Logical Agents

Bùi Tiến Lên

2022

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

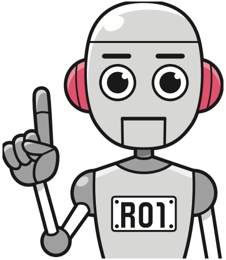
Contents

Knowledge-based Agents

Problem-solving agents

- The problem-solving agents know things, but only in a very limited, inflexible sense.
  - E.g., the 8-puzzle agent cannot deduce that with odd parity cannot be reached from states with even parity
- CSP enables some parts of the agent to work domain-independently
  - Represent states as assignments of values to variables
  - Allow for more efficient algorithms

Motivation

Tell information →  ← **Ask** questions

Which language?

Need to:

- Digest **heterogenous** information
- Reason **deeply** with that information

Language

- Natural languages (informal):
  - English: `Two divides even numbers.`
  - Vietnamese: `Hai là số chẵn.`
- Programming languages (formal):
  - Python: `def even(x): return x % 2 == 0`
  - C++: `bool even(int x) { return x % 2 == 0; }`
- **Logical languages (formal)**:
  - First-order-logic: $\forall x\ \mathrm{Even}(x) \rightarrow \mathrm{Divides}(x, 2)$

Two goals of a logic language

- **Represent** knowledge about the world

- **Reason** with that knowledge

Strength and Problem

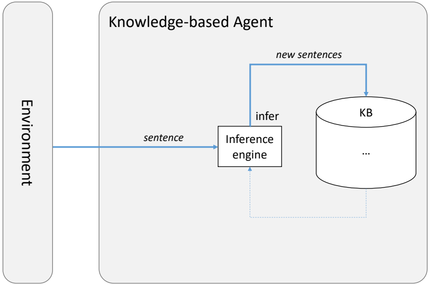
- **Strength**: provides expressiveness in a compact way
- **Problem 1**: deterministic, didn't handle **uncertainty** (probability addresses this)
- **Problem 2**: rule-based, didn't allow fine tuning from **data** (machine learning addresses this)

Knowledge-based agents

- Supported by **logic**
- Knowledge-based agents can combine and recombine information to suit myriad purposes.
  - Accept new tasks in the form of explicitly described goals
  - Achieve competence by learning new knowledge of the environment
  - Adapt to changes by updating the relevant knowledge

Knowledge-based agents (cont.)

- **Knowledge base** (KB): A set of sentences or facts in a *formal* language
    - Each sentence represents some assertion about the world.
    - Axiom = the sentence that is not derived from other sentences
- **Inference**: Using **inference engine** to derive (infer) new sentences from old ones
    - Add new sentences to the knowledge base and query what is known



