Logic

Dr. Melanie Martin
CS 4480

October 22, 2012

Based on slides from
http://aima.eecs.berkeley.edu/2nd-ed/slides-ppt/

Conversion to CNF

$B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$.

   $(B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})$

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \lor \beta$.

   $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg(P_{1,2} \lor P_{2,1}) \lor B_{1,1})$

3. Move $\neg$ inwards using de Morgan's rules and double-negation:

   $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \lor \neg P_{2,1}) \lor B_{1,1})$

4. Apply distributivity law ($\land$ over $\lor$) and flatten:

   $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1})$

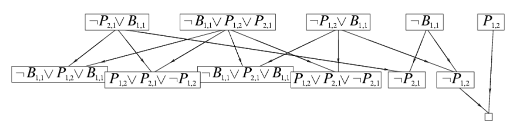
Resolution algorithm

- Proof by contradiction, i.e., show $KB \land \neg\alpha$ 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
```

Resolution example

- $KB = (B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1}$ $\alpha = \neg P_{1,2}$



Forward and backward chaining

- Horn Form (restricted)
  KB = conjunction of Horn clauses
  - Horn clause =
    - proposition symbol; or
    - (conjunction of symbols) $\Rightarrow$ symbol
  - E.g., $C \land (B \Rightarrow A) \land (C \land D \Rightarrow B)$
- Modus Ponens (for Horn Form): complete for Horn KBs

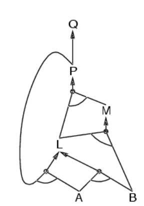
$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \land \ldots \land \alpha_n \Rightarrow \beta}{\beta}$$

- Can be used with forward chaining or backward chaining.
- These algorithms are very natural and run in linear time

Forward chaining

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

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

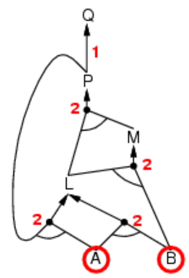


Forward chaining algorithm

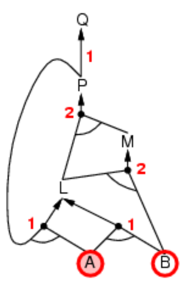
```
function PL-FC-Entails?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known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

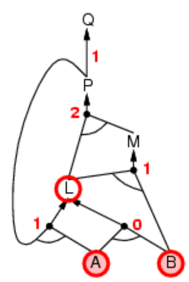
Forward chaining example



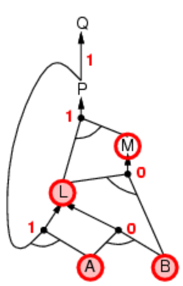
Forward chaining example



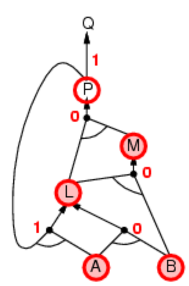
Forward chaining example



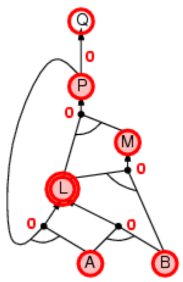
Forward chaining example



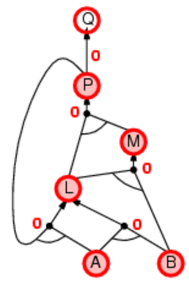
Forward chaining example



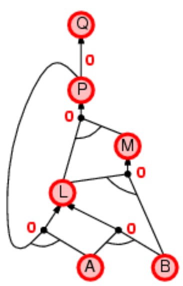
Forward chaining example



Forward chaining example



Forward chaining example



Proof of completeness

- FC derives every atomic sentence that is entailed by *KB*

1. FC reaches a fixed point where no new atomic sentences are derived
2. Consider the final state as a model *m*, assigning true/false to symbols
3. Every clause in the original *KB* is true in *m*

   $a_1 \wedge \ldots \wedge a_k \Rightarrow b$
4. Hence *m* is a model of *KB*
5. If $KB \models q$, *q* is true in every model of *KB*, including *m*

Backward chaining

Idea: work backwards from the query *q*:

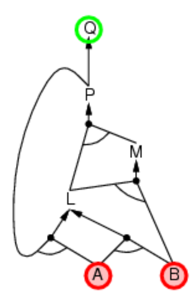
to prove *q* by BC,
check if *q* is known already, or
prove by BC all premises of some rule concluding *q*

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

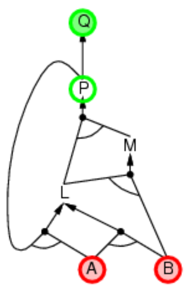
Avoid repeated work: check if new subgoal

