# Logical Agents: Propositional Logic

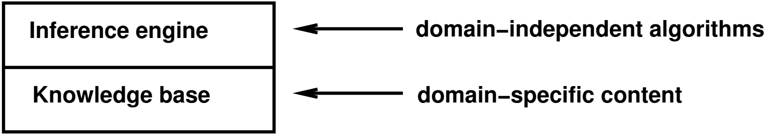
Chapter 7

# Outline

Topics:

- Knowledge-based agents
- Example domain: The Wumpus World
- Logic in general
  - models and entailment
- Propositional (Boolean) logic
- Equivalence, validity, satisfiability
- Inference rules and theorem proving
  - forward chaining
  - backward chaining
  - resolution

# Knowledge bases




- *Knowledge base* = set of *sentences* in a *formal* language
- *Declarative* approach to building an agent (or other system).
  - Declarative: Sentences express assertions about the domain
- Knowledge base operations:
  - *Tell* it what it needs to know
  - *Ask* (itself?) what to do – *query*
    - ☞ Answers should follow from the contents of the KB

# Knowledge bases

Agents can be viewed:

- at the *knowledge level*
    - i.e., *what they know*, regardless of how implemented
- at the *implementation level* (also called the *symbol level*)
    - i.e., data structures and algorithms that manipulate them

☞ Compare: abstract data type vs. data structure used to implement an ADT.

# A simple knowledge-based agent

Function KB-Agent(percept) returns an action
  static: KB, a knowledge base
          t, a counter, initially 0, indicating time

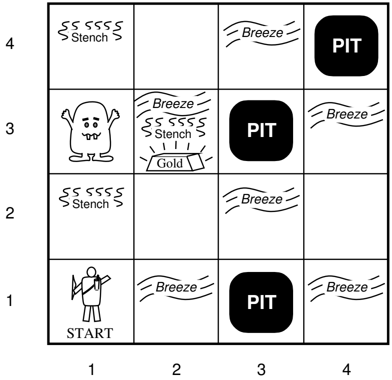
  *Tell*(KB, Make-Percept-Sentence(percept, t))
  action $\leftarrow$*Ask*(KB, Make-Action-Query(t))
  *Tell*(KB, Make-Action-Sentence(action, t))
  t $\leftarrow$t $+ 1$
  return action

# A simple knowledge-based agent

In the most general case, the agent must be able to:

- Represent states, actions, etc.
- Incorporate new percepts
- Update internal representations of the world
- Deduce hidden/implicit properties of the world
- Deduce appropriate actions

# The Wumpus World



# Wumpus World PEAS description

*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

*Actuators:* Left turn, Right turn, Forward, Grab, Release, Shoot

*Sensors:* Breeze, Glitter, Smell, Bump, Scream

# Wumpus world characterisation

Observable: ??

# Wumpus world characterisation

Observable: No – only *local* perception

Deterministic: ??

# Wumpus world characterisation

Observable: No – only *local* perception

Deterministic: Yes – outcomes exactly specified

Episodic: ??

## Wumpus world characterisation

Observable: No – only *local* perception

Deterministic: Yes – outcomes exactly specified

Episodic: No – sequential at the level of actions

Static: ??

# Wumpus world characterisation

Observable: No – 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: ??

# Wumpus world characterisation

Observable: No – 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: ??

# Wumpus world characterisation

Observable: No – only *local* perception

Deterministic: Yes – outcomes exactly specified

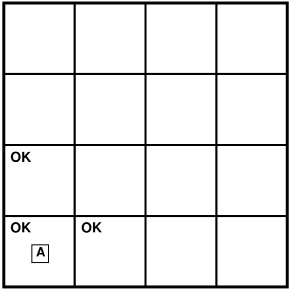
Episodic: No – sequential at the level of actions

Static: Yes – Wumpus and pits do not move

Discrete: Yes

Single-agent: Yes – Wumpus is essentially a natural feature

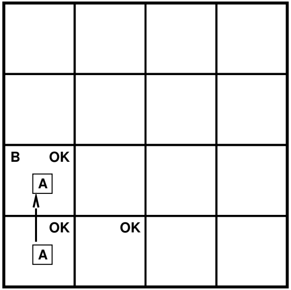
# Exploring a wumpus world




Percept:

[Stench: No, Breeze: No, Glitter: No, Bump: No, Scream: No]

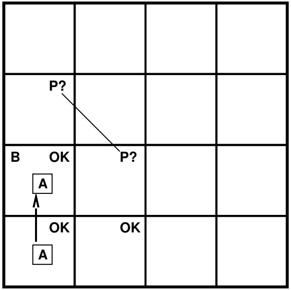
# Exploring a wumpus world




Percept:

