}) \wedge \cdots \wedge (L_1 \vee L_2 \vee \cdots \vee L_{m_n}) \big)$

Note: We have a useful encoding of literals as integers for computer implementation, but we will not use this in examples.

Propositional Logic: Resolution

However, since we are going to implement resolution using a computer, it is convenient to use sets to represent clauses and CNFs:

Clause: $\{L_1, L_2, \cdots, L_m\} \quad \equiv \quad (L_1 \vee L_2 \vee \cdots L_m)$

EMPTY CLAUSE IS CONTRADICTION

$\{\} \equiv \bot$

CNF: $\{ \{L_1, L_2, \cdots, L_{m_1}\}, \{L_1, L_2, \cdots, L_{m_2}\}, \cdots, \{L_1, L_2, \cdots, L_{m_n}\} \}$

$\equiv$

$\big((L_1 \vee L_2 \vee \cdots \vee L_{m_1}) \wedge (L_1 \vee L_2 \vee \cdots \vee L_{m_2}) \wedge \cdots \wedge (L_1 \vee L_2 \vee \cdots \vee L_{m_n}) \big)$

Note: We have a useful encoding of literals as integers for computer implementation, but we will not use this in examples.

NOTE: I SATISFIES CLAUSE C IF MAKES ≥ 1 LITERAL TRUE.

IF NO LITERALS, IS UNSATISFIABLE!

Propositional Logic: Resolution

Using this data structure, the Resolution rule alone is sound and complete calculus for propositional logic.

Tautology: $\dfrac{A \vee A}{A}$

Contradiction: $\dfrac{A \neg A}{\bot}$

A clause can NOT have duplicate literals: ~~$\{\cdots, A, A, \cdots\}$~~

The empty clause can represent contradiction: $\{\}$

(OR SET OF FMLAS)

ANY FORMULA CAN BE TURNED INTO A SET OF CLAUSES.

ONE FORMULA (SUCH AS Q) MIGHT TURN INTO A SET:

$C \wedge (D \vee A)$

$\Downarrow$

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

$\{\{C, D\}, \{C, A\}\}$

Also: Note that both KB and Q are now sets of clauses!

$\{A \vee B, C \wedge (D \vee \neg A), \neg D\}$

$\hookrightarrow (A \vee B) \wedge C \wedge (D \vee \neg A) \wedge \neg D\}$

$\hookrightarrow \{\{A, B\}, \{C\}, \{D, \neg A\}, \{\neg D\}\}$

COMMA IN CLAUSE MEANS $\vee$

" " CLAUSE SET " $\wedge$

Propositional Logic: Resolution

Let's work our previous example using Resolution:

$KB = \{A,\ A \to B \vee C,\ C \to D \wedge E,\ \neg E\}$

$Q = B$

Convert to CNF:

$A \to B \vee C \equiv \neg A \vee B \vee C$

$C \to D \wedge E \equiv \neg C \vee (D \wedge E)$

$\equiv (\neg C \vee D) \wedge (\neg C \vee E)$

$\neg Q$

Note:

KB

$\{\{A\},\ \{\neg A, B, C\},\ \{\neg C, D\},\ \{\neg C, E\},\ \{\neg E\},\ \{\neg B\}\}$

