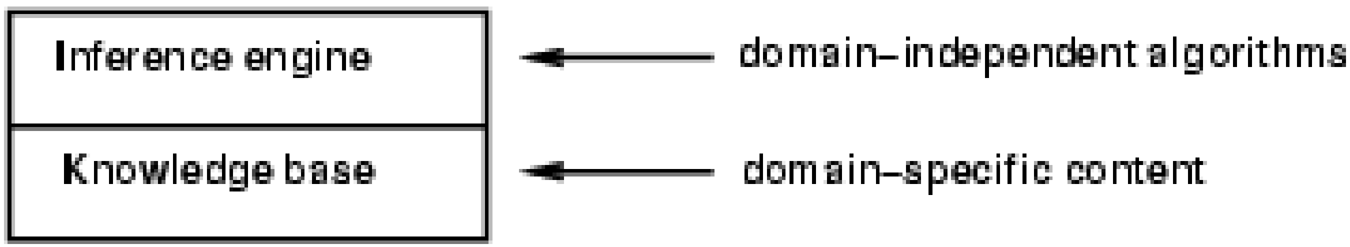
Set 6: Knowledge Representation: The Propositional Calculus

ICS 271 Fall 2014
Kalev Kask

Outline

- **Representing knowledge using logic**
  - Agent that reason logically
  - A knowledge based agent
- **Representing and reasoning with logic**
  - Propositional logic
    - Syntax
    - Semantic
    - Validity and models
    - Rules of inference for propositional logic
    - Resolution
    - Complexity of propositional inference.
- **Reading: Russel and Norvig, Chapter 7**

Knowledge bases



- Knowledge base = set of sentences in a formal language

- Declarative approach to building an agent (or other system):
  - `Tell` it what it needs to know

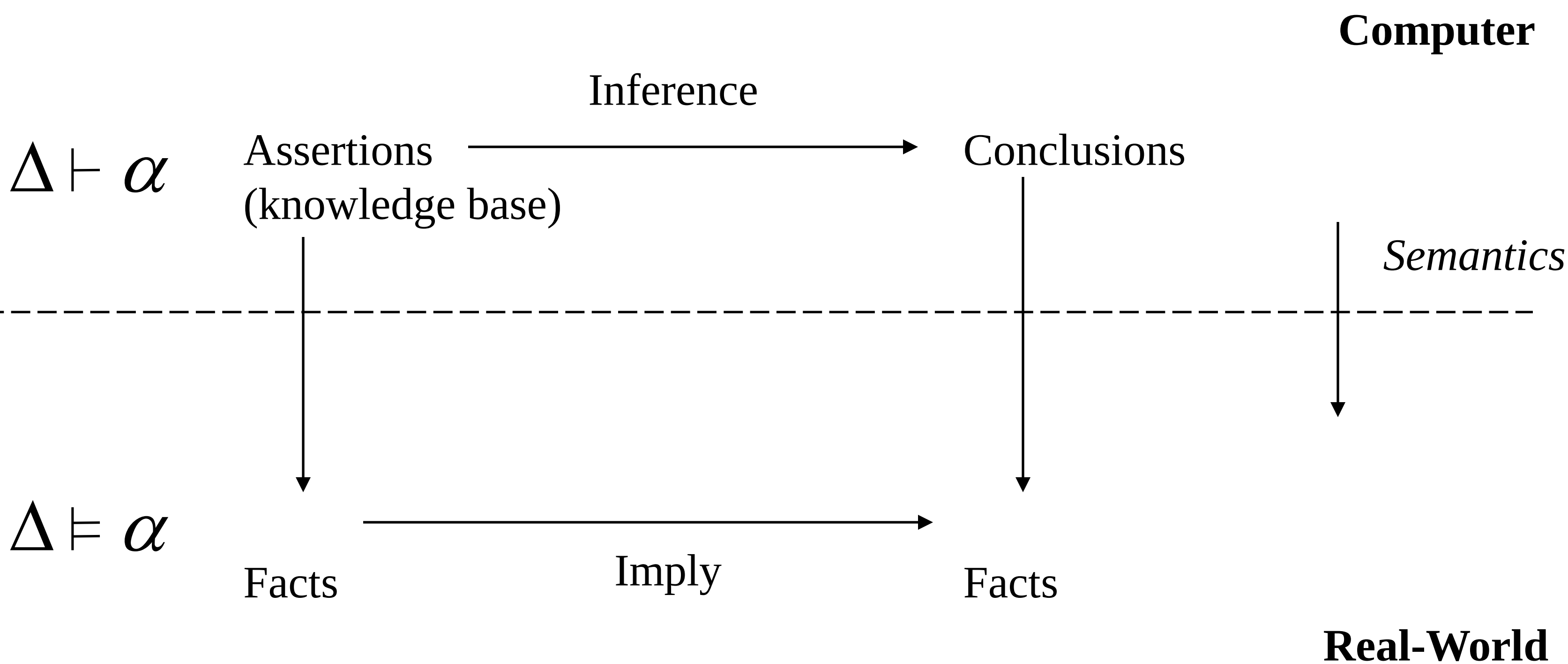
- Then it can `Ask` itself what to do - answers should follow from the KB

- Agents can be viewed at the knowledge level
  i.e., what they know, regardless of how implemented

- Or at the implementation level
  - i.e., data structures in KB and algorithms that manipulate them

Knowledge Representation

Defined by: syntax, semantics



Reasoning: in the syntactic level

Example: $x > y, y > z \;\vDash\; x > z$

The party example

- If Alex goes, then Beki goes: $A \rightarrow B$
- If Chris goes, then Alex goes: $C \rightarrow A$
- Beki does not go: not B
- Chris goes: C
- Query: Is it possible to satisfy all these conditions?

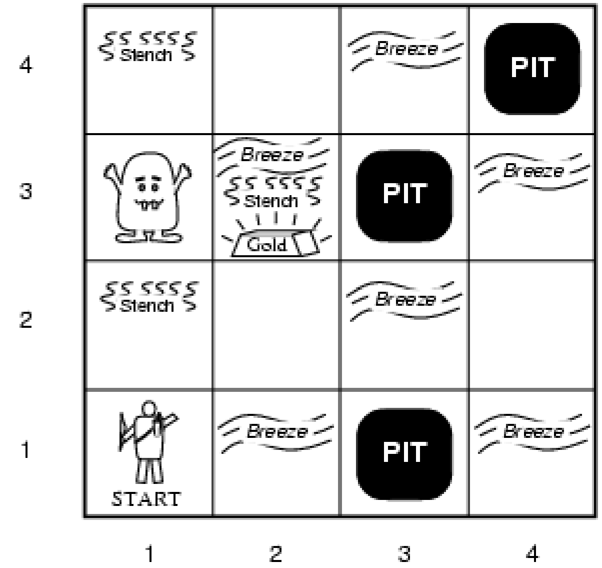
- Should I go to the party?

Example of languages

- **Programming languages:**
  - Formal languages, not ambiguous, but cannot express partial information. Not expressive enough.
- **Natural languages:**
  - Very expressive but ambiguous: ex: small dogs and cats.
- **Good representation language:**
  - Both formal and can express partial information, can accommodate inference
- **Main approach used in AI: Logic-based languages.**

Wumpus World test-bed

- Performance measure
  - gold +1000, death -1000
  - -1 per step, -10 for using the arrow
- Environment
-
  - Squares adjacent to wumpus are smelly
  -
  - Squares adjacent to pit are breezy
  -
  - Glitter iff gold is in the same square
  -
  - Shooting kills wumpus if you are facing it
  -
  - Shooting uses up the only arrow
  -
  - Grabbing picks up gold if in same square
  -
  - Releasing drops the gold in same square
  -

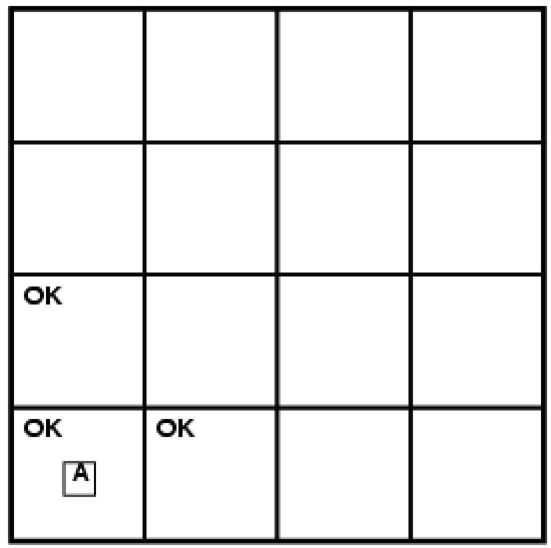


- Sensors: Stench, Breeze, Glitter, Bump, Scream
-
- Actuators: Left turn, Right turn, Forward, Grab, Release, Shoot
-

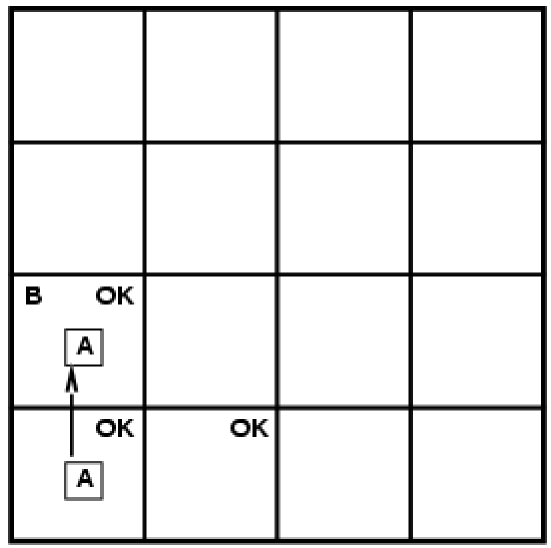
Wumpus world characterization

- Fully Observable **N**o – only local perception
- Deterministic Yes – outcomes exactly specified
- Episodic No – sequential at the level of actions
- Static Yes – Wumpus and Pits do not move
- Discrete Yes
- Single-agent? Yes – Wumpus is essentially a natural feature