A generic knowledge-based agent

```
function KB-AGENT(percept) returns an action
persistent: KB, a knowledge base
            t, a counter, initially 0, indicating time
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

- Given a percept, the agent adds the percept to its knowledge base, asks the knowledge base for the best action, and tells the knowledge base that it has in fact taken that action.

Building an Agent

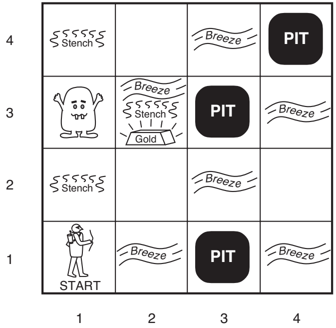
- Procedural approach
  - Encode desired behaviors directly as program code.
- **Declarative** approach to building an agent
  - **Tell** it what it needs to know, then it can **Ask** itself what to do – answers should follow from the KB
- Combined approach → Partially autonomous
- Learning approach → Fully autonomous
  - Provide a knowledge-based agent with mechanisms that allow it to learn for itself

The Wumpus World

Wumpus World PEAS description

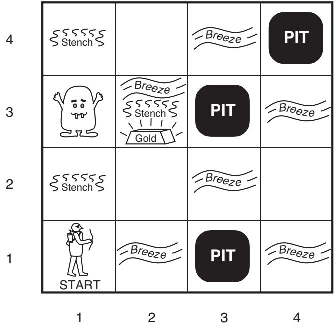
The **wumpus world** is a cave consisting of rooms connected by passageways

- **Performance measure**
  - $+1000$ for climbing out of the cave with gold
  - -1000 for falling into a pit or being eaten by the wumpus
  - -1 each action taken
  - -10 for using the arrow
  - The game ends when agent dies or climbs out of the cave



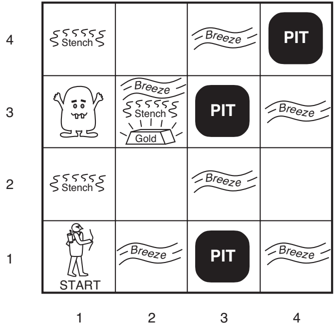
Wumpus World PEAS description (cont.)

- **Environment**
  - A 4×4 grid of rooms
  - Agent starts in the square $[1, 1]$, facing to the right
  - The locations of **gold** and **wumpus** are random
  - Each square can be a **pit**, with probability 0.2



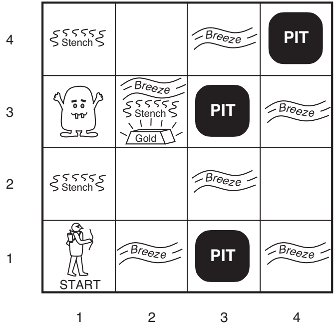
Wumpus World PEAS description (cont.)

- **Actuators**: The agent can
  - *Forward*
  - *Left turn* by 90°
  - *Right turn* by 90°
  - *Shooting* kills wumpus if you are facing it (the agent has only one arrow)
  - *Grabbing* picks up gold if in same square
  - *Releasing* drops the gold in same square



Wumpus World PEAS description (cont.)

- **Sensors**: The agent has five sensors
  - In the square containing the wumpus and in the directly (not diagonally) adjacent squares, the agent will perceive a *Stench*.
  - In the squares directly adjacent to a pit, the agent will perceive a *Breeze*.
  - In the square where the gold is, the agent will perceive a *Glitter*.
  - When an agent walks into a wall, it will perceive a *Bump*.
  - When the wumpus is killed, it emits a woeful *Scream* that can be perceived anywhere in the cave



[*Stench*, *Breeze*, *None*, *None*, *None*]

Characterize the Wumpus World

- Fully 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

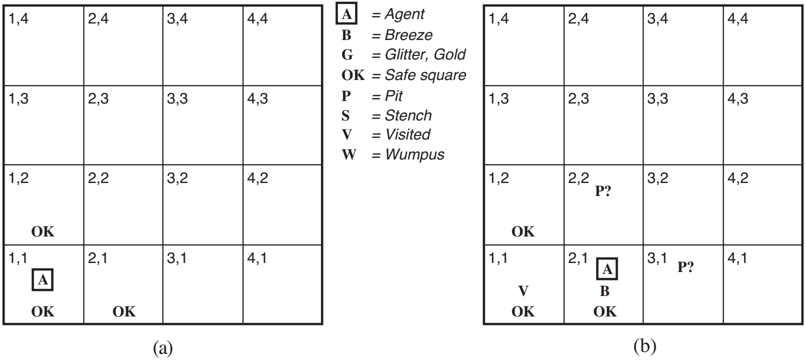


Figure 1: The first step taken by the agent in the wumpus world. (a) The initial situation, after percept [*None, None, None, None, None*]. (b) After one move, with percept [*None, Breeze, None, None, None*].

Exploring a wumpus world (cont.)

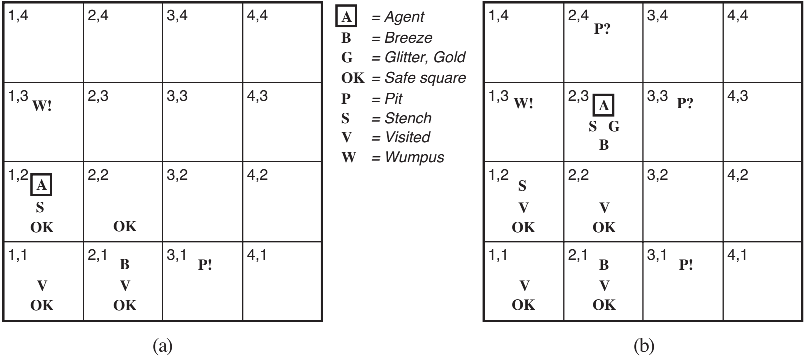
