Notes adapted from lecture notes for CMSC 421 by B.J. Dorr

Artificial Intelligence 1: PL

Lecturer: Tom Lenaerts

Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle (IRIDIA)

Université Libre de Bruxelles

Propositional Logic

Propositional logic is the simplest logic—illustrates basic ideas

The proposition symbols P_1, P_2 etc are sentences

If S is a sentence, $\neg S$ is a sentence (negation)

If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)

If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)

If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)

If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

Propositional Logic

Each model specifies true/false for each proposition symbol

E.g.	$P_{1,2}$	$P_{2,2}$	$P_{3,1}$
	$true$	$true$	$false$

(With these symbols, 8 possible models, can be enumerated automatically.)

Rules for evaluating truth with respect to a model m:

$\neg S$	is true iff	S	is false		
$S_1 \wedge S_2$	is true iff	S_1	is true and	S_2	is true
$S_1 \vee S_2$	is true iff	S_1	is true or	S_2	is true
$S_1 \Rightarrow S_2$	is true iff	S_1	is false or	S_2	is true
i.e.,	is false iff	S_1	is true and	S_2	is false
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$	is true and	$S_2 \Rightarrow S_1$	is true

Simple recursive process evaluates an arbitrary sentence, e.g.,
$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = true \wedge (false \vee true) = true \wedge true = true$

Propositional Logic

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

Wumpus world logic

Let $P_{i,j}$ be true if there is a pit in $[i,j]$.
Let $B_{i,j}$ be true if there is a breeze in $[i,j]$.

$$\neg P_{1,1}$$
$$\neg B_{1,1}$$
$$B_{2,1}$$

"Pits cause breezes in adjacent squares"

Wumpus world logic

Let $P_{i,j}$ be true if there is a pit in $[i,j]$.
Let $B_{i,j}$ be true if there is a breeze in $[i,j]$.

$$\neg P_{1,1}$$
$$\neg B_{1,1}$$
$$B_{2,1}$$

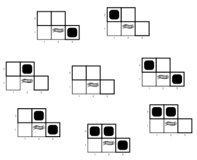
"Pits cause breezes in adjacent squares"

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$
$$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

"A square is breezy *if and only if* there is an adjacent pit"

Truth tables for inference

Enumerate the models and check that α is true in every model
In which KB is true.



$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB	α_1
false	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*true*
false	*false*	*false*	*false*	*false*	*false*	*true*	*false*	*true*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*false*	*true*
false	*true*	*false*	*false*	*false*	*false*	*true*	<u>*true*</u>	<u>*true*</u>
false	*true*	*false*	*false*	*false*	*true*	*false*	<u>*true*</u>	<u>*true*</u>
false	*true*	*false*	*false*	*false*	*true*	*true*	<u>*true*</u>	<u>*true*</u>
false	*true*	*false*	*false*	*true*	*false*	*false*	*false*	*true*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*false*

Inference by enumeration

- Depth-first enumeration of all models is sound and complete

```
function TT-ENTAILS?(KB, α) returns true or false
   symbols ← a list of the proposition symbols in KB and α
   return TT-CHECK-ALL(KB, α, symbols, [])

function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
   if EMPTY?(symbols) then
       if PL-TRUE?(KB, model) then return PL-TRUE?(α, model)
       else return true
   else do
       P ← FIRST(symbols); rest ← REST(symbols)
       return TT-CHECK-ALL(KB, α, rest, EXTEND(P, true, model)) and
                  TT-CHECK-ALL(KB, α, rest, EXTEND(P, false, model))
```

- For n symbols, time complexity is $O(2^n)$, space complexity is $O(n)$

Logical equivalence

- Two sentences are *logically equivalent* iff true in same set of models or $\alpha \equiv \beta$ iff $\alpha \models \beta$ and $\beta \models \alpha$.