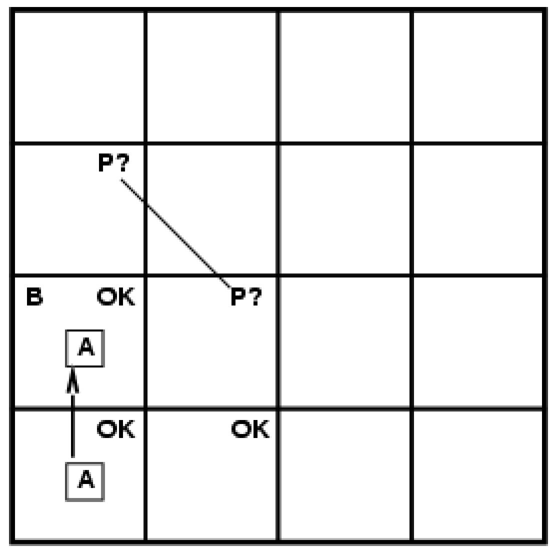
Exploring a wumpus world



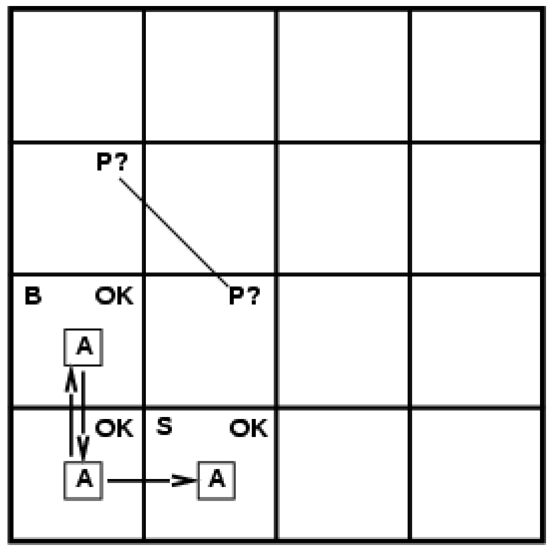
Exploring a wumpus world



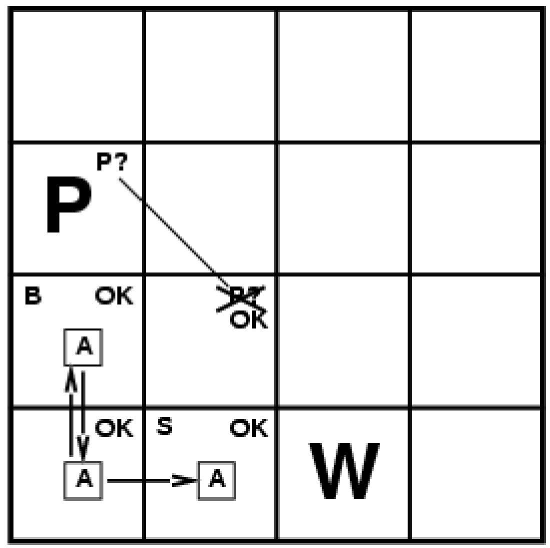
Exploring a wumpus world



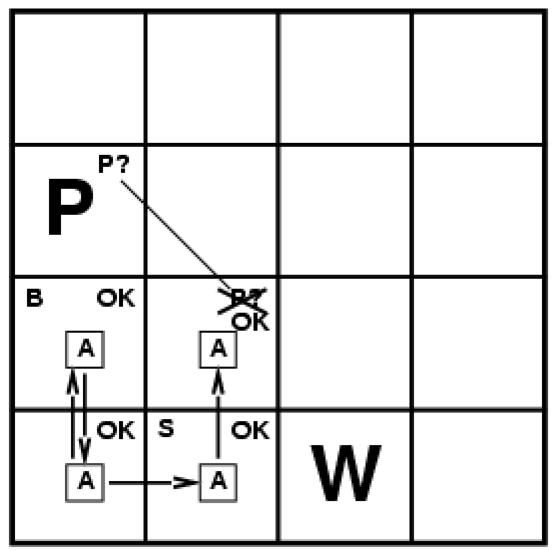
Exploring a wumpus world



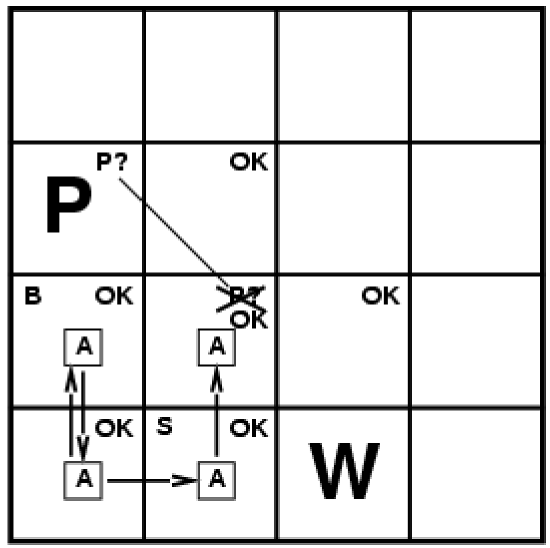
Exploring a wumpus world



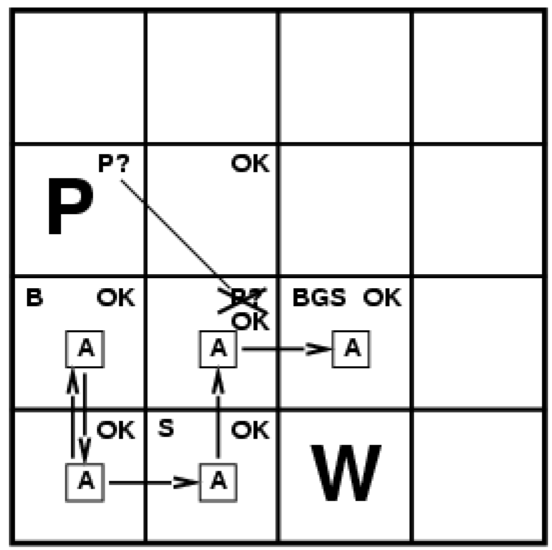
Exploring a wumpus world



Exploring a wumpus world



Exploring a wumpus world



Logic in general

- Logics are formal languages for representing information such that conclusions can be drawn
- Syntax defines the sentences in the language
- Semantics define the "meaning" of sentences;
  - i.e., define truth of a sentence in a world
- E.g., the language of arithmetic
  - $x+2 \geq y$ is a sentence; $x2+y > \{\}$ is not a sentence
  - $x+2 \geq y$ is true iff the number $x+2$ is no less than the number y
  - $x+2 \geq y$ is true in a world where $x = 7$, $y = 1$
  - $x+2 \geq y$ is false in a world where $x = 0$, $y = 6$

Summary so far

- Knowledge representation vs problem solving
- General purpose representation + inference engine
- Declarative approach
  - Encode rules, facts, observations
  - Ask questions (queries)
- Formal languages : syntax, semantics
- Entailment : facts imply facts
- Inference : mechanical manipulation
  - Sound
  - Complete

Entailment

- Entailment means that one thing follows from another:

$$KB \models \alpha$$

- Knowledge base *KB* entails sentence α if and only if α is true in all worlds where *KB* is true
  - E.g., the KB containing "the Giants won" and "the Reds won" entails "Either the Giants won or the Reds won"
  - E.g., x+y = 4 entails 4 = x+y
  - Entailment is a relationship between sentences (i.e. syntax) that is based on semantics

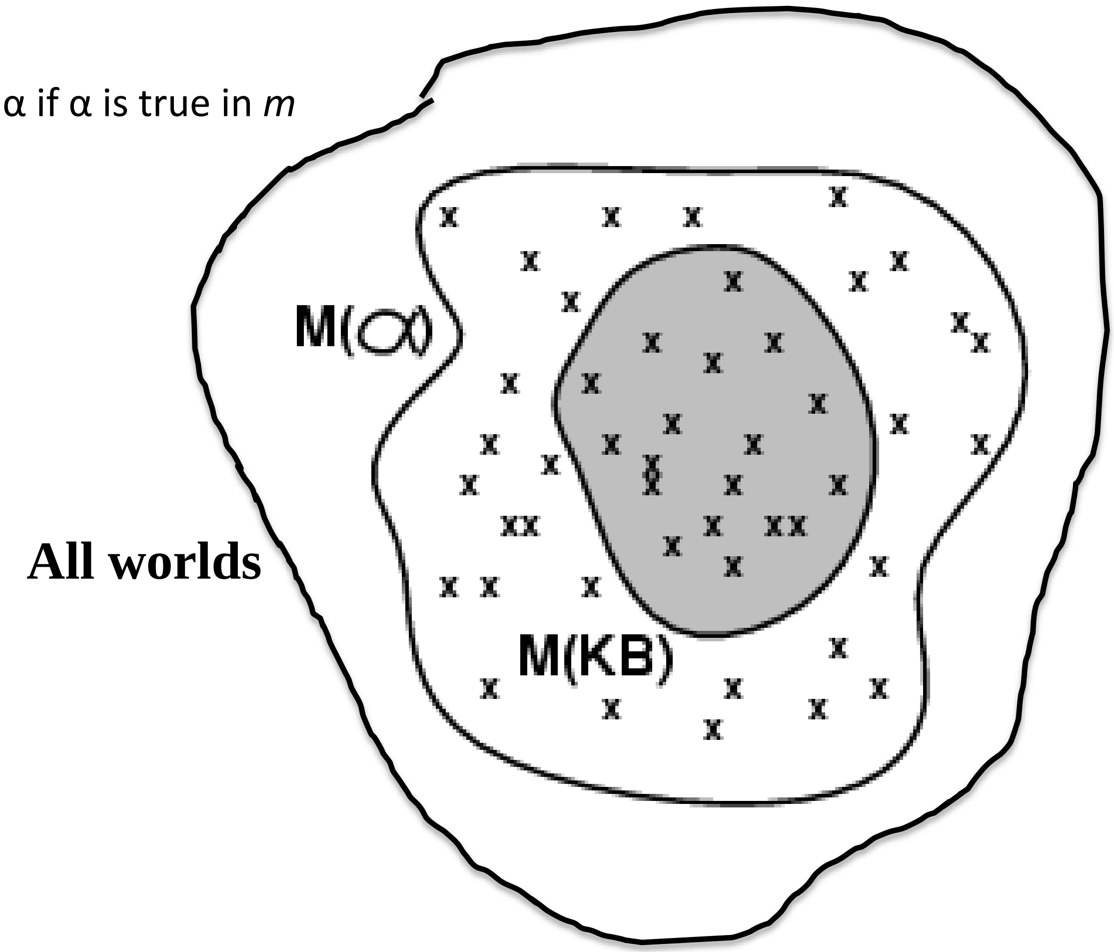
Models

- Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated

- We say m is a model of a sentence α if α is true in m

- $M(\alpha)$ is the set of all models of α

- Then KB $\models$ α iff $M(KB) \subseteq M(\alpha)$
  - E.g. KB = Giants won and Reds won α = Giants won