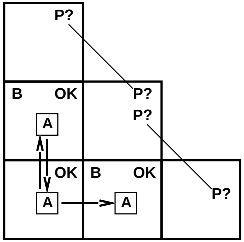
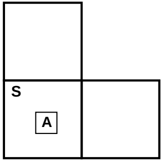


Figure 2: Two later stages in the progress of the agent. (a) After the third move, with percept [*Stench, None, None, None, None*]. (b) After the fifth move, with percept [*Stench, Breeze, Glitter, None, None*].

More



- Breeze in (1,2) and (2,1) $\implies$ no safe actions
- Assuming pits uniformly distributed, (2,2) has pit with probability 0.86 vs. 0.31



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

Logic

Logic

Concept 1

Logics are formal languages for representing information such that conclusions can be drawn

- **Syntax** defines the **sentences** (**statements**, **formulas**) in the language
- **Semantics** define the "meaning" of sentences; i.e., define **truth** of a sentence in a world

The language of arithmetic

- $x + 2 \geq y$ is a sentence
- $x^2 + y >$ is not a sentence
- $x + 2 \geq y$ is true in a world where $x = 7, y = 1$
- $x + 2 \geq y$ is false in a world where $x = 0, y = 6$
- $x + 2 \geq y$ is true iff the number $x + 2$ is no less than the number y

Models

Concept 2

Models are formally structured worlds with respect to which truth can be evaluated

- A model m in propositional logic is an assignment of truth values to propositional symbols

- 3 propositional symbols: A, B, C
- $2^3 = 8$ possible models m_i:

Model	A	B	C	Model	A	B	C
m_1	true	true	true	m_5	false	true	true
m_2	true	true	false	m_6	false	true	false
m_3	true	false	true	m_7	false	false	true
m_4	true	false	false	m_8	false	false	false

Interpretation function/semantic

Concept 3

Let α be a sentence and m be a model. An **interpretation function** $\mathcal{I}(\alpha, m)$ returns:

- **true** (1) say that m **satisfies** α or sometimes m is a **model** of α
- **false** (0) say that m does not satisfies α

- Given a sentence α, $\mathcal{M}(\alpha)$ is the set of all models of α

Entailment

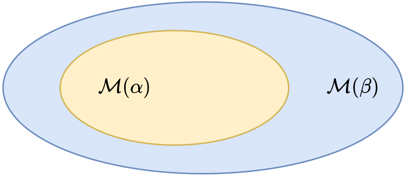
Concept 4

Let α, β be sentences, α **entails** β

$$\alpha \models \beta \tag{1}$$

iff in every model where α is true, β is also true or

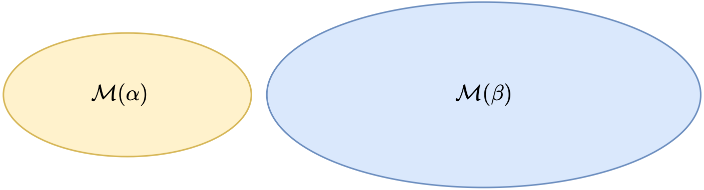
$$\mathcal{M}(\alpha) \subseteq \mathcal{M}(\beta) \tag{2}$$



Contradiction

Concept 5

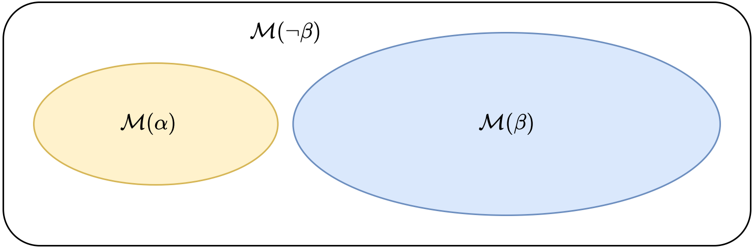
Let α, β be sentences, α **contradicts** β iff $\mathcal{M}(\alpha) \cap \mathcal{M}(\beta) = \emptyset$.



Contradiction vs. entailment

Theorem 1

Let α, β be sentences, α contradicts β iff α entails $\neg\beta$.

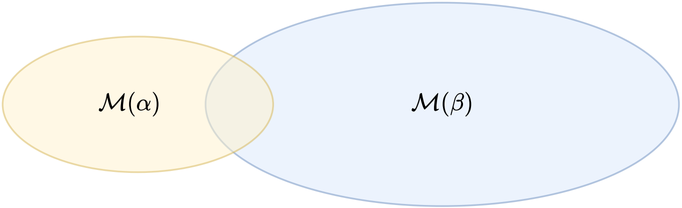


Contingency

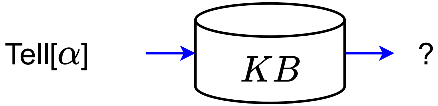
Concept 6

Let α, β be sentences, β is **contingent** on α iff

$$\emptyset \neq \mathcal{M}(\alpha) \cap \mathcal{M}(\beta) \neq \mathcal{M}(\alpha) \tag{3}$$



Tell operation



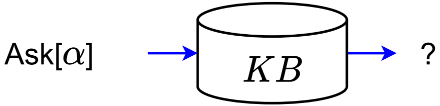
Tell: $\alpha =$ "It is raining".

$$\text{TELL}[\text{KB}, \text{Rain}]$$

Possible responses:

- **Already knew that**: entailment ($KB \models \alpha$)
- **Don't believe that**: contradiction ($KB \models \neg\alpha$)
- **Learned something new** (update KB): contingent $KB \leftarrow KB, \alpha$

Ask operation



Ask: $\alpha =$ "Is it raining?".

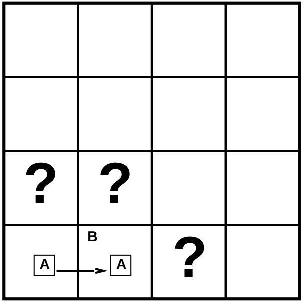
$$\textsc{Ask}[\text{KB, Rain}]$$

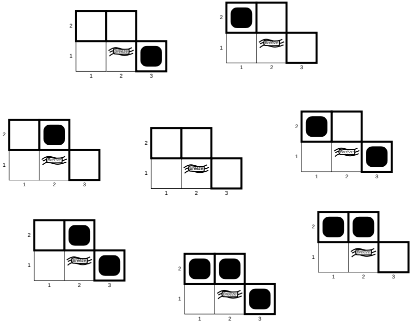
Possible responses:

- **Yes**: entailment ($KB \models \alpha$)
- **No**: contradiction ($KB \models \neg\alpha$)
- **I don't know**: contingent

Wumpus models

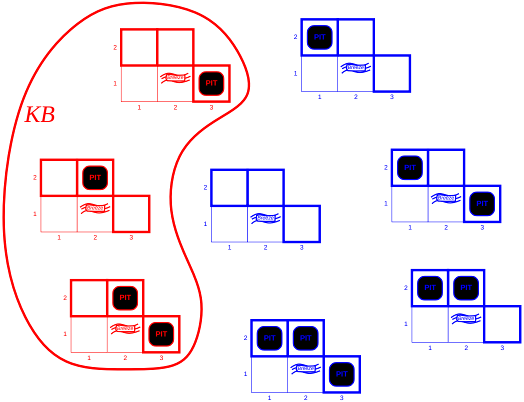
- Situation after the agent detecting nothing in [1,1], moving right and feel breeze in [2,1]
- Consider possible models? (assuming only pits)
  - 3 Boolean choices $\Longrightarrow$ 8 possible models





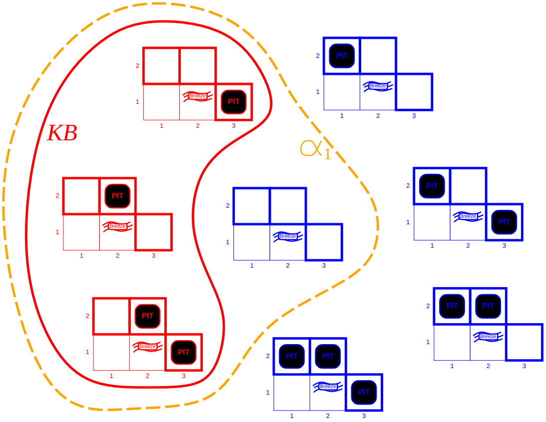
Knowledge base

- The agent *building* **knowledge base** KB from wumpus-world rules $+$ observations



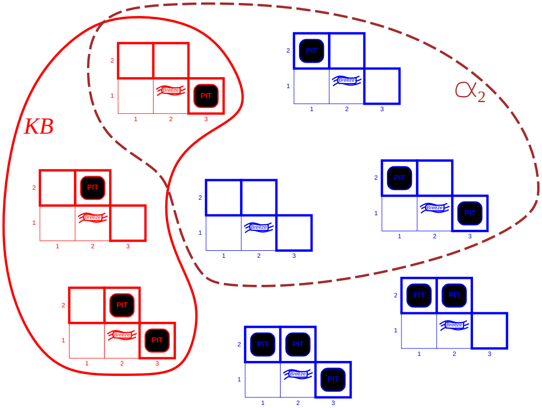
Entailment

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



Contingency

- α_2 = "[2,2] is safe", $KB \not\models \alpha_2$



Propositional Logic

Syntax

Propositional logic (a formal language) is the simplest logic – illustrates basic ideas.

The **syntax** of propositional logic defines

- **Constants**:

$$True,\ False$$

- **Symbols**: stand for propositions

$$A,\ B,\ B_{1,1},\ P_{2,1}$$

- **Logical connectives (operator)**

connectives	**meaning**	**example**
$\neg$	negation (NOT)	$\neg S$
$\wedge$	conjunction (AND)	$S_1 \wedge S_2$
$\vee$	disjunction (OR)	$S_1 \vee S_2$
$\implies$	implication	$S_1 \implies S_2$
$\iff$	equivalence, biconditional	$S_1 \iff S_2$

Syntax (cont.)

- A BNF (Backus–Naur Form) grammar of sentences in propositional logic, along with operator precedences, from highest to lowest.

$$
\begin{array}{rcl}
\mathit{Sentence} & \rightarrow & \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} & \rightarrow & \mathit{True} \mid \mathit{False} \mid P \mid Q \mid R \mid \ldots \\
\mathit{ComplexSentence} & \rightarrow & (\mathit{Sentence}) \mid [\mathit{Sentence}] \\
 & \mid & \neg \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \land \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \lor \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \implies \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \iff \mathit{Sentence} \\
 & & \\
\textsc{Operator Precedence} & : & \neg, \land, \lor, \implies, \iff
\end{array}
$$

Semantic

- The semantics defines the rules for determining the truth of a sentence with respect to a particular model m
  - Each model m specifies *true*/*false* for each proposition symbol
  - Arbitrary sentence can be evaluateed by **recursive process** PL-TRUE and **truth tables**

Table 1: Truth tables for the five logical connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \implies Q$	$P \iff Q$
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*

Semantic (cont.)

