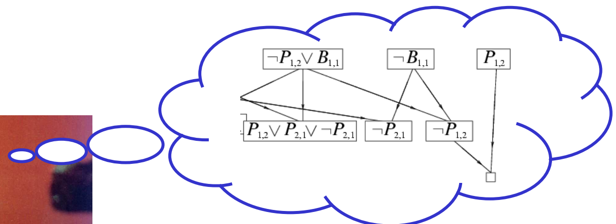
CSE 473

Chapter 7

Inference Techniques for Logical Reasoning



Inference/Proof Techniques

- Two kinds (roughly):

Model checking

- Truth table enumeration (always exponential in *n*)
- Efficient backtracking algorithms,
  e.g., Davis-Putnam-Logemann-Loveland (DPLL)
- Local search algorithms (sound but incomplete)
  e.g., randomized hill-climbing (WalkSAT)

Successive application of inference rules

- Generate new sentences from old in a sound way
- Proof = a sequence of inference rule applications
- Use inference rules as *successor function* in a
  standard search algorithm

2

Inference Technique I: Resolution

Terminology:

Literal = proposition symbol or its negation

E.g., A, $\neg A$, B, $\neg B$, etc.

Clause = disjunction of literals

E.g., $(B \vee \neg C \vee \neg D)$

Resolution assumes sentences are in *Conjunctive Normal Form (CNF):*

sentence = conjunction of clauses

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

3

Conversion to CNF

E.g., $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
3. Move $\neg$ inwards using de Morgan's rules and double-negation:
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
4. Apply distributivity law ($\wedge$ over $\vee$) and flatten:
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

   This is in CNF – Done!

4

Resolution motivation

There is a pit in [1,3] or
There is a pit in [2,2]

There is no pit in [2,2]

There is a pit in [1,3]

More generally,

$$\frac{l_1 \vee \ldots \vee l_k, \qquad \neg l_i}{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k}$$

5

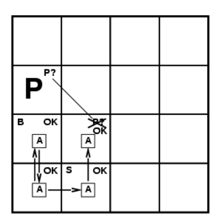
Inference Technique: Resolution

- General Resolution inference rule (for CNF):

$$\frac{l_1 \vee \ldots \vee l_k, \qquad m_1 \vee \ldots \vee m_n}{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

  where l_i and m_j are complementary literals.

  E.g., $$\frac{P_{1,3} \vee P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}}$$



- Resolution is sound
  for propositional logic

6

Resolution

Soundness of resolution inference rule:

$$\neg(l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k) \Rightarrow l_i$$

$$\neg m_j \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)$$

$$\neg(l_i \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k) \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)$$

(since $l_i = \neg m_j$)

7

Resolution algorithm

- To show **KB** $\models \alpha$, use proof by contradiction, i.e., show $KB \wedge \neg\alpha$ unsatisfiable

```
function PL-RESOLUTION(KB, α) returns true or false
   clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
   new ← { }
   loop do
      for each Ci, Cj in clauses do
         resolvents ← PL-RESOLVE(Ci, Cj)
         if resolvents contains the empty clause then return true
         new ← new ∪ resolvents
      if new ⊆ clauses then return false
      clauses ← clauses ∪ new
```

8

Resolution example

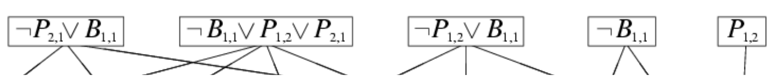
Given no breeze in [1,1], prove there's no pit in [1,2]

$KB = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$ and $\alpha = \neg P_{1,2}$

Resolution: Convert to CNF and show $KB \wedge \neg \alpha$ is unsatisfiable

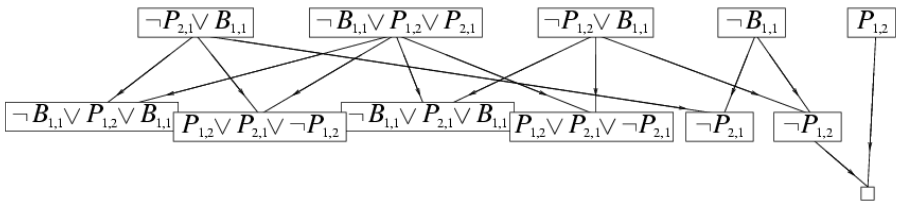
9

Resolution example



10

Resolution example



Empty clause
(i.e., KB $\wedge \neg \alpha$ unsatisfiable)