[Stench: No, Breeze: Yes, Glitter: No, Bump: No, Scream: No]

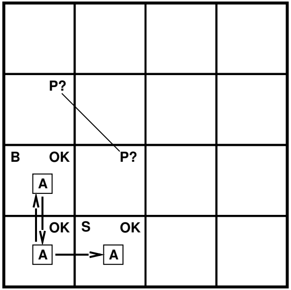
# Exploring a wumpus world




Percept:

[Stench: No, Breeze: Yes, Glitter: No, Bump: No, Scream: No]

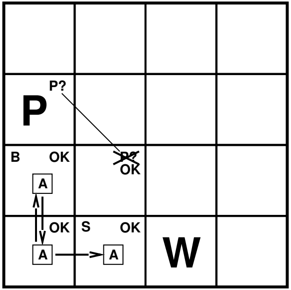
# Exploring a wumpus world




Percept:

[Stench: Yes, Breeze: No, Glitter: No, Bump: No, Scream: No]

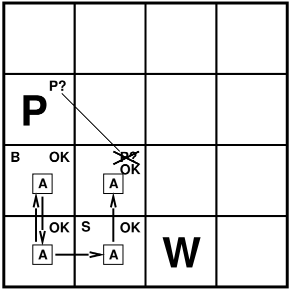
# Exploring a wumpus world




Percept:

[Stench: Yes, Breeze: No, Glitter: No, Bump: No, Scream: No]

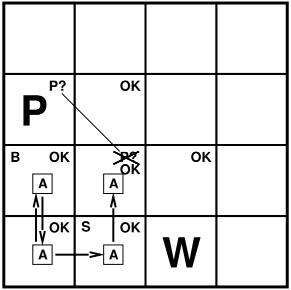
# Exploring a wumpus world




Percept:

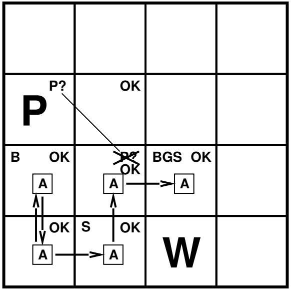
[Stench: No, Breeze: No, Glitter: No, Bump: No, Scream: No]

# Exploring a wumpus world




Percept:
[Stench: No, Breeze: No, Glitter: No, Bump: No, Scream: No]

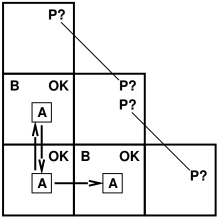
# Exploring a wumpus world




Percept:

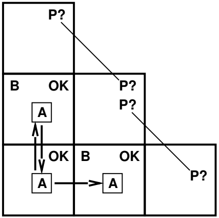
[Stench: Yes, Breeze: Yes, Glitter: Yes, Bump: No, Scream: No]

# Tight spots



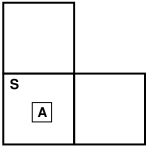

- Breeze in (1,2) and (2,1)
  $\Rightarrow$ no safe actions

# Tight spots



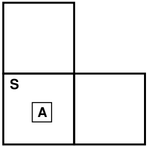

- Breeze in (1,2) and (2,1)
  $\Rightarrow$ no safe actions
- If pits are uniformly distributed, (2,2) is more likely to have a pit than (1,3) + (3,1)

# Tight spots




- Smell in (1,1)
  $\Rightarrow$ cannot safely move

# Tight spots




- Smell in (1,1)
  - $\Rightarrow$ cannot safely move
- Can use a strategy of *coercion*:
  - shoot straight ahead
  - wumpus was there $\Rightarrow$ dead $\Rightarrow$ safe
  - wumpus wasn't there $\Rightarrow$ safe

# Logic in the Wumpus World

- As the agent moves and carries out sensing actions, it performs *logical reasoning*.
  - E.g.: "If (1,3) or (2,2) contains a pit and<br>(2,2) doesn't contain a pit<br>then (1,3) must contain a pit".
- We'll use logic to represent information about the wumpus world, and to reason about this world.