```
function PL-TRUE?(α,model) returns true or false
  if α is a symbol then return LOOKUP(α, model)
  if OP(α) = ¬ then return NOT(PL-TRUE?(ARG1(α),model))
  if OP(α) = ∧ then return AND(PL-TRUE?(ARG1(α),model),
                                PL-TRUE?(ARG2(α),model))
  if OP(α) = ∨ then return OR(PL_TRUE?(ARG1(α),model),
                               PL-TRUE?(ARG2(α),model))
  if OP(α) = ⟹ then return ...
  if OP(α) = ⟺ then return ...
```

Entailment

Problem

Given a set of sentences KB and α. Prove that

$$KB \models \alpha$$

Method 1: model-checking

- Time complexity: $O(2^n)$ (if KB and α contain n symbols $\rightarrow$ there are 2^n models)
- Space complexity: $O(n)$ (depth-first)
- OK for propositional logic; not easy for first-order logic

Method 2: theorem-proving

- Search for a sequence of proof steps (applications of inference rules)

Model checking

```
function TT-ENTAILS?(KB, α) returns true or false
  inputs: KB, the knowledge base, a sentence in propositional logic
          α, the query, a sentence in propositional logic
  symbols ← a list of the propositional 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
    P ← First(symbols)
    rest ← Rest(symbols)
    return (TT-CHECK-ALL(KB, α, rest, model ∪ {P = true})
            and
            TT-CHECK-ALL(KB, α, rest, model ∪ {P = false}))
```

Validity

Concept 7

A sentence is **valid** if it is true in *all* models. Valid sentences are also known as **tautologies**

Theorem 2 (Deduction theorem)

For any sentences α *and* β, $\alpha \models \beta$ *if and only if the sentence* $(\alpha \implies \beta)$ *is valid.*

Satisfiability

Concept 8

A sentence is **satisfiable** if it is true in, or satisfied by, *some* model

The SAT problem

The problem of determining the satisfiability of sentences in propositional logic was the first problem proved to be NP-complete

Validity, satisfiability and entailment

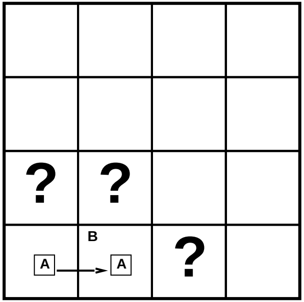
Given two sentences α, β

- α is valid iff $\neg\alpha$ is unsatisfiable
- α is satisfiable iff $\neg\alpha$ is not valid
- $\alpha \models \beta$ iff the sentence $(\alpha \wedge \neg\beta)$ is unsatisfiable (**refutation** or **contradiction**)

A simple knowledge base in Wumpus world

Symbols for each $[x, y]$ location:

- $P_{x,y}$ is true if there is a pit in $[x, y]$.
- $W_{x,y}$ is true if there is a wumpus in $[x, y]$, dead or alive.
- $B_{x,y}$ is true if the agent perceives a breeze in $[x, y]$.
- $S_{x,y}$ is true if the agent perceives a stench in $[x, y]$.



Sentences in Wumpus world

$$
\begin{aligned}
s_1 &: \neg P_{1,1} \\
s_2 &: B_{1,1} \iff (P_{1,2} \lor P_{2,1}) \\
s_3 &: B_{2,1} \iff (P_{1,1} \lor P_{2,2} \lor P_{3,1}) \\
s_4 &: \neg B_{1,1} \\
s_5 &: B_{2,1}
\end{aligned}
$$

Inference in Wumpus world

- A truth table constructed for the knowledge base given in the text. KB is true if s_1 through s_5 are true, which occurs in just 3 of the 128 rows

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	s_1	s_2	s_3	s_4	s_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

- The agent makes some conclusion
  - $KB \models \neg P_{1,2}$ means there is no pit in [1,2]
  - $KB \not\models \neg P_{2,2}$ means there might (or might not) be a pit in [2,2]

Propositional Inference

- Conjunctive Normal Form
- Horn Form

Inference framework

Concept 9

If $\ell_1, \cdots, \ell_k, m$ are sentences, then the following is an **inference rule**:

$$\frac{\ell_1, \cdots, \ell_k}{m} \quad \frac{\text{(premises)}}{\text{(conclusion)}} \tag{4}$$

- **Note**: Rules operate directly on **syntax**, not on **semantics**.

Forward inference

Input: set of inference rules Rs.
Repeat until no changes to KB:
Choose set of formulas $\ell_1, \cdots, \ell_k \in KB$.
If matching rule $\frac{\ell_1, \cdots, \ell_k}{m}$ exists **then** add m to KB.

Concept 10

KB **derives**/**proves** a sentence α, denoted by

$$KB \vdash_{Rs} \alpha \tag{5}$$

iff α eventually gets added to KB.

Soundness

Concept 11

A set of inference rules Rs is **sound** if whenever $KB \vdash_{Rs} \alpha$, it is also true that $KB \models \alpha$ or

$$\{\alpha \mid KB \vdash_{Rs} \alpha\} \subseteq \{\alpha \mid KB \models \alpha\} \tag{6}$$

Completeness

Concept 12

A set of inference rules Rs is **complete** if whenever $KB \models \alpha$, it is also true that $KB \vdash_{Rs} \alpha$

$$\{\alpha \mid KB \models \alpha\} \subseteq \{\alpha \mid KB \vdash_{Rs} \alpha\} \tag{7}$$

World and representation

- if KB is true in the real world, then any sentence α derived from KB by a sound inference procedure is also true in the real world

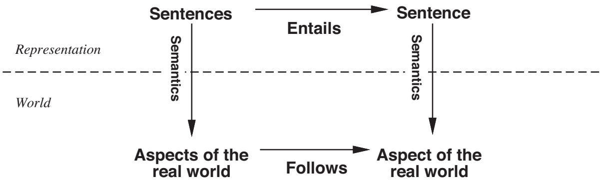


Figure 3: Sentences are physical configurations of the agent, and reasoning is a process of constructing new physical configurations from old ones. Logical reasoning should ensure that the new configurations represent aspects of the world that actually follow from the aspects that the old configurations represent.

Logical equivalence

Concept 13

Two sentences α and β are logically **equivalent** if they are true in the same set of models. We denotes as $\alpha \equiv \beta$

$$\alpha \equiv \beta \text{ if and only if } \alpha \models \beta \text{ and } \beta \models \alpha$$