11

Inference Technique II: Forward/Backward Chaining

- Require sentences to be in **Horn Form**:
  - KB = conjunction of Horn clauses
  - Horn clause =
    - proposition symbol or
    - "(conjunction of symbols) $\Rightarrow$ symbol"
      (i.e. clause with at most 1 positive literal)
  - E.g., KB = $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$
- F/B chaining based on "Modus Ponens" rule:

$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

  Complete for Horn clauses
- Very natural and linear time complexity in size of KB

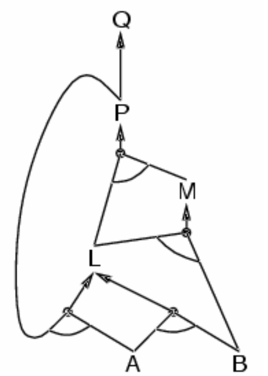
12

Forward chaining

- Idea: fire any rule whose premises are satisfied in *KB*, add its conclusion to *KB*, until query q is found

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

Query = "Is Q true?"



AND-OR Graph

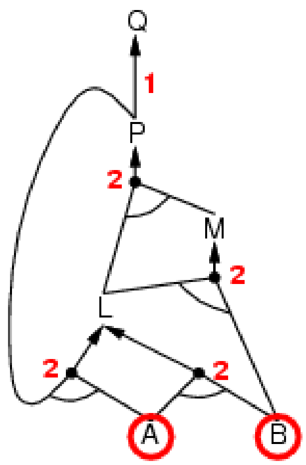
Forward chaining algorithm

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known to be true

  while agenda is not empty do
      p ← POP(agenda)
      unless inferred[p] do
          inferred[p] ← true
          for each Horn clause c in whose premise p appears do
              decrement count[c]
              if count[c] = 0 then do
                  if HEAD[c] = q then return true
                  PUSH(HEAD[c], agenda)
  return false
```

Forward chaining is sound & complete for Horn KB

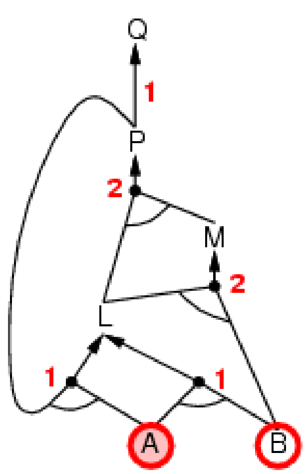
Forward chaining example



Query = Q
(i.e. "Is Q true?")

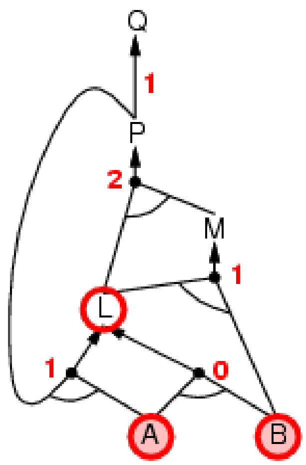
15

Forward chaining example



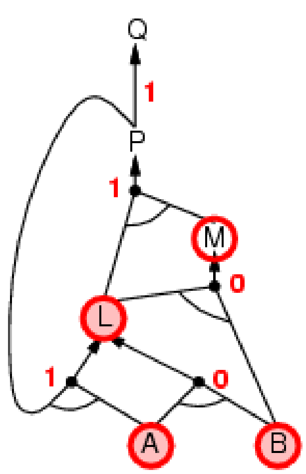
16

Forward chaining example



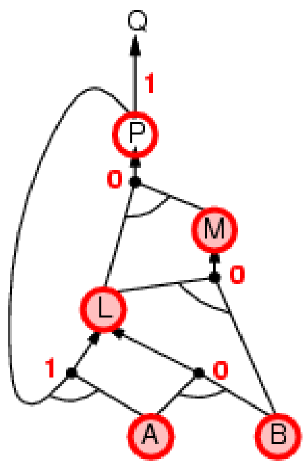
17

Forward chaining example



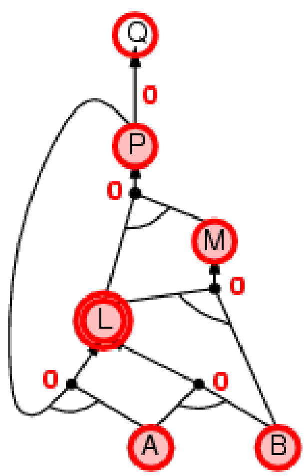
18