# Logic in general

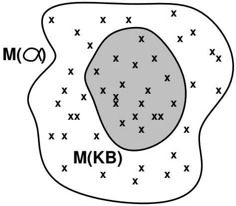
- A *logic* is a formal language for representing information such that conclusions can be drawn
- The *syntax* defines the sentences in the language
- The *semantics* define the "meaning" of sentences;
  - i.e., define *truth* of a *sentence* in a *world*
- E.g., in the language of arithmetic
  - $x + 2 \geq y$ is a sentence; $x2 + y >$ is not a sentence
  - $x + 2 \geq y$ is true iff the number $x + 2$ is not less than $y$
  - $x + 2 \geq y$ is true in a world where $x = 7,\ y = 1$
  - $x + 2 \geq y$ is false in a world where $x = 0,\ y = 6$

# Semantics: Entailment

- *Entailment* means that one thing *follows from* another:
  $KB \models \alpha$
- Knowledge base $KB$ *entails* sentence $\alpha$ if and only if:
  - $\alpha$ is true in all worlds where $KB$ is true
  - Or: if $KB$ is true then $\alpha$ must be true.
- E.g., the KB containing "the Canucks won" entails "either the Canucks won or the Leafs won"
- E.g., $x + y = 4$ entails $4 = x + y$
- Entailment is a relationship between sentences (i.e., *syntax*) that is based on *semantics*
- Note: Brains (arguably) process *syntax* (of some sort).

# Semantics: Models

- Logicians typically think in terms of *models*, which are complete descriptions of a world, with respect to which truth can be evaluated
- We say $m$ *is a model of* a sentence $\alpha$ if $\alpha$ is true in $m$
- $M(\alpha)$ is the set of all models of $\alpha$
- Thus $KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$
- E.g. $KB =$ Canucks won and Leafs won
  $\alpha =$ Canucks won



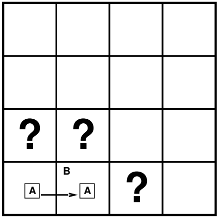
# Aside: Semantics

- Logic texts usually distinguish:
  - an *interpretation*, which is some possible world or complete state of affairs, from
  - a *model*, which is an interpretation that makes a specific sentence or set of sentences true.
- The text uses *model* in both senses (so don't be confused if you've seen the terms interpretation/model from earlier courses).
  - And if you haven't, ignore this slide!
- We'll use the text's terminology.

# Entailment in the Wumpus World

Consider the situation where the agent detects nothing in [1,1], moves right, detects a breeze in [2,1]

- Consider possible models for just the **?**'s, assuming only pits

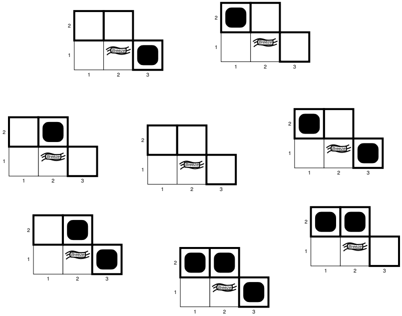



- With no information:

  3 Boolean choices $\Rightarrow$ 8 possible models

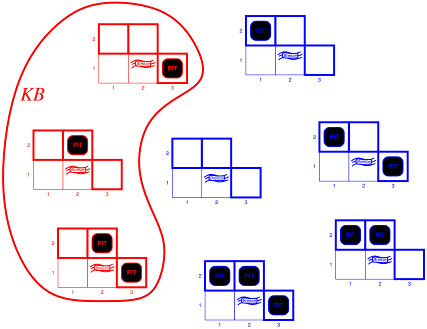
# Wumpus Models

Consider possible arrangements of pits in [1,2], [2,2], and [3,1], along with observations:



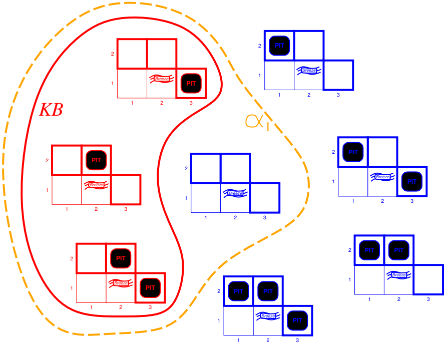
# Wumpus Models

Models of the KB:



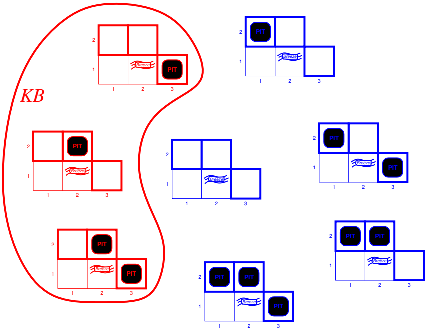

- $KB$ = wumpus-world rules + observations

# Wumpus Models



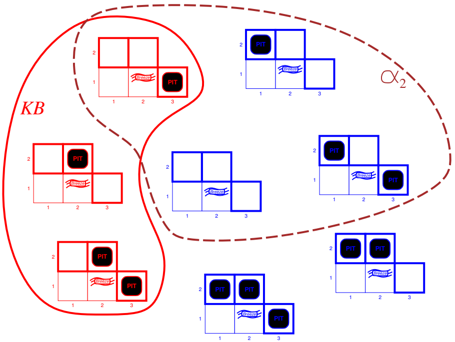

- $KB$ = wumpus-world rules + observations
- $\alpha_1$ = "[1,2] is safe", $KB \models \alpha_1$, proved by *model checking*

# Wumpus Models: Another Example




- $KB$ = wumpus-world rules + observations

# Wumpus Models: Another Example




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

# Inference

In the case of propositional logic, we can use entailment to derive conclusions by enumerating models.

- This is the usual method of computing *truth tables*
- I.e. can use entailment to do *inference*.
- In first order logic we generally can't enumerate all models (since there may be infinitely many of them and they may have an infinite domain).
- An *inference procedure* is a (syntactic) procedure for deriving some formulas from others.

# Inference

- Inference is a procedure for computing entailments.
- $KB \vdash \alpha$ = sentence $\alpha$ can be derived from $KB$ by the inference procedure
- Entailment says what things are implicitly true in a KB.
- Inference is used to *compute* things that are implicitly true.

# Inference

- Inference is a procedure for computing entailments.
- $KB \vdash \alpha$ = sentence $\alpha$ can be derived from $KB$ by the inference procedure
- Entailment says what things are implicitly true in a KB.
- Inference is used to *compute* things that are implicitly true.

Desiderata:

- *Soundness*: An inference procedure is sound if
  whenever $KB \vdash \alpha$, it is also true that $KB \models \alpha$.
- *Completeness*: An inference procedure is complete if
  whenever $KB \models \alpha$, it is also true that $KB \vdash \alpha$.

# Propositional Logic: Syntax

- Propositional logic is a simple logic – illustrates basic ideas
- We first specify the *proposition symbols* or *(atomic) sentences*: $P_1$, $P_2$ etc.
- Then we define the language:

  If $S_1$ and $S_2$ are sentences then:

  - $\neg S_1$ is a sentence (*negation*)
  - $S_1 \wedge S_2$ is a sentence (*conjunction*)
  - $S_1 \vee S_2$ is a sentence (*disjunction*)
  - $S_1 \Rightarrow S_2$ is a sentence (*implication*)
  - $S_1 \equiv S_2$ is a sentence (*biconditional*)

# Propositional Logic: Semantics

- Each model assigns true or false to each proposition symbol
- E.g.: $P_{1,2} \leftarrow \mathit{true}$, $P_{2,2} \leftarrow \mathit{true}$, $P_{3,1} \leftarrow \mathit{false}$
  (With these symbols, 8 possible models, can be enumerated.)
- 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 |
| $S_1 \equiv S_2$ | 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.,
  $\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = \mathit{true} \wedge (\mathit{false} \vee \mathit{true}) = \mathit{true} \wedge \mathit{true} = \mathit{true}$

# 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* |

# Wumpus World Sentences

- 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 $[i,j]$.
- Information from sensors: $\neg P_{1,1}$, $\neg B_{1,1}$, $B_{2,1}$
- Also know: "pits cause breezes in adjacent squares"

## Wumpus World Sentences

- 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 $[i,j]$.
- Information from sensors: $\neg P_{1,1},\ \neg B_{1,1},\ B_{2,1}$
- "A square is breezy *if and only if* there is an adjacent pit"

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

  - Note: $B_{1,1}$ has no "internal structure" – think of it as a string.
  - So must write one formula for each square.

# Wumpus World Sentences

- 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 $[i,j]$.
- Information from sensors: $\neg P_{1,1}$, $\neg B_{1,1}$, $B_{2,1}$
- "A square is breezy *if and only if* there is an adjacent pit"

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

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

  - Note: $B_{1,1}$ has no "internal structure" – think of it as a string.
  - So must write one formula for each square.

- Using logic can conclude $\neg P_{1,2}$ and $\neg P_{2.1}$ from $\neg B_{1,1}$.
- Note, if you wrote the above as:

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

  (I.e. "A breeze implies a pit in an adjacent square")

  you could not derive $\neg P_{1,2}$ and $\neg P_{2.1}$ from $\neg B_{1,1}$.