$$
\begin{array}{rcll}
(\alpha \wedge \beta) & \equiv & (\beta \wedge \alpha) & \text{commutativity of } \wedge \\
(\alpha \vee \beta) & \equiv & (\beta \vee \alpha) & \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) & \equiv & (\alpha \wedge (\beta \wedge \gamma)) & \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) & \equiv & (\alpha \vee (\beta \vee \gamma)) & \text{associativity of } \vee \\
\neg(\neg\alpha) & \equiv & \alpha & \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\beta \Rightarrow \neg\alpha) & \text{contraposition} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\alpha \vee \beta) & \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) & \equiv & ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) & \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) & \equiv & (\neg\alpha \vee \neg\beta) & \text{de Morgan} \\
\neg(\alpha \vee \beta) & \equiv & (\neg\alpha \wedge \neg\beta) & \text{de Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) & \equiv & ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) & \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) & \equiv & ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) & \text{distributivity of } \vee \text{ over } \wedge
\end{array}
$$

Validity and satisfiability

- A sentence is *valid* if it is true in all models,
  - **e.g., *True*, A $\vee\neg$A, A $\Rightarrow$ A, (A $\wedge$ (A $\Rightarrow$ B)) $\Rightarrow$ B**
- Validity is connected to inference via the *Deduction Theorem*:
  - ***KB* $\models \alpha$ if and only if (*KB* $\Rightarrow \alpha$) is valid**
- A sentence is *satisfiable* if it is true in *some* model
  - **e.g., A $\vee$ B, C**
- A sentence is *unsatisfiable* if it is true in *no* models
  - **e.g., A$\wedge\neg$A**
- Satisfiability is connected to inference via the following:
  - ***KB* $\models \alpha$ if and only if (*KB* $\wedge \neg\alpha$) is unsatisfiable**
  - **Remember proof by contradiction.**

Inference rules in PL

- Modens Ponens $\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$
- And-elimination: from a conjuction any conjunction can be inferred: $\frac{\alpha \wedge \beta}{\alpha}$
- All logical equivalences of slide 39 can be used as inference rules. $\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}$

Example

- Assume R1 through R5:

$$\neg P_{1,1}, B_{1,1} \Rightarrow (P_{1,1} \vee P_{2,1}), B_{2,1} \Rightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1}), \neg B_{1,1}, B_{2,1}$$

- How can we prove $\neg P_{1,2}$?

$R_6 : (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$ **Biconditional elim.**

$R_7 : (P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}$ **And elim.**

$R_8 : \neg B_{1,1} \Rightarrow \neg (P_{1,2} \vee P_{2,1})$ **Contraposition**

$R_9 : \neg (P_{1,2} \vee P_{2,1})$ **Modens ponens**

$R_{10} : \neg P_{1,2} \wedge \neg P_{2,1}$ **Morgan's rule**

Searching for proofs

- Finding proofs is exactly like finding solutions to search problems.
- Search can be done forward (forward chaining) to derive goal or backward (backward chaining) from the goal.
- Searching for proofs is not more efficient than enumerating models, but in many practical cases, it is more efficient because we can ignore irrelevant properties.
- *Monotonicity*: the set of entailed sentences can only increase as information is added to the knowledge base.

$$\textit{for any sentence } \alpha \textit{ and } \beta : \textit{if } KB \models \alpha \textit{ then } KB \wedge \beta \models \alpha$$

Proof methods

- Proof methods divide into (roughly) two kinds:
  - **Application of inference rules**
    - Legitimate (sound) generation of new sentences from old
    - Proof = a sequence of inference rule application can use inference rules as operators in a standard search algorithm
    - Typically require transformation of sentences into a normal form
  - **Model checking**
    - truth table enumeration (always exponential in n)
    - improved backtracking, e.g., Davis--Putnam-Logemann-Loveland (DPLL)
    - heuristic search in model space (sound but incomplete)

      e.g., min-conflicts-like hill-climbing algorithms

Resolution

- Start with Unit Resolution Inference Rule:

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k}$$

- Full Resolution Rule is a generalization of this rule:

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

- For clauses of length two:

$$\frac{\ell_1 \vee \ell_2, \qquad \neg \ell_2 \vee \ell_3}{\ell_1 \vee \ell_3}$$

Resolution in Wumpus world

- At some point we can derive the absence of a pit in square 2,2:

$$\neg P_{1,1}, B_{1,1} \Rightarrow (P_{1,1} \vee P_{2,1}), B_{2,1} \Rightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1}), \neg B_{1,1}, B_{2,1}, \neg P_{2,2}, \neg P_{3,1}$$

- Now after biconditional elimination of R3 followed by a modens ponens with R5:

$$R_{15} : (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

- Resolution : $\dfrac{P_{1,1} \vee P_{2,2} \vee P_{3,1} \qquad \neg P_{2,2}}{P_{1,1} \vee P_{3,1}}$

Resolution

- Uses CNF (Conjunctive normal form)
  - **Conjunction of disjunctions of literals (clauses)**
- The resolution rule is sound:
  - **Only entailed sentences are derived**
- Resolution is complete in the sense that it can always be used to either confirm or refute a sentence (it can not be used to enumerate true sentences.)