Logical equivalence (cont.)

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) \text{ commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) \text{ commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) \text{ associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) \text{ associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha \text{ double-negation elimination} \\
(\alpha \implies \beta) &\equiv (\neg\beta \implies \neg\alpha) \text{ contraposition} \\
(\alpha \implies \beta) &\equiv (\neg\alpha \vee \beta) \text{ implication elimination} \\
(\alpha \iff \beta) &\equiv ((\alpha \implies \beta) \wedge (\beta \implies \alpha)) \text{ biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) \text{ De Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) \text{ De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \text{ distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \text{ distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Inference rule approach

- **Theorem proving**: Apply rules of inference directly to the sentences in *KB* to construct a proof of the desired sentence without consulting models.
  - More efficient than model checking when the number of models is large but the length of the proof is short
  - Legitimate (sound) generation of new sentences from old
- **Proof** = a sequence of inference rule applications to the desired goal
  - Can use inference rules as operators in a **standard search algorithm**. Typically require translation of sentences into a **normal form**
- **Note**: Logical systems is **monotonicity**, which says that the set of entailed sentences can only *increase* as information is added to the knowledge base. For any sentences α and β,

$$\textbf{\textit{if}}\ KB \models \alpha\ \textbf{\textit{then}}\ KB \wedge \beta \models \alpha$$

Important inference rules

Modus ponens $$\frac{\alpha \implies \beta, \quad \alpha}{\beta}$$

Modus tollens $$\frac{\alpha \implies \beta, \quad \neg\beta}{\neg\alpha}$$

And-introduction $$\frac{\alpha, \quad \beta}{\alpha \land \beta}$$

And-elimination $$\frac{\alpha \land \beta}{\alpha}$$

Example 1

Problem

Given $KB = \{P \wedge Q, P \implies R, Q \wedge R \implies S\}$, prove that $KB \models S$

Solution

#	Sentence	Explanation
s_1	$P \wedge Q$	from KB
s_2	$P \implies R$	from KB
s_3	$Q \wedge R \implies S$	from KB
s_4	P	(1) and-elimination
s_5	R	(4,2) modus ponens
s_6	Q	(1) and-elimination
s_7	$Q \wedge R$	(5,6) and-introduction
s_8	S	(3,7) modus ponens

■

Example 2

Problem

In Wumpus wolrd, given $KB = \{s_1, s_2, s_3, s_4, s_5\}$, prove that $KB \models \neg P_{1,2}$

Solution

#	Sentence	Explanation
s_1	$\neg P_{1,1}$	from KB
s_2	$B_{1,1} \iff (P_{1,2} \lor P_{2,1})$	from KB
s_3	$B_{2,1} \iff (P_{1,1} \lor P_{2,2} \lor P_{3,1})$	from KB
s_4	$\neg B_{1,1}$	from KB
s_5	$B_{2,1}$	from KB
s_6	$(B_{1,1} \implies (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \implies B_{1,1})$	Bi-conditional elimination to s_2
s_7	$(P_{1,2} \lor P_{2,1}) \implies B_{1,1}$	And-elimination to s_6
s_8	$\neg B_{1,1} \implies \neg(P_{1,2} \lor P_{2,1})$	Contrapositives to s_7
s_9	$\neg(P_{1,2} \lor P_{2,1})$	Modus ponens to s_4, s_8
s_{10}	$\neg P_{1,2} \land \neg P_{2,1}$	De Morgan's rule to s_9
s_{11}	$\neg P_{1,2}$	And-elimination to s_{10}

■

Proving by search

Any search algorithms can be applied to find a sequence of steps that constitutes a proof:

- **INITIAL STATE**: the initial knowledge base KB.
- **ACTIONS**: the set of actions consists of all the inference rules applied to all the sentences that match the top half of the inference rule.
- **RESULT**: the result of an action is to add the sentence in the bottom half of the inference rule.
- **GOAL**: the goal is a state that contains the sentence we are trying to prove.

Conjunctive Normal Form

Concept 14

Conjunctive Normal Form (CNF—universal)

$$\textit{conjunction of} \quad \underbrace{\textit{disjunctions of literals}}_{\textit{clauses}}$$

A BNF (Backus–Naur Form) grammar for conjunctive normal form

$$\begin{aligned}
CNFSentence &\rightarrow Clause_1 \wedge \ldots \wedge Clause_n \\
Clause &\rightarrow Literal_1 \vee \ldots \vee Literal_m \\
Literal &\rightarrow Symbol \mid \neg Symbol \\
Symbol &\rightarrow P \mid Q \mid R \ldots
\end{aligned}$$

Conversion to CNF

Given a sentence $B_{1,1} \iff (P_{1,2} \vee P_{2,1})$

1. Eliminate $\iff$, replacing $\alpha \iff \beta$ with $(\alpha \implies \beta) \wedge (\beta \implies \alpha)$.

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

2. Eliminate $\implies$, replacing $\alpha \implies \beta$ 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:

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

Resolution inference rule

Resolution inference rule (for CNF):

$$\frac{\ell_1 \vee \cdots \vee \ell_i \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_j \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n} \tag{8}$$

where ℓ_i and m_j are complementary literals

Theorem 3

Resolution inference rule is sound and complete for CNF KB

The resolution algorithm

- Proof by contradiction: To show that $KB \models \alpha$, prove that $KB \wedge \neg\alpha$ is unsatisfiable