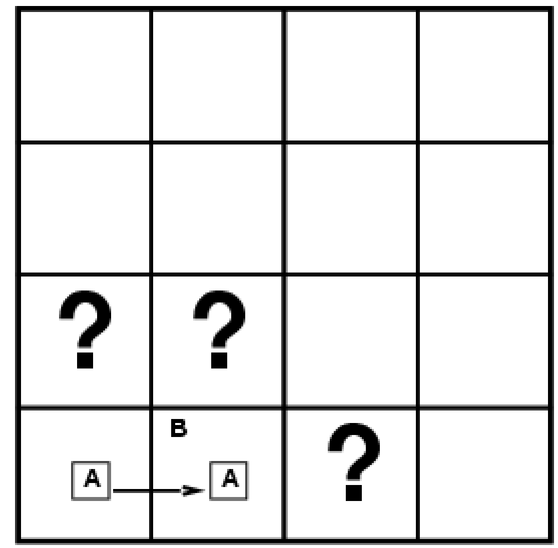


Entailment in the wumpus world

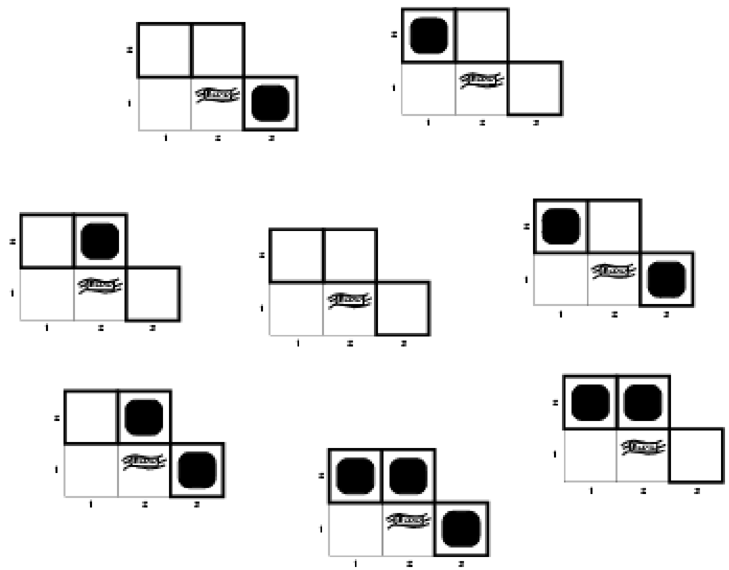
Situation after detecting nothing in [1,1], moving right, breeze in [2,1]

Consider possible models for *KB* assuming only pits

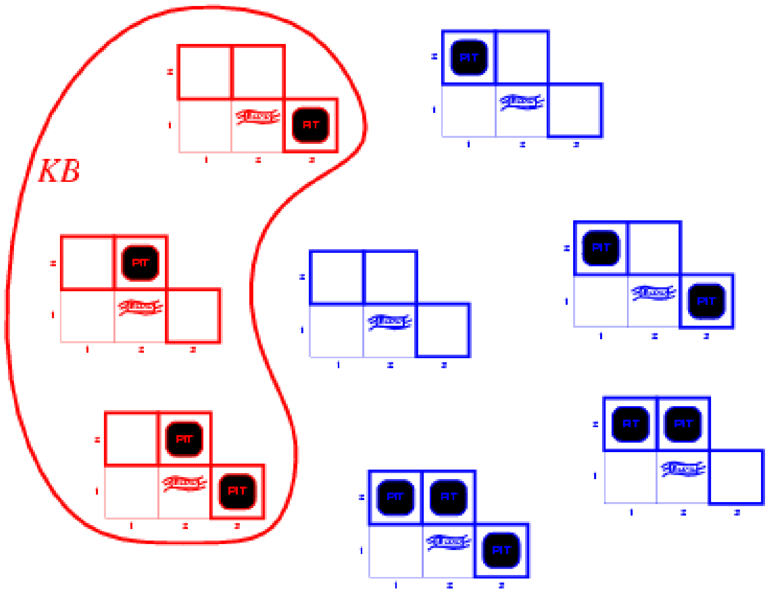
3 Boolean choices $\Rightarrow$ 8 possible models



Wumpus models

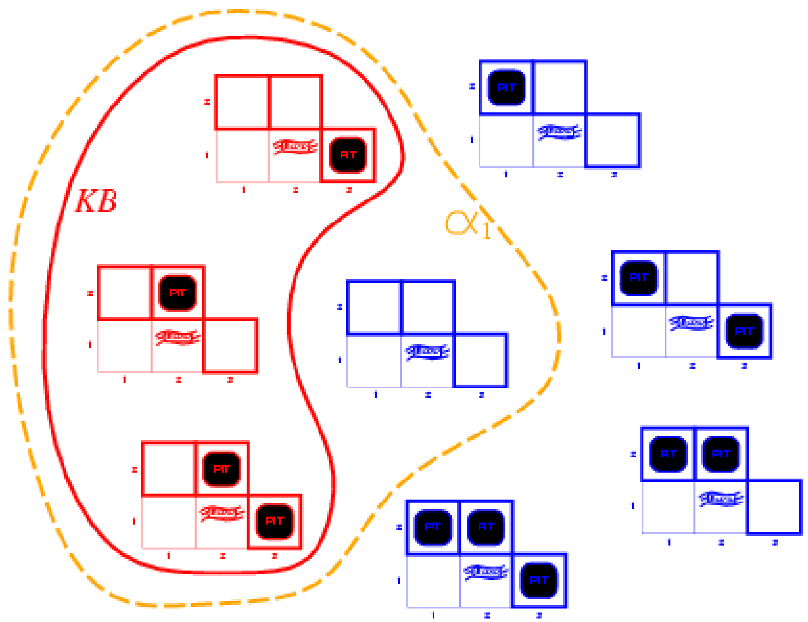


Wumpus models



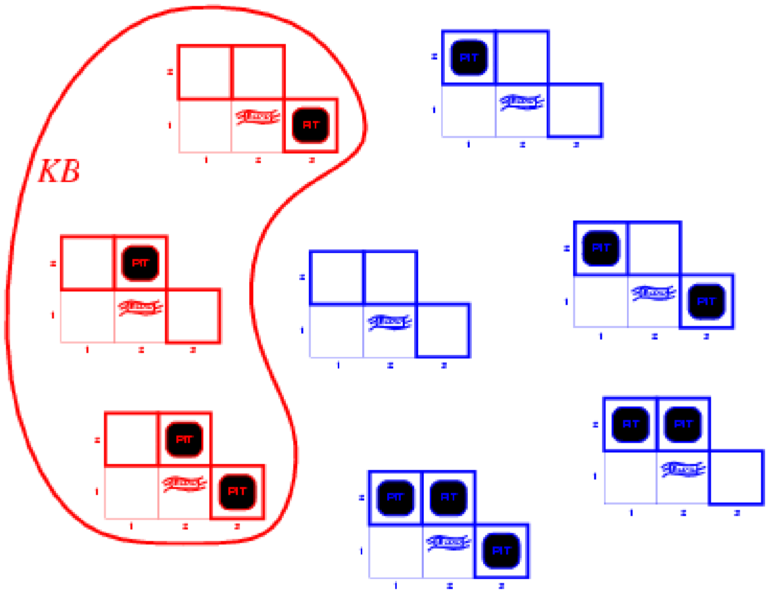
- *KB* = wumpus-world rules + observations

Wumpus models



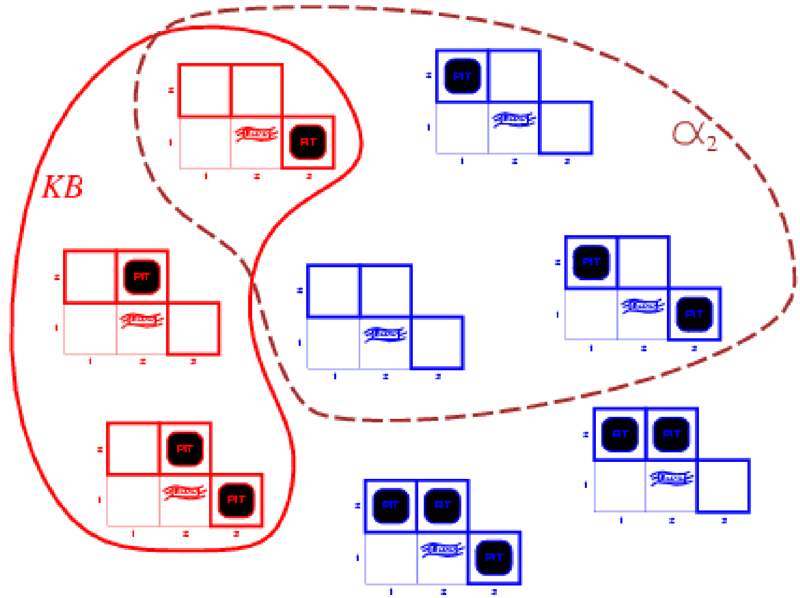
- *KB* = wumpus-world rules + observations
- α_1 = "[1,2] is safe", *KB* $\models \alpha_1$, proved by model checking

Wumpus models



- *KB* = wumpus-world rules + observations

Wumpus models



- *KB* = wumpus-world rules + observations
- α_2 = "[2,2] is safe", $KB \not\models \alpha_2$

Propositional logic: Syntax

- 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: Semantics

Each world specifies true/false for each proposition symbol

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

With these symbols 8 possible worlds can be enumerated automatically.

Rules for evaluating truth with respect to a world *w*:

$\neg S$	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.,

$$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = \mathit{true} \wedge (\mathit{true} \vee \mathit{false}) = \mathit{true} \wedge \mathit{true} = \mathit{true}$$

Truth tables for connectives

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

Logical equivalence

Two sentences are logically equivalent iff true in same models: α ≡ ß iff α ╞ β and β ╞α

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

Wumpus world sentences

- Rules

  - "Pits cause breezes in adjacent squares“

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

- Observations

  - 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].

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

Wumpus world sentences

KB

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“

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

Truth table for KB

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	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*

<u>α_1= no pit in (1,2)</u>

<u>α_2= no pit in (2,2)</u>

Truth Tables

- Truth tables can be used to compute the truth value of any wff (well formed formula)
  - Can be used to find the truth of $((P \rightarrow R) \rightarrow Q) \vee \neg S$
- Given n features there are 2^n different worlds (interpretations).
- Interpretation: any assignment of true and false to atoms
- An interpretation satisfies a wff (sentence) if the sentence is assigned true under the interpretation
- A model: An interpretation is a model of a sentence if the sentence is satisfied in that interpretation.
- Satisfiability of a sentence can be determined by the truth-table
  - Bat_on and turns-key_on → Engine-starts
- A sentence is unsatisfiable or inconsistent it has no models
  - $P \wedge (\neg P)$
  - $(P \vee Q) \wedge (P \vee \neg Q) \wedge (\neg P \vee Q) \wedge (\neg P \vee \neg Q)$

Inference

$KB \vdash_i \alpha$ = sentence α can be derived from KB by procedure i

Consequences of KB are a haystack; α is a needle.
Entailment = needle in haystack; inference = finding it

Soundness: i is sound if
whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$

Completeness: i is complete if
whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$

Preview: we will define a logic (first-order logic) which is expressive enough to say almost anything of interest, and for which there exists a sound and complete inference procedure.

That is, the procedure will answer any question whose answer follows from what is known by the KB.

Decidability – there exists a procedure that will correctly answer Y/N (valid or not) for any formula

Gödel's incompleteness theorem (1931) – any deductive system that includes number theory is either incomplete or unsound.