Conversion to CNF

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

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

Resolution algorithm

- Proof by contradiction, i.e., show $KB \wedge \neg \alpha$ unsatisfiable

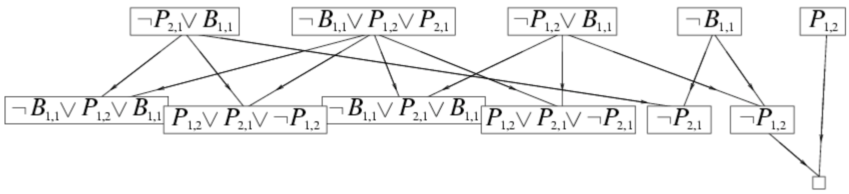
```
function PL-RESOLUTION(KB, α) returns true or false
   clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
   new ← { }
   loop do
       for each Ci, Cj in clauses do
           resolvents ← PL-RESOLVE(Ci, Cj)
           if resolvents contains the empty clause then return true
   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


Resolution algorithm

- First $KB \wedge \neg \alpha$ is converted into CNF
- Then apply resolution rule to resulting clauses.
- The process continues until:
  - **There are no new clauses that can be added**
    - Hence α **does not** ential β
  - **Two clauses resolve to entail the empty clause.**
    - Hence α **does** ential β

Resolution example

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



Forward and backward chaining

- The completeness of resolution makes it a very important inference model.
- Real-world knowledge only requires a restricted form of clauses:
  - **Horn clauses = disjunction of literals with at most one positive literal**
  - **Three important properties**
    - Can be written as an implication

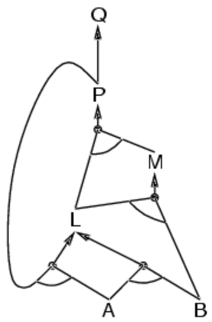
$$\left(\neg L_{1,1} \vee \neg Breeze \vee B_{1,1}\right) \rightarrow \left(L_{1,1} \vee Breeze\right) \Rightarrow B_{1,1}$$