Forward chaining example



19

Forward chaining example



20

Backward chaining

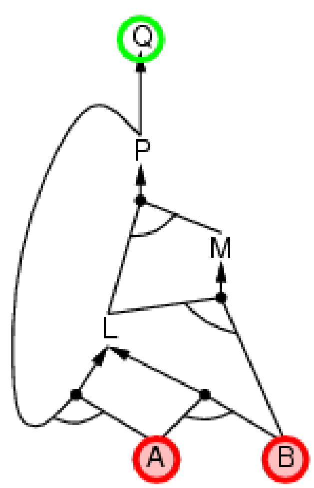
Idea: work backwards from the query *q*:
- to prove *q* by BC,
  - check if *q* is known already, or
  - prove by BC all premises of some rule concluding *q*

Avoid loops: check if new subgoal is already on goal stack

Avoid repeated work: check if new subgoal
1. has already been proved true, or
2. has already failed

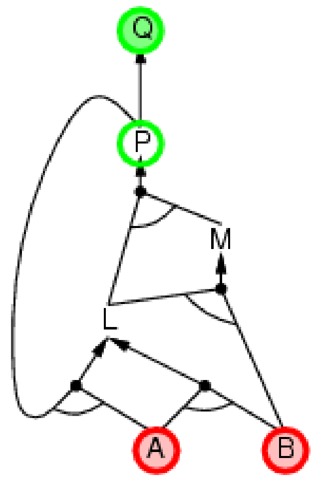
21

Backward chaining example



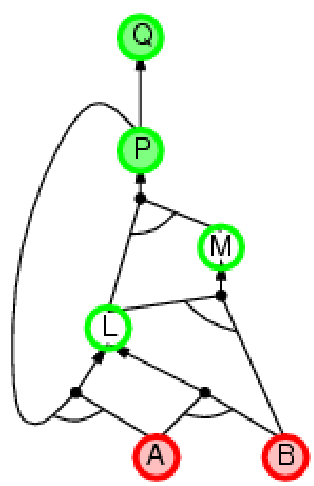
22

Backward chaining example



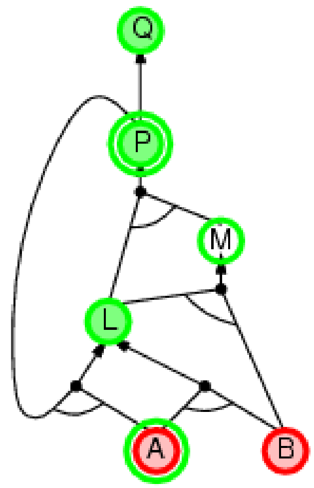
23

Backward chaining example



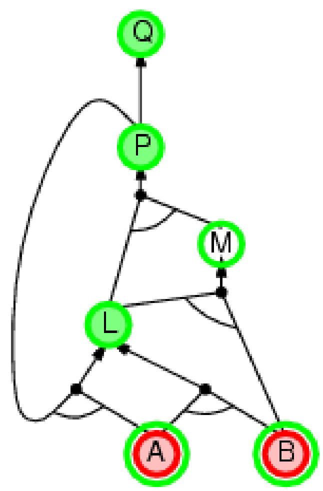
24

Backward chaining example



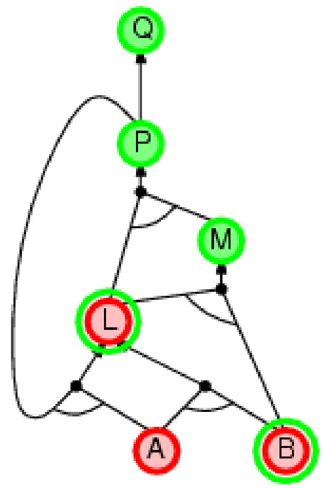
25

Backward chaining example



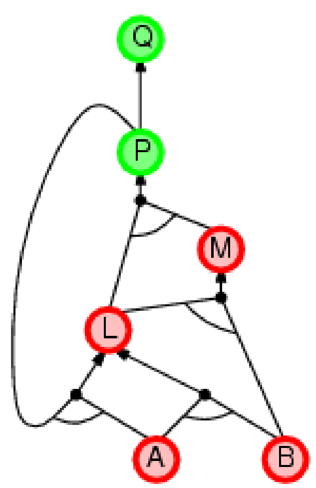
26

Backward chaining example



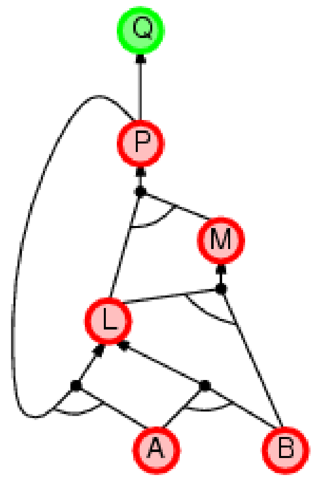
27

Backward chaining example



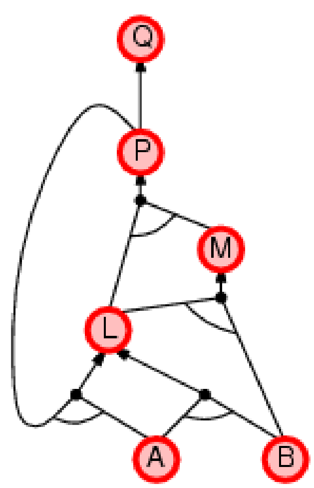
28

Backward chaining example



29

Backward chaining example



30

Forward vs. backward chaining

- FC is data-driven, automatic, unconscious processing,
  - e.g., object recognition, routine decisions
- FC may do lots of work that is irrelevant to the goal
- BC is goal-driven, appropriate for problem-solving,
  - e.g., How do I get an A in this class?
  - e.g., What is my best exit strategy out of the classroom?
  - e.g., How can I impress my date tonight?
- Complexity of BC can be much less than linear in size of KB

31

Efficient propositional inference

Two families of efficient algorithms for propositional inference based on model checking:

Complete backtracking search algorithms

- DPLL algorithm (Davis, Putnam, Logemann, Loveland)