Validity and satisfiability

A sentence is valid if it is true in all worlds,

e.g., *True*, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$

A sentence is satisfiable if it is true in some world (has a model)

e.g., $A \vee B$, C

A sentence is unsatisfiable if it is true in no world (has no model)

e.g., $A \wedge \neg A$

Validity is connected to inference via the Deduction Theorem:

$KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

(note : $(KB \Rightarrow \alpha)$ is the same as $(\neg KB \vee \alpha)$)

Satisfiability is connected to inference via the following:

$KB \models \alpha$ if and only if $(KB \wedge \neg \alpha)$ is unsatisfiable

Validity

P	H	$P \vee H$	$(P \vee H) \wedge \neg H$	$((P \vee H) \wedge \neg H) \Rightarrow P$
False	*False*	*False*	*False*	*True*
False	*True*	*True*	*False*	*True*
True	*False*	*True*	*True*	*True*
True	*True*	*True*	*False*	*True*

Figure 6.10 Truth table showing validity of a complex sentence.

Inference methods

- Proof methods divide into (roughly) two kinds:
  - Model checking
    - truth table enumeration (always exponential in *n*)
    - improved backtracking, e.g., Davis--Putnam-Logemann-Loveland (DPLL), Backtracking with constraint propagation, backjumping.
    - heuristic search in model space (sound but incomplete)
      e.g., min-conflicts-like hill-climbing algorithms
  - Deductive systems
    - Legitimate (sound) generation of new sentences from old
    - Proof = a sequence of inference rule applications
      Can use inference rules as operators in a standard search algorithm
    - Typically require transformation of sentences into a normal form

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 $O(n)$

Deductive systems : rules of inference

◇ **Modus Ponens** or **Implication-Elimination**: (From an implication and the premise of the implication, you can infer the conclusion.)

$$\frac{\alpha \Rightarrow \beta, \qquad \alpha}{\beta}$$

◇ **And-Elimination**: (From a conjunction, you can infer any of the conjuncts.)

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_i}$$

◇ **And-Introduction**: (From a list of sentences, you can infer their conjunction.)

$$\frac{\alpha_1, \alpha_2, \quad \ldots, \quad \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}$$

◇ **Or-Introduction**: (From a sentence, you can infer its disjunction with anything else at all.)

$$\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n}$$

◇ **Double-Negation Elimination**: (From a doubly negated sentence, you can infer a positive sentence.)

$$\frac{\neg\neg\alpha}{\alpha}$$

◇ **Unit Resolution**: (From a disjunction, if one of the disjuncts is false, then you can infer the other one is true.)

$$\frac{\alpha \vee \beta, \qquad \neg\beta}{\alpha}$$

◇ **Resolution**: (This is the most difficult. Because β cannot be both true and false, one of the other disjuncts must be true in one of the premises. Or equivalently, implication is transitive.)

$$\frac{\alpha \vee \beta, \qquad \neg\beta \vee \gamma}{\alpha \vee \gamma} \quad \text{or equivalently} \quad \frac{\neg\alpha \Rightarrow \beta, \qquad \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}$$

Figure 6.13 Seven inference rules for propositional logic. The unit resolution rule is a special case of the resolution rule, which in turn is a special case of the full resolution rule for first-order logic discussed in Chapter 9.

Resolution in Propositional Calculus

- **Using clauses as wffs**
  - Literal, clauses, conjunction of clauses (CNFs) $(P \vee Q \vee \neg R)$
- **Resolution rule:**
  - Resolving (P V Q) and (P V $\neg$ Q) $\vdash$ P
  - Generalize modus ponens, chaining .
  - Resolving a literal with its negation yields empty clause.
- **Resolution rule is sound**
- **Resolution rule is NOT complete:**
  - P and R entails P V R but you cannot infer P V R From (P and R) by resolution
- **Resolution is complete for refutation:** adding ($\neg$P) and ($\neg$R) to (P and R) we can infer the empty clause.
- **Decidability of propositional calculus by resolution refutation:** if a sentence w is not entailed by KB then resolution refutation will terminate without generating the empty clause.

Resolution

Conjunctive Normal Form (CNF—universal)

$\underbrace{\textit{conjunction} \text{ of } \textit{disjunctions} \text{ of } \textit{literals}}$ wait

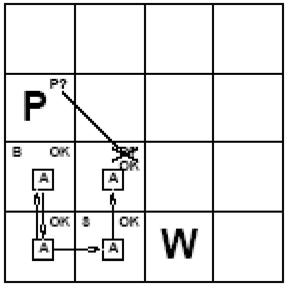
E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Resolution inference rule (for CNF): complete for propositional logic

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

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 and complete for propositional logic

Conversion to CNF

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

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
2.

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

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:

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

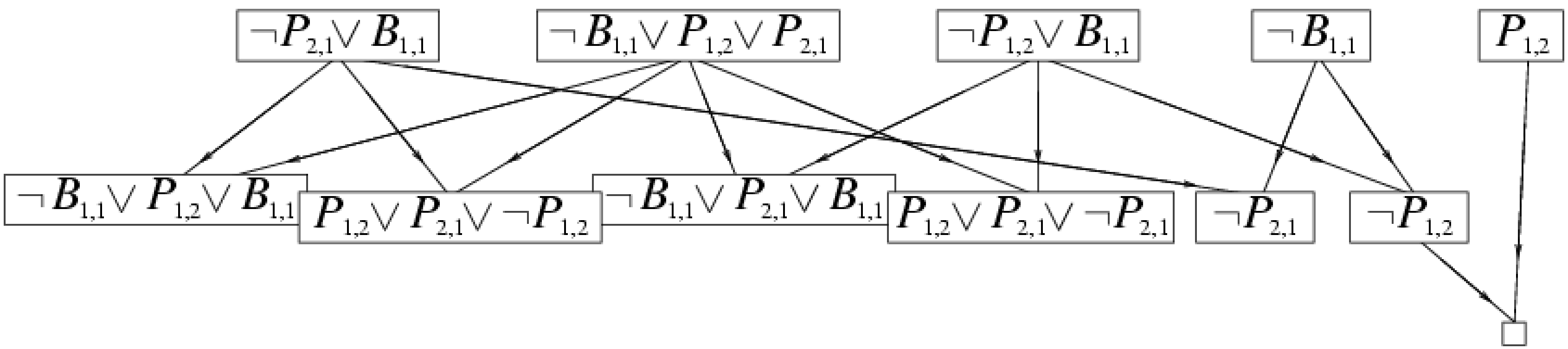
Resolution algorithm

- 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 example

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



Soundness of resolution

α	β	γ	$\alpha \vee \beta$	$\neg\beta \vee \gamma$	$\alpha \vee \gamma$
False	*False*	*False*	*False*	*True*	*False*
False	*False*	*True*	*False*	*True*	*True*
False	*True*	*False*	*True*	*False*	*False*
<u>*False*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>
<u>*True*</u>	<u>*False*</u>	<u>*False*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>
<u>*True*</u>	<u>*False*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>
True	*True*	*False*	*True*	*False*	*True*
<u>*True*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>

Figure 6.14 A truth table demonstrating the soundness of the resolution inference rule. We have underlined the rows where both premises are true.

The party example

- If Alex goes, then Beki goes: $A \rightarrow B$
- If Chris goes, then Alex goes: $C \rightarrow A$
- Beki does not go: not B
- Chris goes: C
- Query: Is it possible to satisfy all these conditions?

- Should I go to the party?

Example of proof by Refutation

- **Assume the claim is false and prove inconsistency:**
  - Example: can we prove that Chris will not come to the party?

$$A \to B, \neg B$$
$$C \to A$$

- **Prove by generating the desired goal.**
- **Prove by refutation: add the negation of the goal and prove no model**
- **Proof:**

$$from\ A \to B, \neg B\ infer\ \neg A$$
$$from\ C \to A, \neg A\ infer\ \neg C$$

- **Refutation:**

$$\begin{tikzcd}[column sep=small]
A \to B \arrow[dr] & & \neg B \arrow[dl] & & C \to A \arrow[ddl] & & \neg(\neg C) \arrow[dddl] \\
& \neg A \arrow[drr] & & & & & \\
& & & \neg C \arrow[drr] & & & \\
& & & & & \varphi &
\end{tikzcd}$$

Summary so far

- Propositional logic : syntax, semantics
  - Truth tables
- Inference
  - KB $\models \alpha$ iff KB $\Rightarrow \alpha$ is valid
  - Valididity, (un)satisfiability
  - Soundness, completeness
  - Basic methods
    - Model checking - DPLL
    - Application of inference rules – resolution
- Proof by refutation
  - $KB \models \alpha$ if and only if $(KB \wedge \neg\alpha)$ is unsatisfiable
  - Derive empty (CNF) clause : resolution
  - Prove that $(KB \wedge \neg\alpha)$ has no model : search A*, Backtracking, etc.

Proof by refutation

- **Given a database in clausal normal form KB**
  - Find a sequence of resolution steps from KB to the empty clauses
    - Use the search space paradigm:
      - <u>States:</u> current cnf KB + new clauses
      - <u>Operators:</u> resolution
      - <u>Initial state:</u> KB + negated goal
      - <u>Goal State:</u> a database containing the empty clause
      - Search using any search method

Proof by refutation (cont.)

- **Or:**
  - Prove that KB has no model - PSAT
    - A CNF theory is a constraint satisfaction problem:
      - variables: the propositions
      - domains: true, false
      - constraints: clauses (or their truth tables)
      - Find a solution to the csp. If no solution no model.
      - This is the satisfiability question
      - Methods: Backtracking arc-consistency ≈ unit resolution, local search

Resolution refutation search strategies

- **Ordering strategies**
  - Breadth-first, depth-first
  - I-level resolvents are generated from level-(I-1) or less resolvents
  - Unit-preference: prefer resolutions with a literal
- **Set of support**:
  - Allows resolutions in which one of the resolvents is in the set of support
  - The set of support: those clauses coming from negation of the theorem or their decendents.
  - The set of support strategy is refutation complete
- **Linear input**
  - Restricted to resolutions when one member is in the input clauses
  - Linear input is not refutation complete
  - Example: (PVQ) (P V not Q) (not P V Q) (not P V not Q) have no model

Complexity of propositional inference

- **Checking truth tables is exponential**
- **Satisfiability is NP-complete**
- **However, frequently generating proofs is easy.**
- **Propositional logic is monotonic**
  - If you can entail alpha from knowledge base KB and if you add sentences to KB, you can infer alpha from the extended knowledge-base as well.
- **Inference is local**
  - Tractable Classes: Horn, 2-SAT
- **Horn theories:**
  - Q <-- P_1,P_2,...$P_{n,}$
  - Pi is an atom in the language, Q can be false.