  ☞ Crucial to express *all* information

## Wumpus World KB

For the part of the Wumpus world we're looking at, let

$$KB = \{R_1, R_2, R_3, R_4, R_5\}$$

where

$$\begin{array}{lll} R_1 & \text{is} & \neg P_{1,1} \\ R_2 & \text{is} & B_{1,1} \equiv (P_{1,2} \vee P_{2,1}) \\ R_3 & \text{is} & B_{2,1} \equiv (P_{1,1} \vee P_{2,2} \vee P_{3,1}) \\ R_4 & \text{is} & \neg B_{1,1} \\ R_5 & \text{is} & B_{2,1} \end{array}$$

# Truth Tables for Inference

| $B_{1,1}$ | $B_{2,1}$ | $P_{1,1}$ | $P_{1,2}$ | $P_{2,1}$ | $P_{2,2}$ | $P_{3,1}$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $KB$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $f$ | $f$ | $f$ | $f$ | $f$ | $f$ | $f$ | $t$ | $t$ | $t$ | $t$ | $f$ | $f$ |
| $f$ | $f$ | $f$ | $f$ | $f$ | $f$ | $t$ | $t$ | $t$ | $f$ | $t$ | $f$ | $f$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $f$ | $t$ | $f$ | $f$ | $f$ | $f$ | $f$ | $t$ | $t$ | $f$ | $t$ | $t$ | $f$ |
| $f$ | $t$ | $f$ | $f$ | $f$ | $f$ | $t$ | $t$ | $t$ | $t$ | $t$ | $t$ | $\underline{t}$ |
| $f$ | $t$ | $f$ | $f$ | $f$ | $t$ | $f$ | $t$ | $t$ | $t$ | $t$ | $t$ | $\underline{t}$ |
| $f$ | $t$ | $f$ | $f$ | $f$ | $t$ | $t$ | $t$ | $t$ | $t$ | $t$ | $t$ | $\underline{t}$ |
| $f$ | $t$ | $f$ | $f$ | $t$ | $f$ | $f$ | $t$ | $f$ | $f$ | $t$ | $t$ | $f$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $t$ | $t$ | $t$ | $t$ | $t$ | $t$ | $t$ | $f$ | $t$ | $t$ | $f$ | $t$ | $f$ |

- Enumerate rows (different assignments to symbols),
- For $KB \models \alpha$, if KB is true in row, check that $\alpha$ is too

# Inference by Enumeration

Function TT-Entails?(KB, $\alpha$) returns true or false

inputs: KB, the knowledge base, a sentence in propositional logic

$\alpha$ the query, a sentence in propositional logic

symbols $\leftarrow$a list of the proposition symbols in KB and $\alpha$

return TT-Check-All(KB, $\alpha$, symbols, [])

# Inference by Enumeration

Function TT-Check-All(KB, $\alpha$, symbols, model) returns true or false
    if *Empty?*(symbols) then
        if PL-True?(KB, model) then return PL-True?($\alpha$, model)
            else return true
    else do
        $P$ ←*First*(symbols); rest ←*Rest*(symbols)
        return TT-Check-All(KB, $\alpha$, rest, model $\cup$ {P $=$ true}) and
                TT-Check-All(KB, $\alpha$, rest, model $\cup$ {P $=$ false})

- Depth-first enumeration of all models
    - Hence, sound and complete
- Algorithm is $O(2^n)$ for $n$ symbols; problem is *co-NP-complete*

# Other Means of Computing Logical Inference

- We'll briefly consider other means of computing entailments:
  - Resolution theorem proving
  - Specialised rule-based approaches
- But first, some more terminology

# Logical Equivalence

- Two sentences are *logically equivalent* iff true in same models:
  $\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$
- The following should be familiar:

$$
\begin{array}{rcl}
(\alpha \wedge \beta) & \equiv & (\beta \wedge \alpha) \\
(\alpha \vee \beta) & \equiv & (\beta \vee \alpha) \\
((\alpha \wedge \beta) \wedge \gamma) & \equiv & (\alpha \wedge (\beta \wedge \gamma)) \\
((\alpha \vee \beta) \vee \gamma) & \equiv & (\alpha \vee (\beta \vee \gamma)) \\
\neg(\neg\alpha) & \equiv & \alpha \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\beta \Rightarrow \neg\alpha) \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\alpha \vee \beta) \\
(\alpha \equiv \beta) & \equiv & ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \\
\neg(\alpha \wedge \beta) & \equiv & (\neg\alpha \vee \neg\beta) \\
\neg(\alpha \vee \beta) & \equiv & (\neg\alpha \wedge \neg\beta) \\
(\alpha \wedge (\beta \vee \gamma)) & \equiv & ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \\
(\alpha \vee (\beta \wedge \gamma)) & \equiv & ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))
\end{array}
$$