- Similar to TT enumeration from last class but with heuristics

Incomplete local search algorithms

- WalkSAT algorithm

32

The DPLL algorithm

Determine if an input propositional logic sentence (in CNF) is satisfiable.

Improvements over truth table enumeration:

1. Early termination
   A clause is true if any literal is true.
   A sentence is false if any clause is false.

2. Pure symbol heuristic
   Pure symbol: always appears with the same "sign" in all clauses.
   e.g., In the three clauses $(A \vee \neg B)$, $(\neg B \vee \neg C)$, $(C \vee A)$, A and B are pure, C is impure.
   Make a pure symbol literal true.

3. Unit clause heuristic
   Unit clause: only one literal in the clause
   The only literal in a unit clause must be true.

33

The DPLL algorithm

```
function DPLL-SATISFIABLE?(s) returns true or false
   inputs: s, a sentence in propositional logic

   clauses ← the set of clauses in the CNF representation of s
   symbols ← a list of the proposition symbols in s
   return DPLL(clauses, symbols, [])

function DPLL(clauses, symbols, model) returns true or false

   if every clause in clauses is true in model then return true
   if some clause in clauses is false in model then return false
   P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
   if P is non-null then return DPLL(clauses, symbols–P, [P = value|model])
   P, value ← FIND-UNIT-CLAUSE(clauses, model)
   if P is non-null then return DPLL(clauses, symbols–P, [P = value|model])
   P ← FIRST(symbols); rest ← REST(symbols)
   return DPLL(clauses, rest, [P = true|model]) or
          DPLL(clauses, rest, [P = false|model])
```

34

Next Time

- WalkSAT
- HW #1 due
- Read Chapter 8

  First-Order Logic

35