– : only implications. Clauses have a positive literal

- **Solved by modus ponens or “unit resolution”.**

Forward and backward chaining

Horn Form (restricted)

KB = *conjunction* of *Horn clauses*

Horn clause =

- ♢ proposition symbol; or
- ♢ (conjunction of symbols) $\Rightarrow$ symbol

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

Modus Ponens (for Horn Form): complete for Horn KBs

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

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

Forward chaining algorithm

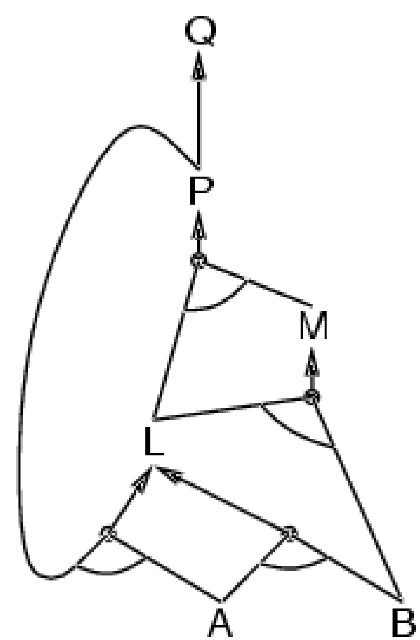
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known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

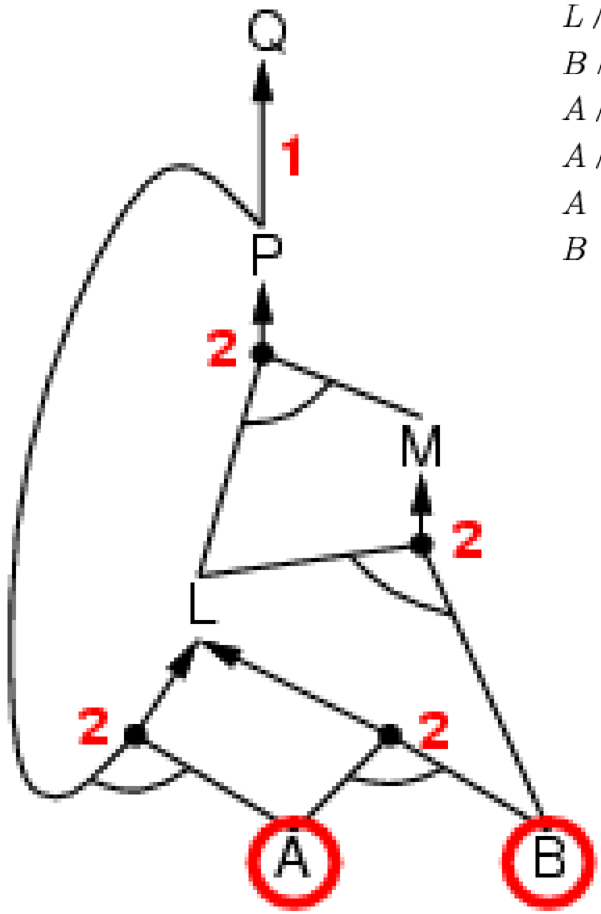
Forward chaining

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

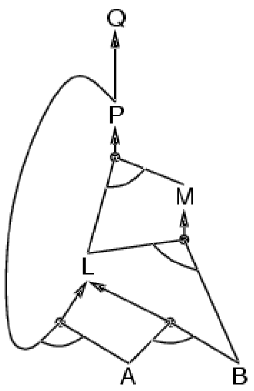
$$
\begin{aligned}
&P \Rightarrow Q\\
&L \wedge M \Rightarrow P\\
&B \wedge L \Rightarrow M\\
&A \wedge P \Rightarrow L\\
&A \wedge B \Rightarrow L\\
&A\\
&B
\end{aligned}
$$



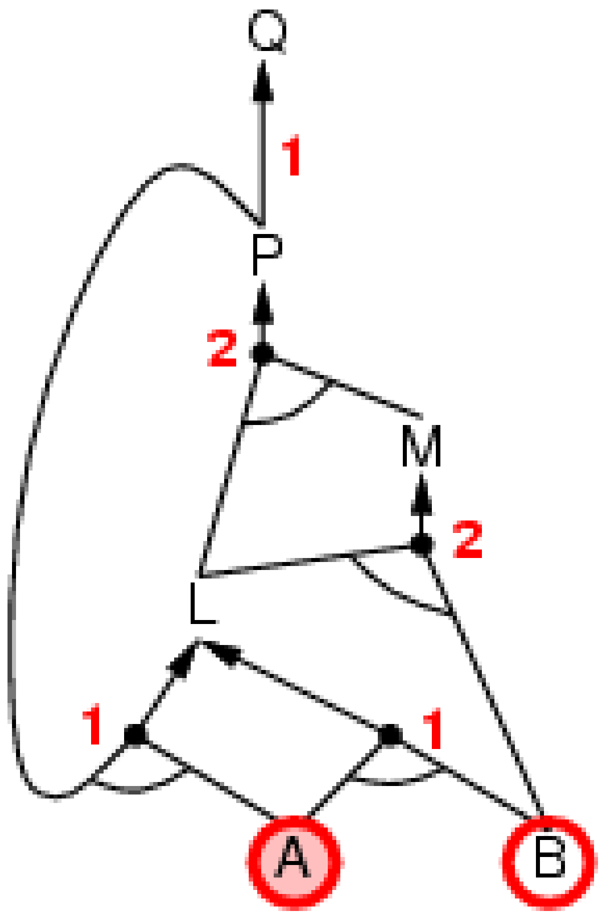
Forward chaining example



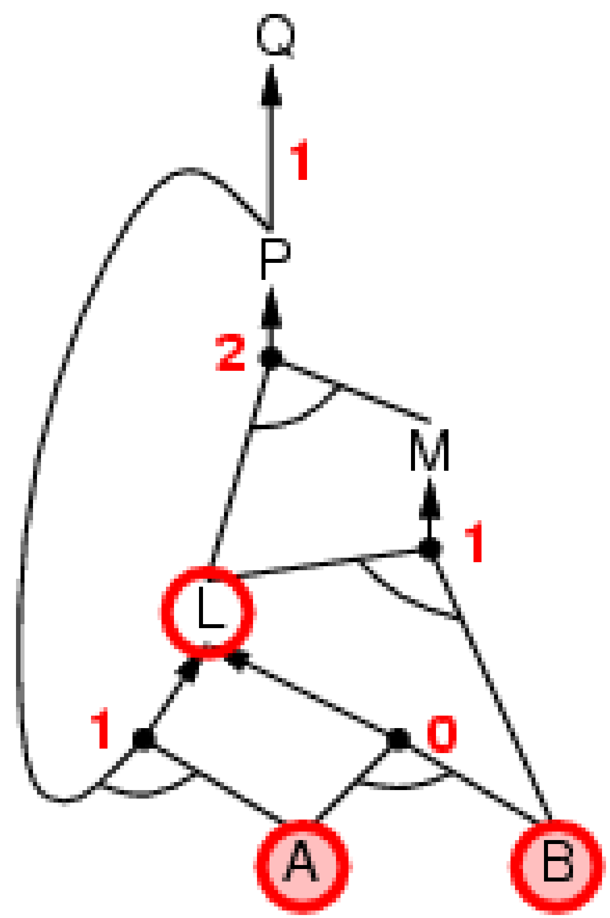
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