1. has already been proved true, or
2. has already failed

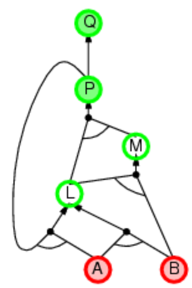
Backward chaining example



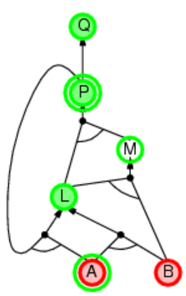
Backward chaining example



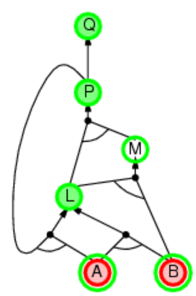
Backward chaining example



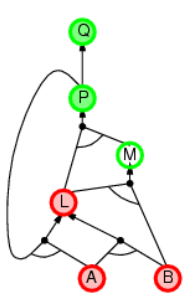
Backward chaining example



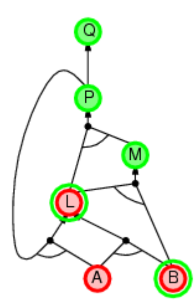
Backward chaining example



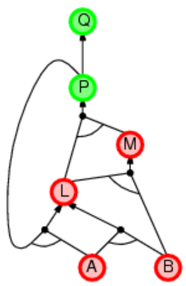
Backward chaining example



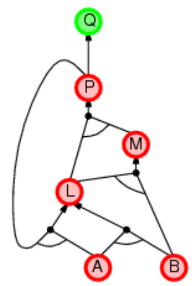
Backward chaining example



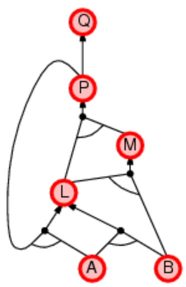
Backward chaining example



Backward chaining example



Backward chaining example



Forward vs. backward chaining

- FC is data-driven, automatic, unconscious processing,
  - e.g., object recognition, routine decisions
- May do lots of work that is irrelevant to the goal
- BC is goal-driven, appropriate for problem-solving,
  - e.g., Where are my keys? How do I get into a PhD program?
- Complexity of BC can be much less than linear in size of KB

Efficient propositional inference

Two families of efficient algorithms for propositional inference:

Complete backtracking search algorithms

- DPLL algorithm (Davis, Putnam, Logemann, Loveland)
- Incomplete local search algorithms
  - `WalkSAT` algorithm

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.

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])
```

The `WalkSAT` algorithm

- Incomplete, local search algorithm
- Evaluation function: The min-conflict heuristic of minimizing the number of unsatisfied clauses
- Balance between greediness and randomness

The `WalkSAT` algorithm

```
function WALKSAT(clauses, p, max-flips) returns a satisfying model or failure
   inputs: clauses, a set of clauses in propositional logic
           p, the probability of choosing to do a "random walk" move
           max-flips, number of flips allowed before giving up

   model ← a random assignment of true/false to the symbols in clauses
   for i = 1 to max-flips do
      if model satisfies clauses then return model
      clause ← a randomly selected clause from clauses that is false in model
      with probability p flip the value in model of a randomly selected symbol
            from clause
      else flip whichever symbol in clause maximizes the number of satisfied clauses
   return failure
```

Hard satisfiability problems

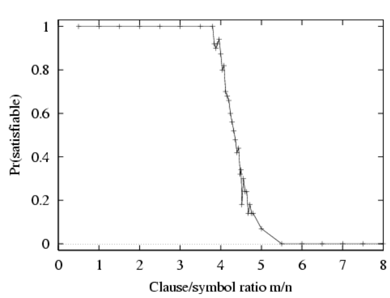
- Consider random 3-CNF sentences. e.g.,

(¬D ∨ ¬B ∨ C) ∧ (B ∨ ¬A ∨ ¬C) ∧ (¬C ∨ ¬B ∨ E) ∧ (E ∨ ¬D ∨ B) ∧ (B ∨ E ∨ ¬C)

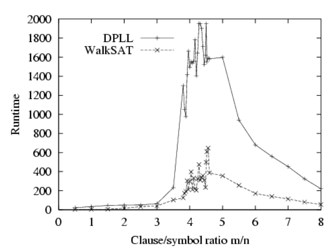
m = number of clauses
n = number of symbols

– Hard problems seem to cluster near m/n = 4.3 (critical point)

Hard satisfiability problems



Hard satisfiability problems



- Median runtime for 100 satisfiable random 3-CNF sentences, n = 50

Inference-based agents in the wumpus world

A wumpus-world agent using propositional logic:

$\neg P_{1,1}$
$\neg W_{1,1}$
$B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$
$S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y})$
$W_{1,1} \vee W_{1,2} \vee \ldots \vee W_{4,4}$
$\neg W_{1,1} \vee \neg W_{1,2}$
$\neg W_{1,1} \vee \neg W_{1,3}$
…

$\Rightarrow$ 64 distinct proposition symbols, 155 sentences

```
function PL-WUMPUS-AGENT( percept) returns an action
   inputs: percept, a list, [stench,breeze,glitter]
   static: KB, initially containing the "physics" of the wumpus world
           x, y, orientation, the agent's position (init. [1,1]) and orient. (init. right)
           visited, an array indicating which squares have been visited, initially false
           action, the agent's most recent action, initially null
           plan, an action sequence, initially empty

   update x,y,orientation, visited based on action
   if stench then TELL(KB, S_x,y) else TELL(KB, ¬ S_x,y)
   if breeze then TELL(KB, B_x,y) else TELL(KB, ¬ B_x,y)
   if glitter then action ← grab
   else if plan is nonempty then action ← POP(plan)
   else if for some fringe square [i,j], ASK(KB, (¬ P_i,j ∧ ¬ W_i,j)) is true or
           for some fringe square [i,j], ASK(KB, (P_i,j ∨ W_i,j)) is false then do
       plan ← A*-GRAPH-SEARCH(ROUTE-PB([x,y], orientation, [i,j], visited))
       action ← POP(plan)
   else action ← a randomly chosen move
   return action
```

Expressiveness limitation of propositional logic

- KB contains "physics" sentences for every single square

- For every time t and every location $[x,y]$,

$L^t_{x,y} \wedge FacingRight^t \wedge Forward^t \Rightarrow L^t_{x+1,y}$

- Rapid proliferation of clauses

Summary

- Logical agents apply inference to a knowledge base to derive new information and make decisions
- Basic concepts of logic:
  - syntax: formal structure of sentences
  - semantics: truth of sentences wrt models
  - entailment: necessary truth of one sentence given another
  - inference: deriving sentences from other sentences
  - soundness: derivations produce only entailed sentences
  - completeness: derivations can produce all entailed sentences
- Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.
- Resolution is complete for propositional logic
  Forward, backward chaining are linear-time, complete for Horn clauses
- Propositional logic lacks expressive power