Apply Resolution

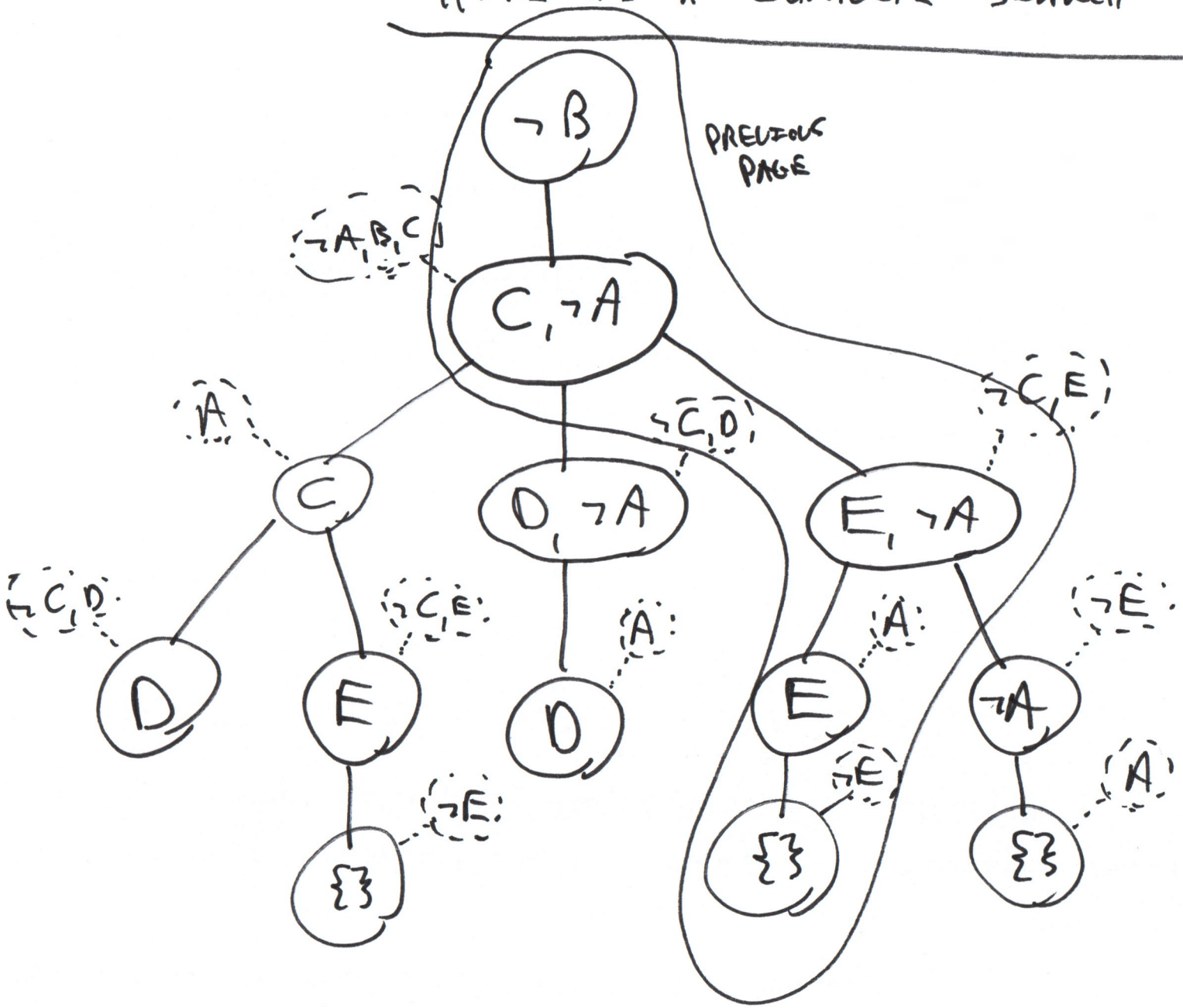
$$\frac{\{\neg A, C, B\} \quad \{\neg B\}}{\{\neg A, C\}} \qquad \frac{\{\neg A, C\} \quad \{\neg C, E\}}{\{\neg A, E\}} \qquad \frac{\{\neg A, E\} \quad \{\neg E\}}{\{\neg A\}} \qquad \frac{\{A\} \quad \{\neg A\}}{\{\} \equiv \bot}$$

(CONTRADICTION)

Propositional Logic: Resolution

Resolution theorem proving now amounts to searching for the empty clause {} from all the resolvents of $KB \cup \neg Q$. However, you only need to search among resolvents which had clauses from $\neg Q$ as ancestors (this is called the "set of support" strategy).

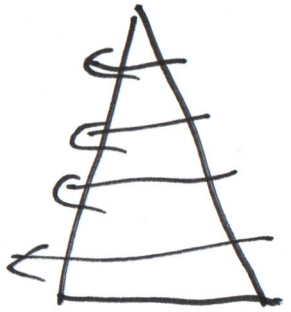
HERE IS A COMPLETE SEARCH TREE



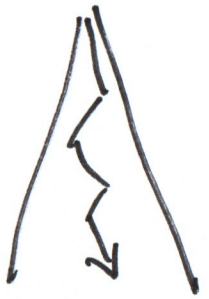
Propositional Logic: Resolution

We will study search strategies in AI in the next chapter but for now note that we can use

Breadth-First Search:



Depth-First Search:



Another strategy that is sometimes used is a combination of the two called Iterative Deepening:

USE DFS TO SUCCESSIVELY DEEPER LEVELS

Propositional Logic: Resolution

Implementation of these is straight-forward:

BFS uses a Queue:

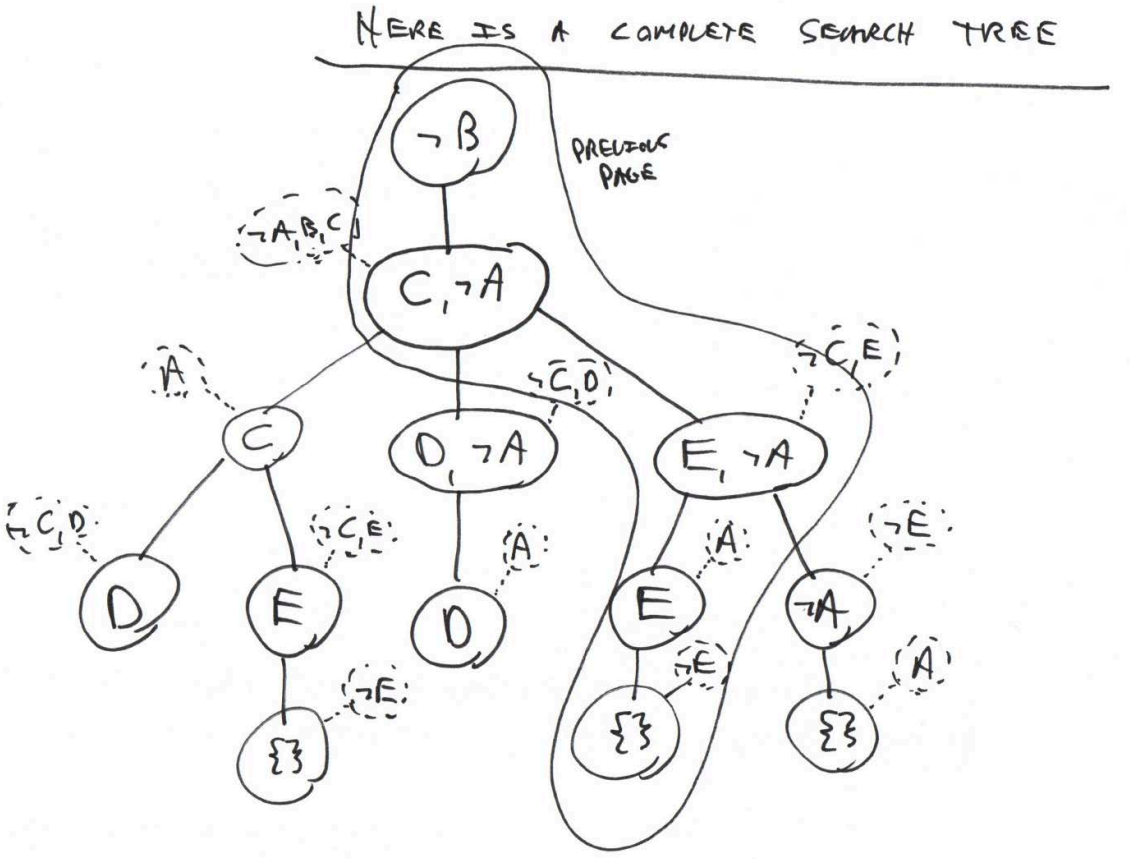
```
KB = { ... }
NQ = {C1, C2, ..., Cn}   # clause set for ¬Q
Initialize empty Queue
for q in NQ:
    Queue.enqueue(q)
while(not Queue.empty()):
    R = Queue.dequeue()
    for each resolvent A between R and (KB ∪ NQ) ):
        if (A == {}):
            Halt with success
        else:
            Queue.enqueue(A)
Halt with failure
```

DFS uses a Stack: (Same but use a stack!)

Propositional Logic: Resolution

```
KB: [ { A }, { -A, B, C }, { -C, D }, { -C, E }, { -E } ]
NQ: [ { -B } ]

Queue: [ { -B } ]
Queue: [ { C, -A } ]
Queue: [ { E, -A }, { D, -A }, { C } ]
Queue: [ { E }, { D }, { E, -A }, { D, -A } ]
Queue: [ { D }, { E }, { D }, { E, -A } ]
Queue: [ { -A }, { E }, { D }, { E }, { D } ]
Queue: [ { -A }, { E }, { D }, { E } ]
Unsatisfiable!
```



Propositional Logic: Horn Clauses

There is an important special case of Resolution which

- Can use Depth-First Search

- With some additional "bells and whistles" can be the basis for a programming language (Prolog).

Horn Clauses:

Definition 2.10 Clauses with at most one positive literal of the form

$$(\neg A_1 \vee \cdots \vee \neg A_m \vee B) \quad \text{or} \quad (\neg A_1 \vee \cdots \vee \neg A_m) \quad \text{or} \quad B$$

or (equivalently)

$$A_1 \wedge \cdots \wedge A_m \Rightarrow B \quad \text{or} \quad A_1 \wedge \cdots \wedge A_m \Rightarrow f \quad \text{or} \quad B.$$

are named *Horn clauses* (after their inventor). A clause with a single positive literal is a *fact*. In clauses with negative and one positive literal, the positive literal is called the *head*.

A Horn clause with no positives is called a goal clause, and a clause with exactly one positive literal is called a definite clause.

Propositional Logic: Horn Clauses

Important facts about Horn Clause logic:

A set of Horn clauses all of which have a positive literal (i.e., not goal clauses) is always satisfiable; in fact there is always a minimal model (which makes as few symbols true as possible).

Linear resolution (only one resolvent is produced at each step) is a complete strategy, except that which resolvent to choose is difficult to determine.
A selection rule is a rule for choosing which negative literal in the goal clause to resolve against at each step in the linear derivation.

SLD Resolution (Selection rule driven Linear resolution for Definite clauses) is the basis for Prolog.

Propositional Logic: Horn Clauses

Example of SLD Resolution with Horn Clauses.

KB

A

$B \leftarrow A$

$C \leftarrow A \wedge B$

D

$E \leftarrow C \wedge D$

$Q = E$

$\neg E$

$\neg C, \neg D$

$\neg A, \neg B, \neg D$

$\neg B, \neg D$

$\neg A, \neg D$

$\neg D$

$\bot$