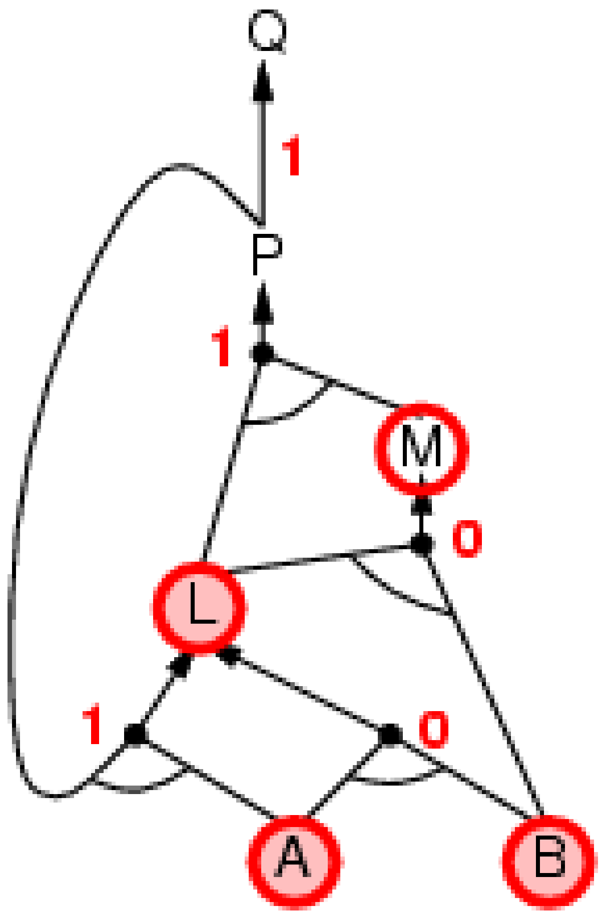
Forward chaining example



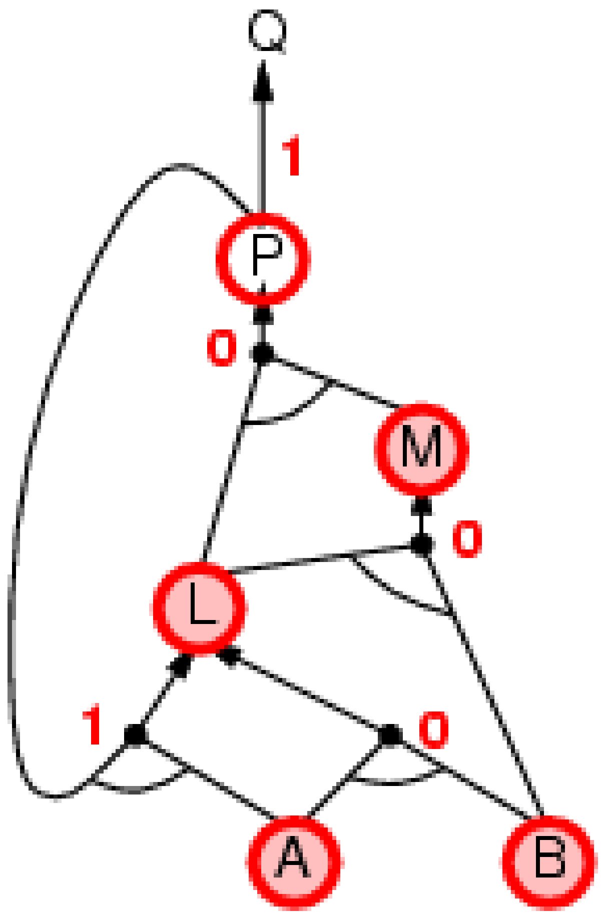
Forward chaining example



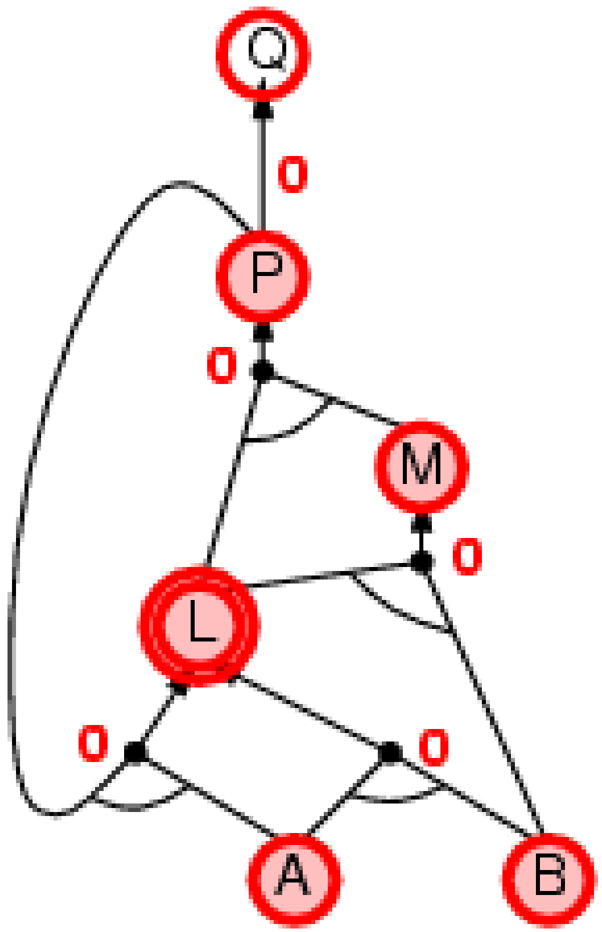
Forward chaining example



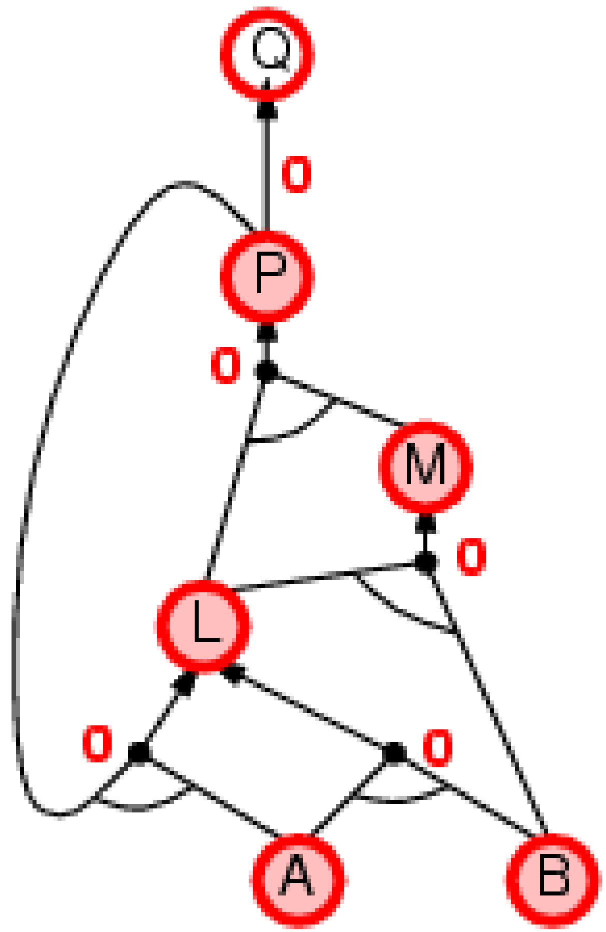
Forward chaining example



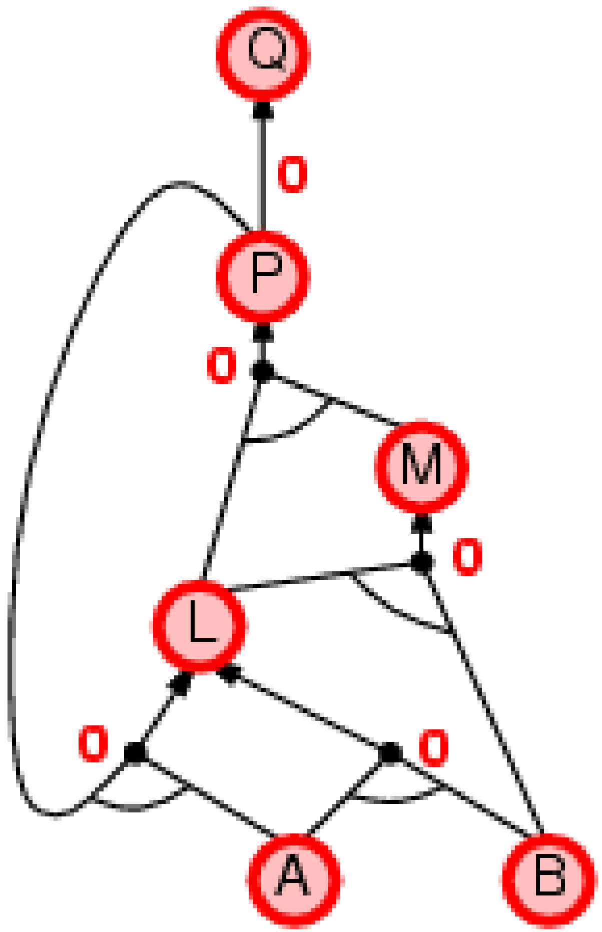
Forward chaining example



Forward chaining example



Forward chaining example



Backward chaining (BC)

Idea: work backwards from the query *q*:

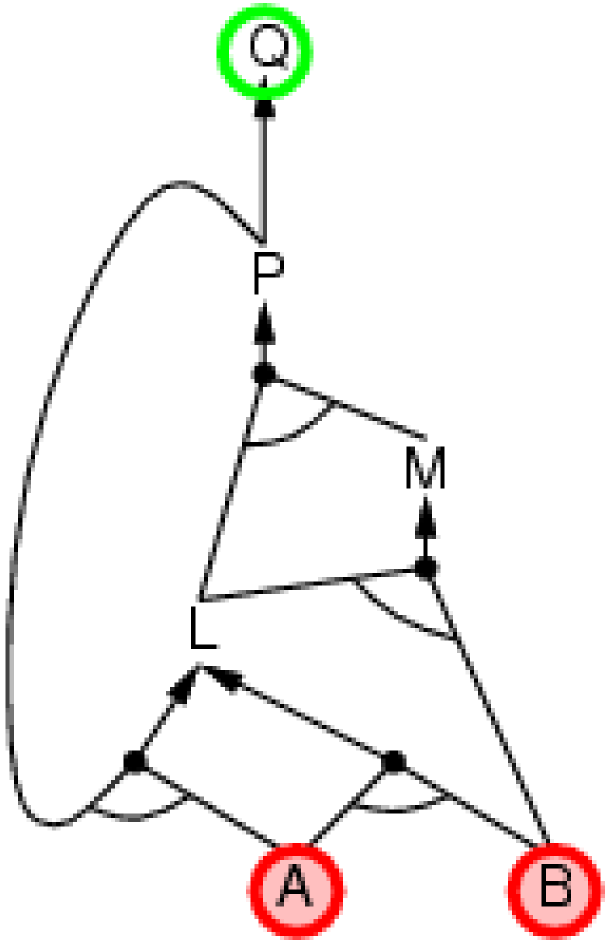
to prove *q* by BC,
check if *q* is known already, or
prove by BC all premises of some rule concluding *q*

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

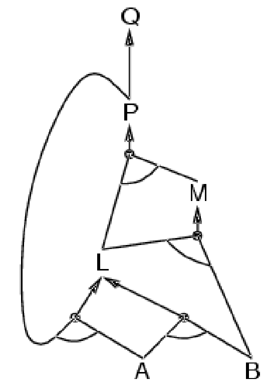
Avoid repeated work: check if new subgoal

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

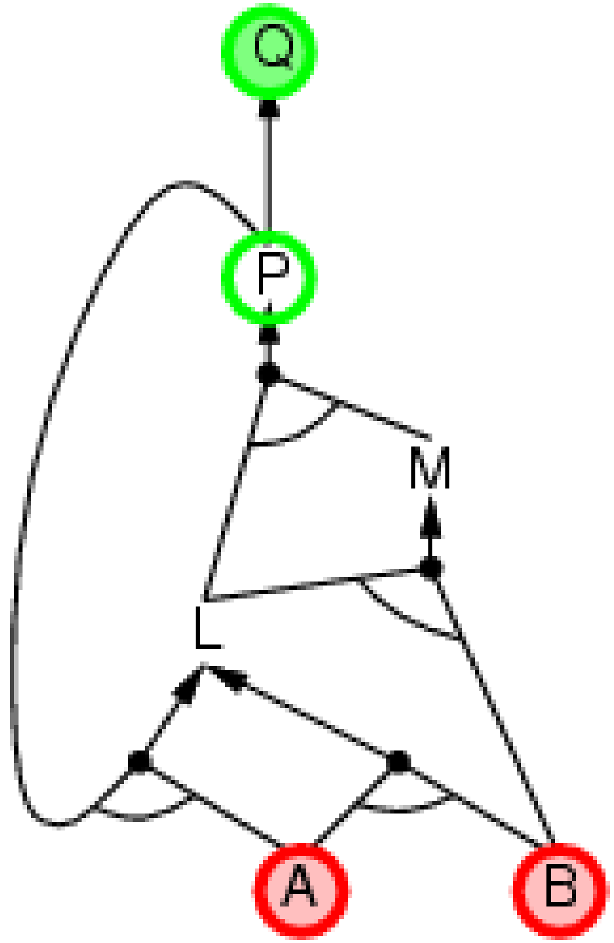
Backward chaining example



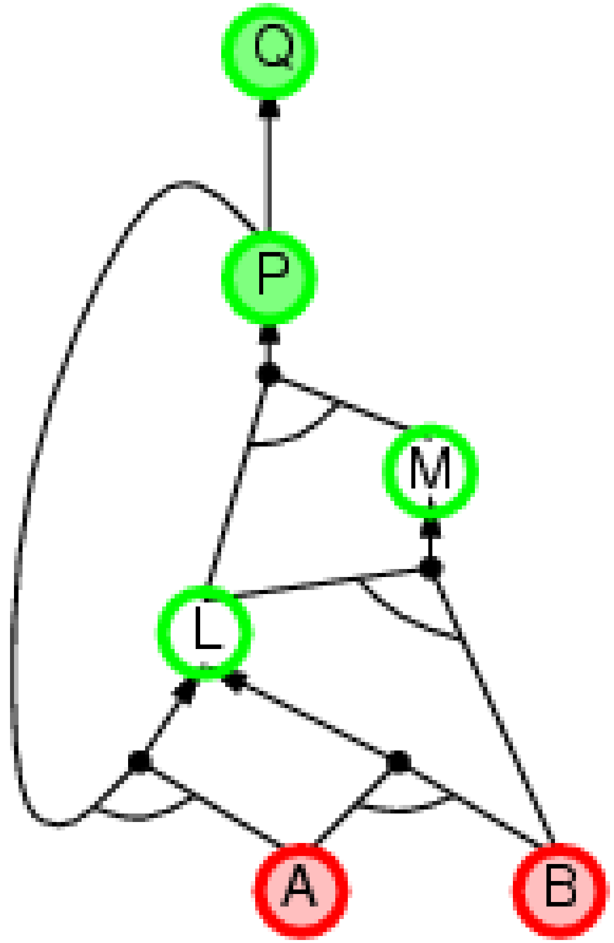
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