    - Inference through forward chaining and backward chaining.
    - Deciding entailment can be done in a time linear size of the knowledge base.

Forward chaining

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

$$
\begin{aligned}
&P \Rightarrow Q\\
&L \wedge M \Rightarrow P\\
&B \wedge L \Rightarrow M\\
&A \wedge P \Rightarrow L\\
&A \wedge B \Rightarrow L\\
&A\\
&B
\end{aligned}
$$



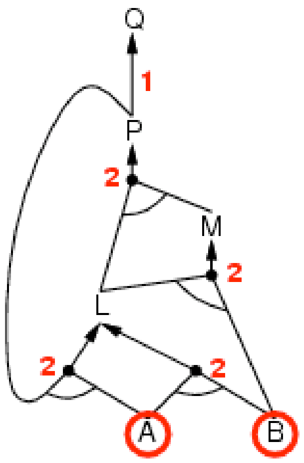
Forward chaining algorithm

- Forward chaining is sound and complete for Horn KB

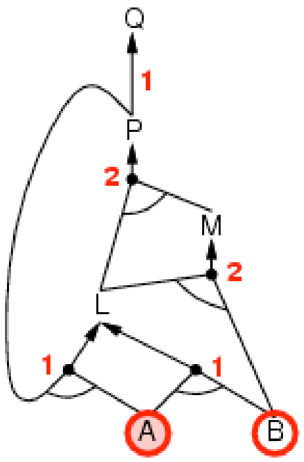
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known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

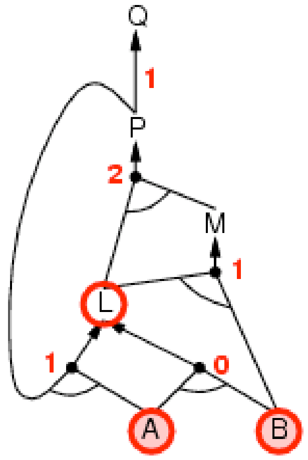
Forward chaining example



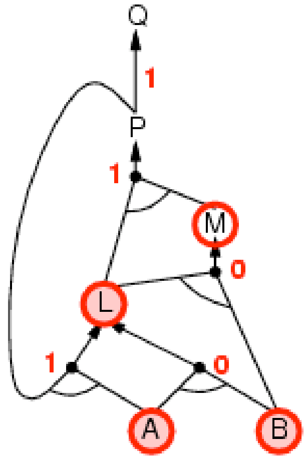
Forward chaining example



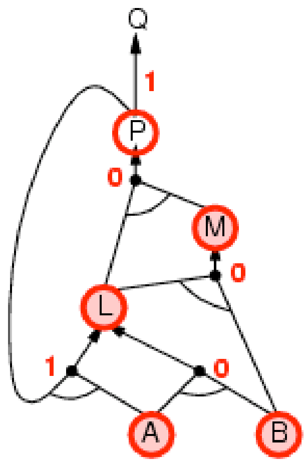
Forward chaining example



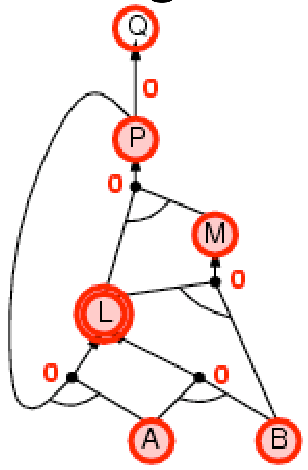
Forward chaining example



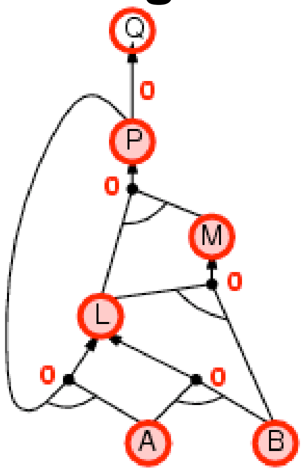
Forward chaining example



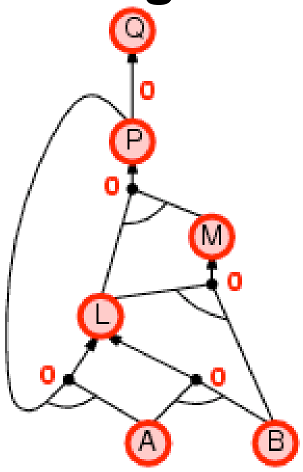
Forward chaining example



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* |= *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

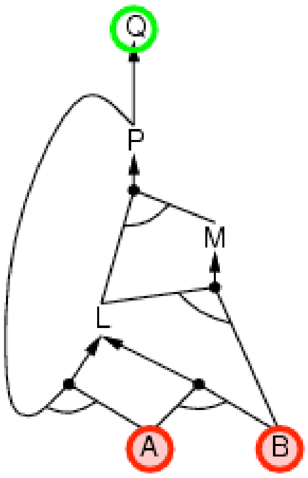
prove by BC all premises of some rule concluding *q*

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

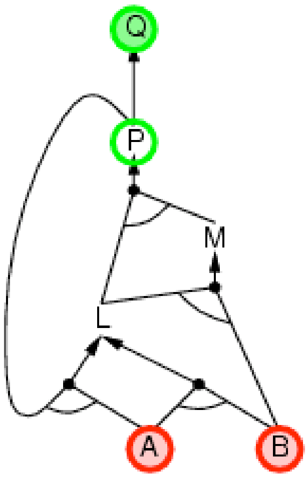
Avoid repeated work: check if new subgoal