```
function PL-RESOLUTION(KB, α) returns true or false
inputs: KB, the knowledge base, a sentence in propositional logic
        α, the query, a sentence in propositional logic
  clauses ← the set of CNF clauses of KB ∧ ¬α
  new ← ∅
  loop do
    for each pair of clauses 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
```

Example 3

- $KB = \{(B_{1,1} \iff (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}\}$ and $\alpha = \neg P_{1,2}$
- **Note**: many resolution steps are pointless.

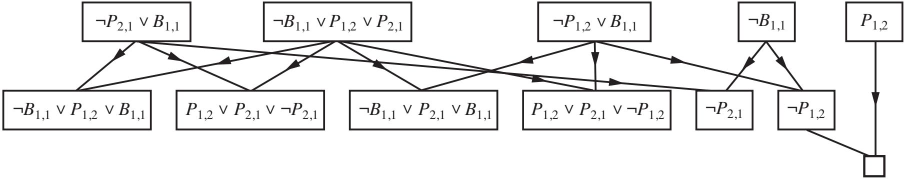


Figure 4: Partial application of PL-RESOLUTION to a simple inference in the wumpus world. $\neg P_{1,2}$ is shown to follow from the first four clauses in the top row

Horn Form

- In many practical situations, the full power of resolution is not needed. Some real-world knowledge bases satisfy certain restrictions (Horn form) on the form of sentences

Concept 15

Horn Form (restricted)

conjunction of *Horn clauses*

A BNF (Backus–Naur Form) grammar for Horn form

$$
\begin{aligned}
\textit{HornClauseForm} &\rightarrow \textit{DefiniteClauseForm} \mid \textit{Symbol} \\
\textit{DefiniteClauseForm} &\rightarrow (\textit{Symbol}_1 \wedge \ldots \wedge \textit{Symbol}_n) \implies \textit{Symbol} \\
\textit{Symbol} &\rightarrow P \mid Q \mid R \ldots
\end{aligned}
$$

Modus ponens inference rule

- **Modus ponens inference rule** (for Horn Form)

$$\frac{\alpha_1, \ldots, \alpha_n, \alpha_1 \wedge \cdots \wedge \alpha_n \implies \beta}{\beta} \tag{9}$$

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

Theorem 4

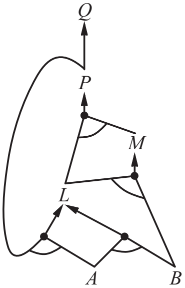
Modus ponens inference rule is sound and complete for Horn KB

Forward chaining (FC)

- **Idea**: fire any rule whose *premises* are satisfied in the KB, add its *conclusion* to the KB, until query is found
- **Example**: Given the following KB, prove that $KB \models Q$

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

Figure 5: The corresponding AND–OR graph.



The forward-chaining algorithm