# Validity and Satisfiability

- A sentence is *valid* if it is true in *all* models,

  e.g., $A \lor \neg A, \quad A \Rightarrow A, \quad (A \land (A \Rightarrow B)) \Rightarrow B$

## Validity and Satisfiability

- A sentence is *valid* if it is true in *all* models,
  e.g., $A \vee \neg A, \quad A \Rightarrow A, \quad (A \wedge (A \Rightarrow B)) \Rightarrow B$
- Validity is connected to inference via the *Deduction Theorem*:
  $KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

# Validity and Satisfiability

- A sentence is *valid* if it is true in *all* models,
  e.g., $A \vee \neg A, \quad A \Rightarrow A, \quad (A \wedge (A \Rightarrow B)) \Rightarrow B$
- 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., $A \vee B, \quad C$

# Validity and Satisfiability

- A sentence is *valid* if it is true in *all* models,
  e.g., $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$
- 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., $A \wedge \neg A$

# Validity and Satisfiability

- A sentence is *valid* if it is true in *all* models,
  e.g., $A \lor \neg A, \quad A \Rightarrow A, \quad (A \land (A \Rightarrow B)) \Rightarrow B$
- 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., $A \lor B, \qquad C$
- A sentence is *unsatisfiable* if it is true in *no* models
  e.g., $A \land \neg A$
- Satisfiability is connected to inference via the following:
  $KB \models \alpha$ if and only if $(KB \land \neg\alpha)$ is unsatisfiable
    - I.e., prove $\alpha$ by *reductio ad absurdum*

# Validity and Satisfiability

- A sentence is *valid* if it is true in *all* models,

  e.g., $A \vee \neg A, \quad A \Rightarrow A, \quad (A \wedge (A \Rightarrow B)) \Rightarrow B$
- 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., $A \vee B, \qquad C$
- 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
  - I.e., prove $\alpha$ by *reductio ad absurdum*
- What often proves better for determining $KB \models \alpha$ is to show that $KB \wedge \neg\alpha$ is unsatisfiable.

## General Propositional Inference: Resolution

Resolution is a rule of inference defined for *Conjunctive Normal Form* (CNF)

- *CNF: conjunction* of *disjunctions* of *literals*
- A *clause* is a *disjunctions* of *literals*.
- E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$.
  ☞ Write as: $(A \vee \neg B)$, $(B \vee \neg C \vee \neg D)$

# Resolution

- *Resolution* inference rule:

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

  where $\ell_i$ and $m_j$ are complementary literals. (I.e. $\ell_i \equiv \neg m_j$.)

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

# Resolution

- *Resolution* inference rule:

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

  where $\ell_i$ and $m_j$ are complementary literals. (I.e. $\ell_i \equiv \neg m_j$.)

- E.g., $\dfrac{P_{1,3} \vee P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}}$
- If you can derive the "empty clause" from a set of clauses $C$, then $C$ is unsatisfiable.
  - E.g. $\{A, \neg A \vee B, \neg B\}$ is unsatisfiable.

# Resolution

- *Resolution* inference rule:

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

  where $\ell_i$ and $m_j$ are complementary literals. (I.e. $\ell_i \equiv \neg m_j$.)

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

- If you can derive the "empty clause" from a set of clauses $C$, then $C$ is unsatisfiable.
  - E.g. $\{A, \neg A \vee B, \neg B\}$ is unsatisfiable.

- Resolution is sound and complete for propositional logic

- I.e. $KB \models \alpha$ iff
  $KB \wedge \neg\alpha$ is unsatisfiable iff
  the empty clause can be obtained from $KB \wedge \neg\alpha$
  by resolution

# Using resolution to compute entailments

To show whether $KB \models \alpha$, show instead that $KB \wedge \neg\alpha$ is unsatisfiable:

1. Convert $KB \wedge \neg\alpha$ into conjunctive normal form.
2. Use resolution to determine whether $KB \wedge \neg\alpha$ is unsatisfiable.
3. If so then $KB \models \alpha$; otherwise $KB \not\models \alpha$.