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

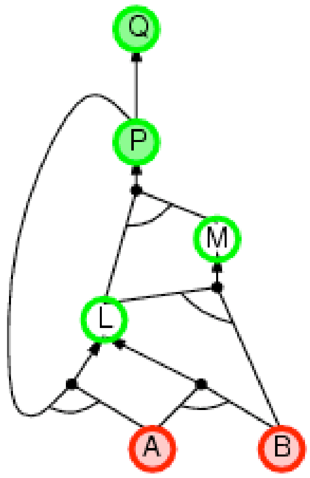
Backward chaining example



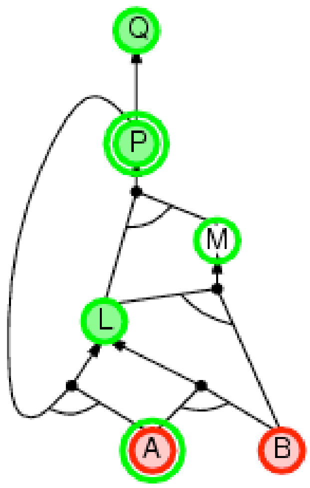
Backward chaining example



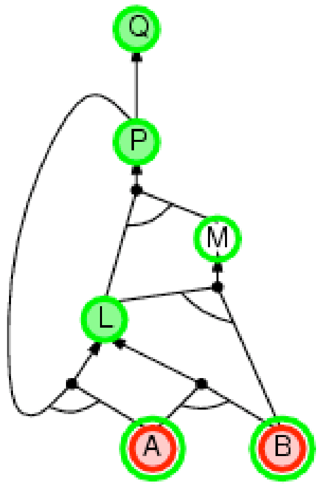
Backward chaining example



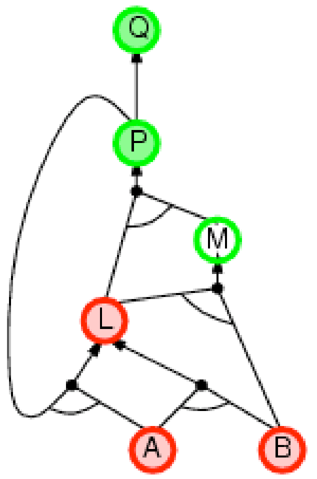
Backward chaining example



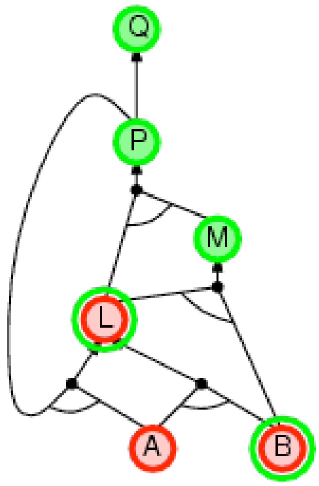
Backward chaining example



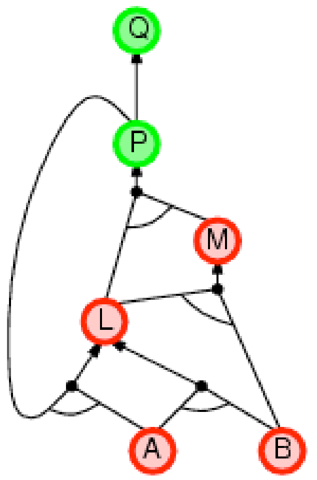
Backward chaining example



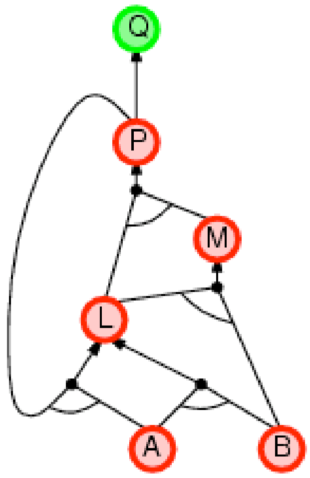
Backward chaining example



Backward chaining example

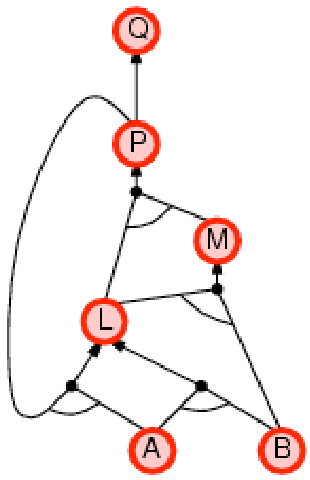


Backward chaining example



November 16, 2004 TLo (IRIDIA) 43

Backward chaining example



November 16, 2004 TLo (IRIDIA) 44

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

Effective propositional inference

- Two families of efficient algorithms for propositional inference based on model checking:
- Are used for checking satisfiability
- 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 ∨ ¬B), (¬B ∨ ¬C), (C ∨ 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.
- Steps are taken in the space of complete assignments, flipping the truth value of one variable at a time.
- Balance between greediness and randomness.
  - **To avoid local minima**

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

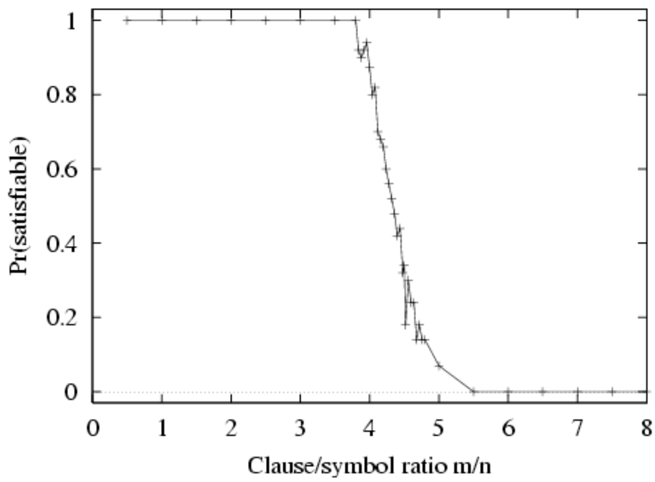
- Consider random 3-CNF sentences. e.g.,