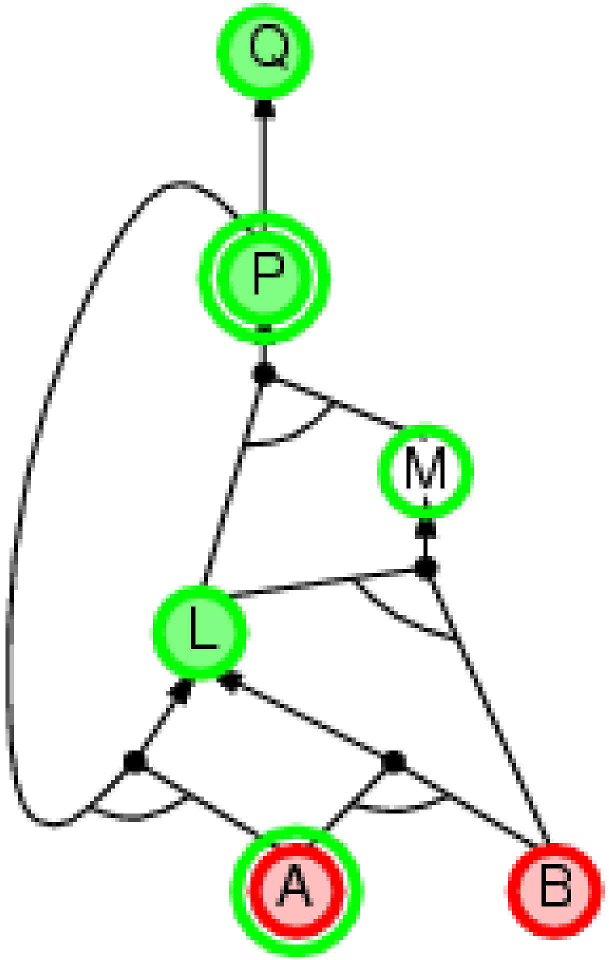
Backward chaining example



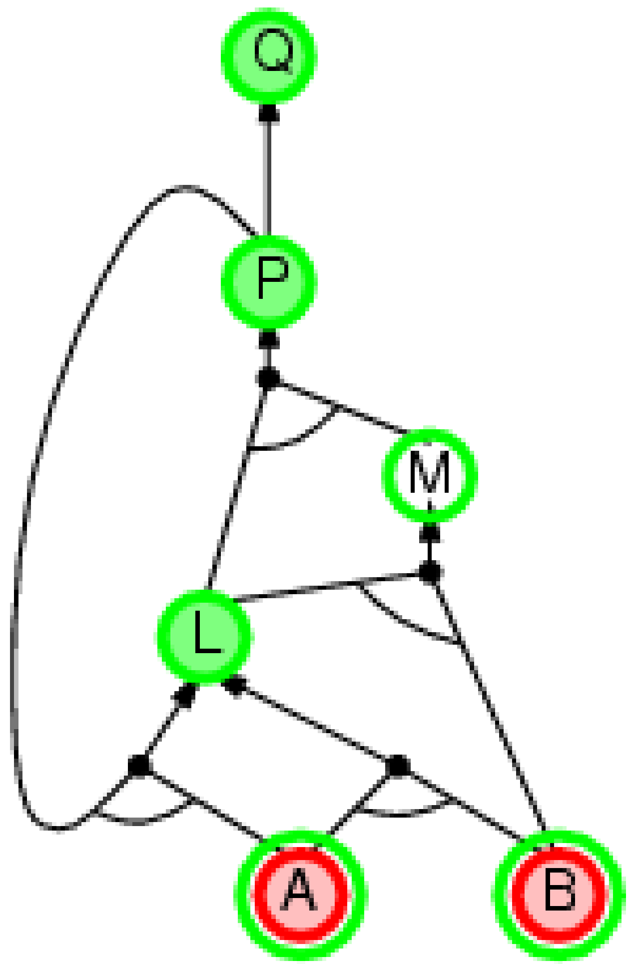
Backward chaining example



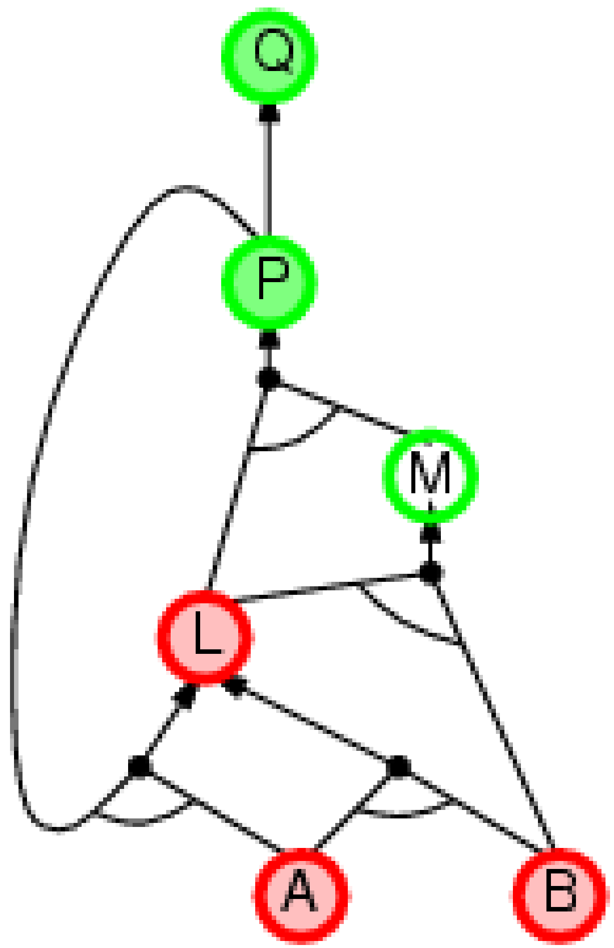
Backward chaining example



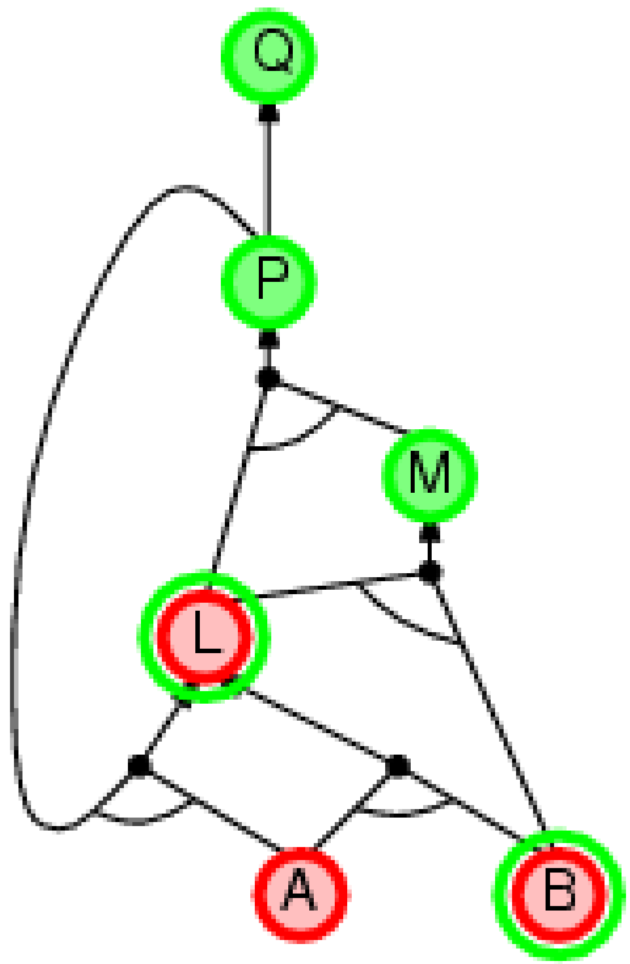
Backward chaining example



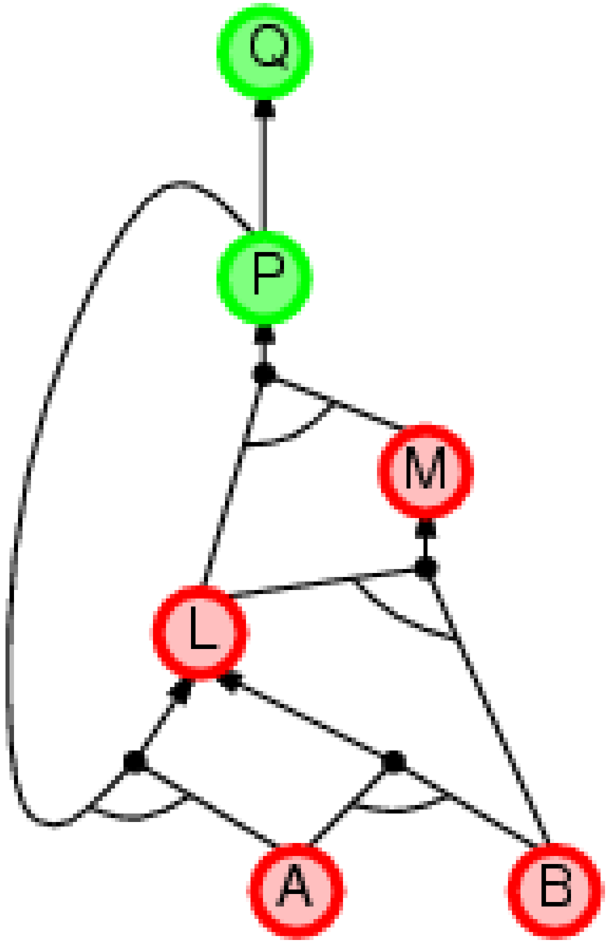
Backward chaining example



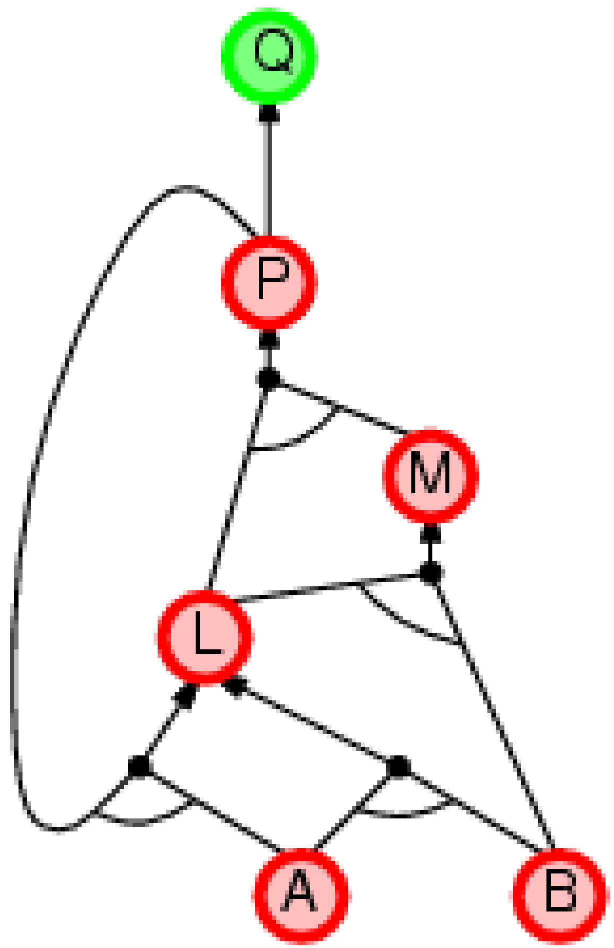
Backward chaining example



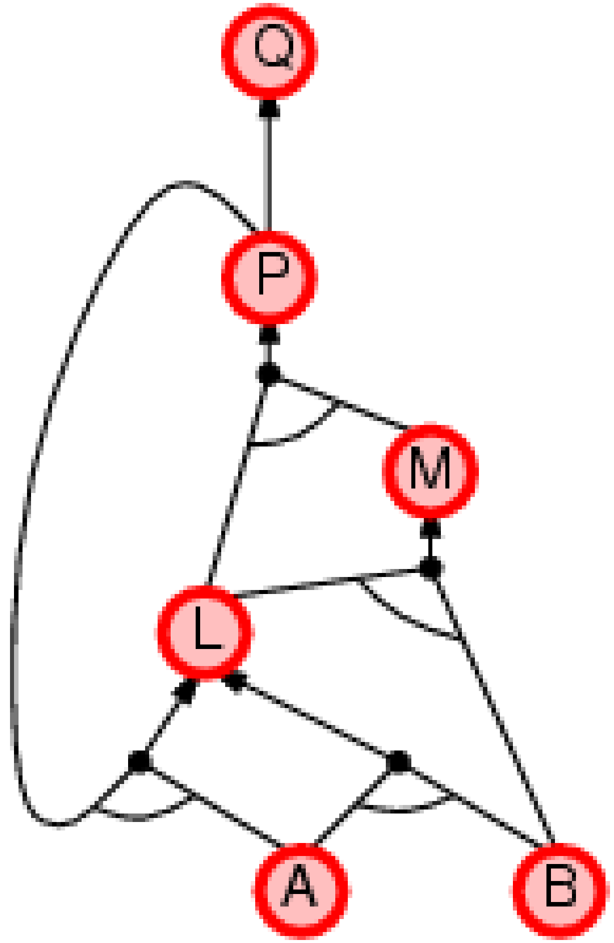
Backward chaining example



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**

Propositional inference in practice

Two families of efficient algorithms for propositional inference:

1. Apply inference rules : $KB \models \alpha$ if and only if
   - ($KB \wedge \neg \alpha$) in unsatisfiable
   - (KB $\Rightarrow \alpha$) is valid
2. Prove that a set of sentences has no model
   - ($KB \wedge \neg \alpha$) in unsatisfiable

Complete backtracking search algorithms; on CNF formulas

- 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
  - Pick an unsatisfied clause
    - With some probability pick literal to flip randomly