# Conversion to CNF

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

# Conversion to CNF

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

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

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

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

1. Eliminate $\equiv$, replacing $\alpha \equiv \beta$ 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 $\neg\alpha \vee \beta$.
   $$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$$

# Conversion to CNF

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

1. Eliminate $\equiv$, replacing $\alpha \equiv \beta$ 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:
   $$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$$

# Conversion to CNF

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

1. Eliminate $\equiv$, replacing $\alpha \equiv \beta$ 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 ($\vee$ over $\wedge$) and flatten:
   $$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$$

# Conversion to CNF

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

1. Eliminate $\equiv$, replacing $\alpha \equiv \beta$ 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 ($\vee$ over $\wedge$) and flatten:
   $$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$$

For resolution, then write as

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

# Resolution Algorithm

Function PL-Resolution(KB, $\alpha$) returns true or false

inputs: KB, the knowledge base, a sentence in propositional logic

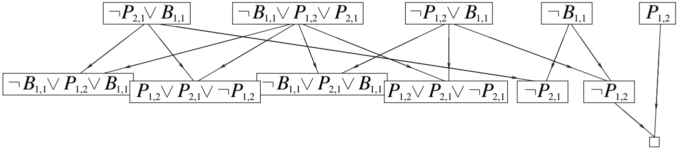
$\alpha$, the query, a sentence in propositional logic

```
clauses ←the set of clauses in CNF(KB ∧ ¬α)
loop do
   if clauses contains the empty clause then return true
   if Ci, Cj are resolvable clauses where
           PL-Resolve(Ci, Cj) ∉ clauses
      then clauses ←clauses ∪ PL-Resolve(Ci, Cj)
      else return false
```

☞ Note that the algorithm in the text is buggy

# Resolution Example

- E.g.: $KB = (B_{1,1} \equiv (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$,
  $\alpha = \neg P_{1,2}$
- Show $KB \models \alpha$ by showing that $KB \wedge \neg\alpha$ is unsatisfiable:



## Resolution: Continued

There is a great deal that can be done to improve the basic algorithm:

- Unit resolution: propagate unit clauses (e.g. $\neg B_{1,1}$) as much as possible.
  - Note that this correspoinds to the *minimum remaining values* heuristic in constraint satisfaction!
- Eliminate tautologies
- Eliminate redundant clauses
- Eliminate clauses with literal $\ell$ where the complement of $\ell$ doesn't appear elsewhere.
- Set of support: Do resolutions on clauses with ancestor in $\neg\alpha$.
  - I.e. keep a focus on the goal.

# Specialised Inference: Rule-Based Reasoning

- We consider a very useful, restricted case: *Horn Form*
  - KB = *conjunction* of *Horn clauses*
- Horn clause =
  - proposition symbol; or
  - A rule of the form:

    (conjunction of symbols) $\Rightarrow$ symbol
- E.g., $C$, $(B \Rightarrow A)$, $(C \wedge D \Rightarrow B)$
  Not: $(\neg B \Rightarrow A)$, $(B \vee A)$
- Use Horn clauses to derive individual facts (or atoms), not arbitrary formulas.

# Horn clauses

Technically a Horn clause is a *clause* or disjunction of literals, with *at most* one positive literal.

- I.e. of form $A_0 \vee \neg A_1 \vee \cdots \vee \neg A_n$ or
  $\neg A_1 \vee \cdots \vee \neg A_n$
- These can be written: $A_1 \wedge \cdots \wedge A_n \Rightarrow A_0$ or
  $A_1 \wedge \cdots \wedge A_n \Rightarrow \bot$
- We won't bother with rules of the form $A_1 \wedge \cdots \wedge A_n \Rightarrow \bot$
  - Rules of this form are called *integrity constraints*.
  - They don't allow new facts to be derived, but rather rule out certain combinations of facts.

# Reasoning with Horn clauses

- *Modus Ponens* (for Horn form): Complete for Horn KBs

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

## Reasoning with Horn clauses

- *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*.
- *Forward chaining:* Iteratively add new derived facts

# Reasoning with Horn clauses

- *Modus Ponens* (for Horn form): Complete for Horn KBs

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

- Can be used with *forward chaining* or *backward chaining*.
- *Forward chaining:* Iteratively add new derived facts
- *Backward chaining:* From a query, work backwards through the rules to known facts.