  (¬D ∨ ¬B ∨ C) ∧ (B ∨ ¬A ∨ ¬C) ∧ (¬C ∨ ¬B ∨ E) ∧ (E ∨ ¬D ∨ B) ∧ (B ∨ E ∨ ¬C)

  m **= number of clauses**

  n **= number of symbols**

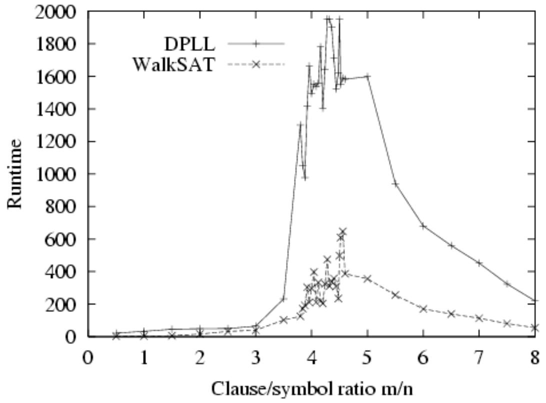
  - **Hard problems seem to cluster near *m/n* = 4.3 (critical point)**

Hard satisfiability problems



Hard satisfiability problems

- Median runtime for 100 satisfiable random 3-CNF sentences, n = 5 0



Inference-based agents in the wumpus world

A wumpus-world agent using propositional logic:

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

⇒ 64 distinct proposition symbols, 155 sentences
⇒ A fringe square is provably safe if the sentence $(\neg P_{i,j} \wedge \neg W_{i,j})$ is entailed by the knowledge base.

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($KB, \neg S_{x,y}$)
if *breeze* **then** TELL($KB, B_{x,y}$) **else** TELL($KB, \neg B_{x,y}$)
if *glitter* **then** *action* ← *grab*
else if *plan* is nonempty then *action* ← POP(*plan*)
else if for some fringe square [*i*,*j*], ASK($KB, (\neg P_{i,j} \land \neg W_{i,j})$) is *true* **or**
for some fringe square [*i*,*j*], ASK($KB, (P_{i,j} \lor W_{i,j})$) 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
  - **With all consequences for large KB**

t **Better would be to have just two sentences foor breezes and stenches for all squares.** t

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

$$L_{x,y} \land FacingRight^{t} \land Forward^{t} \Rightarrow L_{x+1,y}$$

- 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