    - Otherwise pick a literal that minimizes the min-conflict value
  - Restart every once in awhile

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

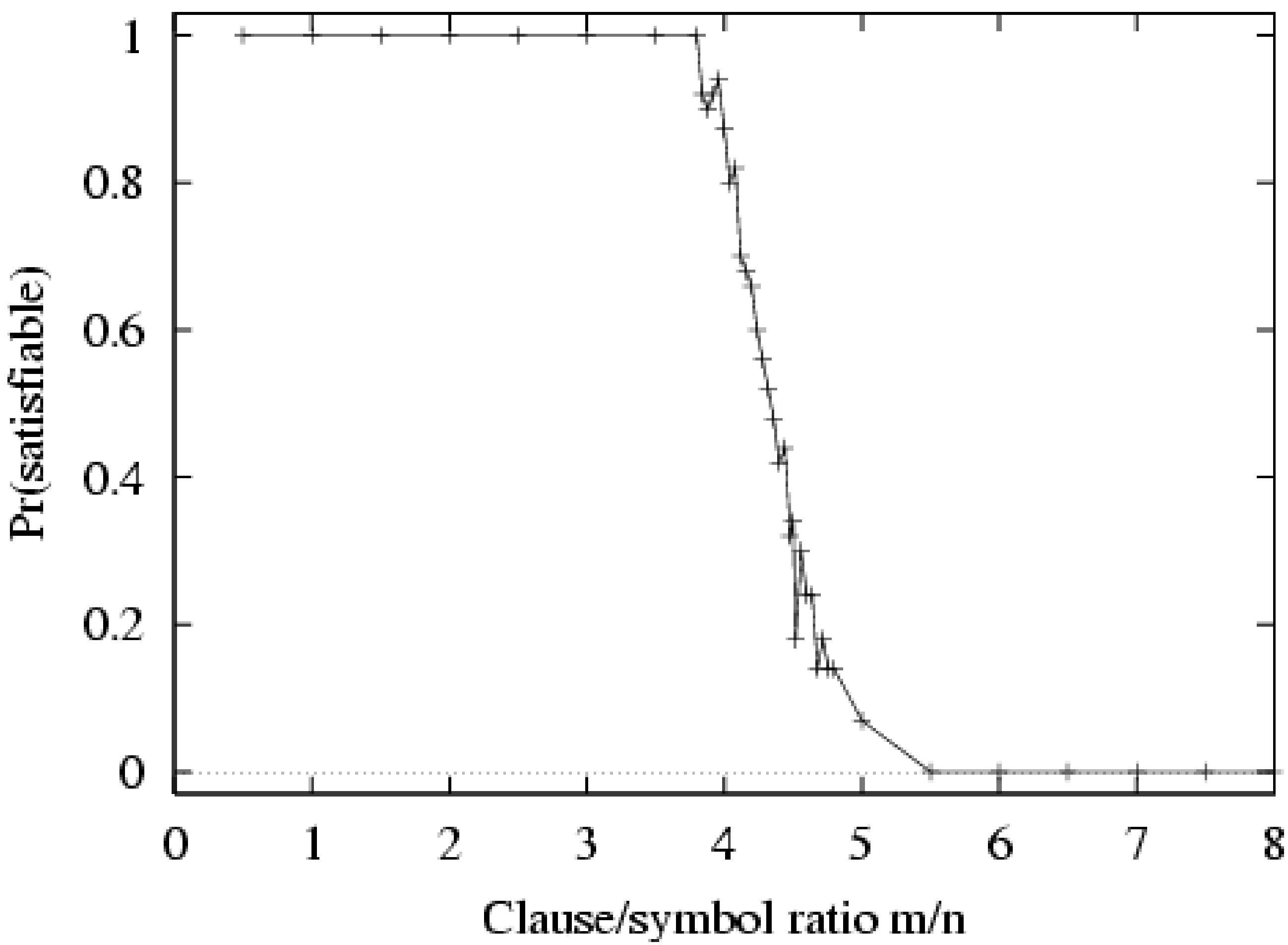
- Consider random 3-CNF sentences. e.g.,

  $(\neg D \lor \neg B \lor C) \land (B \lor \neg A \lor \neg C) \land (\neg C \lor \neg B \lor E) \land (E \lor \neg D \lor B) \land (B \lor E \lor \neg C)$

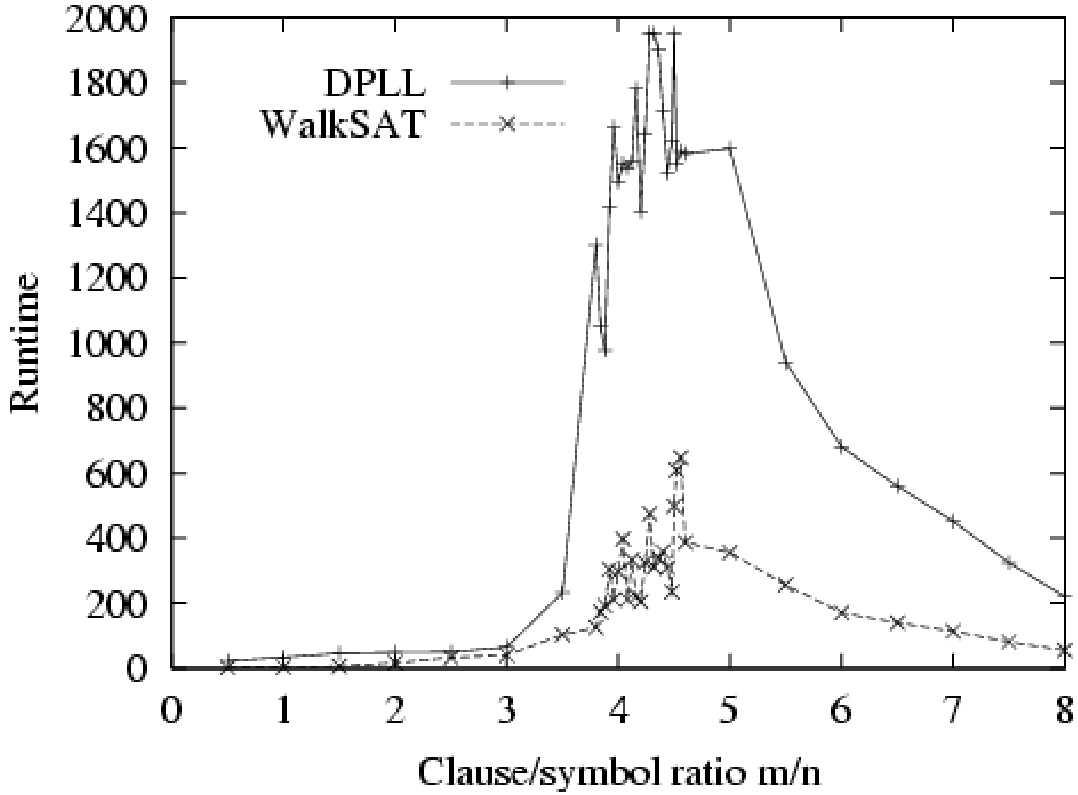
  m = number of clauses
  n = number of symbols

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

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, $\neg S_{x,y}$)

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, $(\neg P_{i,j} \wedge \neg W_{i,j})$) is *true* **or**

for some fringe square [*i*,*j*], ASK(KB, $(P_{i,j} \vee W_{i,j})$) 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*

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
- soundess: 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.

Forward, backward chaining are linear-time, complete for Horn clauses
Resolution is complete for propositional logic

Propositional logic lacks expressive power