# Reasoning with Horn clauses

- *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*.
- *Forward chaining:* Iteratively add new derived facts
- *Backward chaining:* From a query, work backwards through the rules to known facts.
- These algorithms are very natural; forward chaining runs in *linear* time

## Example

KB:

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

# Forward chaining

Idea:

- Fire any rule whose premises are satisfied in the $KB$,
- Add its conclusion to the $KB$, until query is found

# Forward chaining algorithm

**Procedure:**

$C := \{\}$;

*repeat*

*choose* $r \in A$ *such that*

$r$ *is* '$b_1 \wedge \cdots \wedge b_m \Rightarrow h$'

$b_i \in C$ *for all* $i$, *and*

$h \notin C$;

$C := C \cup \{h\}$

*until no more choices*

# Forward chaining example

KB:

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

Query $Q$:

# Forward chaining example

KB:

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

Query $Q$:

- From $A$ and $B$, conclude $L$

# Forward chaining example

KB:

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

Query $Q$:

- From $A$ and $B$, conclude $L$
- From $L$ and $B$, conclude $M$

# Forward chaining example

KB:

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

Query $Q$:

- From $A$ and $B$, conclude $L$
- From $L$ and $B$, conclude $M$
- From $L$ and $M$, conclude $P$

# Forward chaining example

KB:

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

Query $Q$:

- From $A$ and $B$, conclude $L$
- From $L$ and $B$, conclude $M$
- From $L$ and $M$, conclude $P$
- From $P$ conclude $Q$

# Backward chaining

- We won't develop an algorithm for backward chaining, but will just consider it informally.
- Idea with backward chaining:

  Start from query $q$ and work backwards.
- To prove $q$ by BC:
  - check if $q$ is known already;
  - otherwise 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

KB:

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

Query $Q$:

# Backward chaining example

KB:

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

Query $Q$:

- Establish $P$ as a subgoal.

# Backward chaining example

KB:

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

Query $Q$:

- Establish $P$ as a subgoal.
- Can prove $P$ by proving $L$ and $M$

## Backward chaining example

KB:

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

Query $Q$:

- Establish $P$ as a subgoal.
- Can prove $P$ by proving $L$ and $M$
- For $M$:
  - Can prove $M$ if we can prove $B$ and $L$

# Backward chaining example

KB:

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

Query $Q$:

- Establish $P$ as a subgoal.
- Can prove $P$ by proving $L$ and $M$
- For $M$:
    - Can prove $M$ if we can prove $B$ and $L$
    - $B$ is known to be true

# Backward chaining example

KB:

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

Query $Q$:

- Establish $P$ as a subgoal.
- Can prove $P$ by proving $L$ and $M$
- For $M$:
  - Can prove $M$ if we can prove $B$ and $L$
  - $B$ is known to be true
  - $L$ can be proven by proving $A$ and $B$.

# Backward chaining example

KB:

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

Query $Q$:

- Establish $P$ as a subgoal.
- Can prove $P$ by proving $L$ and $M$
- For $M$:
    - Can prove $M$ if we can prove $B$ and $L$
    - $B$ is known to be true
    - $L$ can be proven by proving $A$ and $B$.
    - $A$ and $B$ are known to be true

# Backward chaining example

KB:

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

Query $Q$:

- Establish $P$ as a subgoal.
- Can prove $P$ by proving $L$ and $M$
- For $M$:
    - Can prove $M$ if we can prove $B$ and $L$
    - $B$ is known to be true
    - $L$ can be proven by proving $A$ and $B$.
    - $A$ and $B$ are known to be true
- For $L$:
    - $L$ can be proven by proving $A$ and $B$.
    - $A$ and $B$ are known to be true

## Backward chaining example

KB:

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

Query $Q$:

- Establish $P$ as a subgoal.
- Can prove $P$ by proving $L$ and $M$
- For $M$:
    - Can prove $M$ if we can prove $B$ and $L$
    - $B$ is known to be true
    - $L$ can be proven by proving $A$ and $B$.
    - $A$ and $B$ are known to be true
- For $L$:
    - $L$ can be proven by proving $A$ and $B$.
    - $A$ and $B$ are known to be true
- $L$ and $M$ are true, thus $P$ is true, thus $Q$ is true

# Forward vs. backward chaining

- FC is *data-driven*, cf. automatic, unconscious processing,
  - E.g., object recognition, routine decisions
  - May do lots of work that is irrelevant to the goal
  - Good for reactive agents

# Forward vs. backward chaining

- FC is *data-driven*, cf. automatic, unconscious processing,
  - E.g., object recognition, routine decisions
  - May do lots of work that is irrelevant to the goal
  - Good for reactive agents
- BC is *goal-driven*, appropriate for problem-solving,
  - E.g., Where are my keys? How do I get a job?
  - Complexity of BC can be *much less* than linear in size of KB
  - Can also sometimes be exponential in size of KB
  - Good for question-answering and explanation

# 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

# Summary (Continued)

- 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 complete for Horn clauses.
- Forward chaining is linear-time for Horn clauses.
- Propositional logic lacks expressive power