```
function PL-FC-ENTAILS?(KB, q) returns true or false
inputs: KB, the knowledge base, a set of propositional definite clauses
        q, the query, a proposition symbol
  count ← a table, where count[c] is the number of symbols in c's premise
  inferred ← a table, where inferred[s] is initially false for all symbols
  agenda ← a queue of symbols, initially symbols known to be true in KB
  while agenda ≠ ∅ do
    p ← POP(agenda)
    if p = q then return true
    if inferred[p] = false then
      inferred[p] ← true
      for each clause c in KB where p is in c.PREMISE do
        decrement count[c]
        if count[c] = 0 then add c.CONCLUSION to agenda
  return false
```

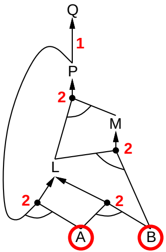
Proof of completeness

FC derives every atomic sentence that is entailed by KB

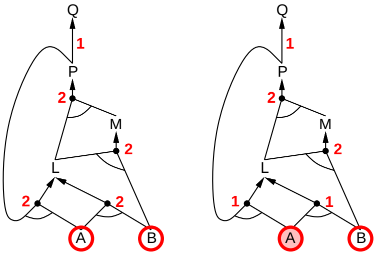
1. FC reaches a **fixed point** where no new atomic sentences are derived
2. Consider the final state as a model m, assigning true/false to symbols
3. Every clause in the original KB is true in m
   Proof: Suppose a clause $a_1 \wedge \ldots \wedge a_k \Rightarrow b$ is false in m
   - Then $a_1 \wedge \ldots \wedge a_k$ is true in m and b is false in m
   - Therefore the algorithm has not reached a fixed point!
4. Hence m is a model of KB
5. If $KB \models q$, q is true in *every* model of KB, including m

- **General idea**: construct any model of KB by sound inference, check α

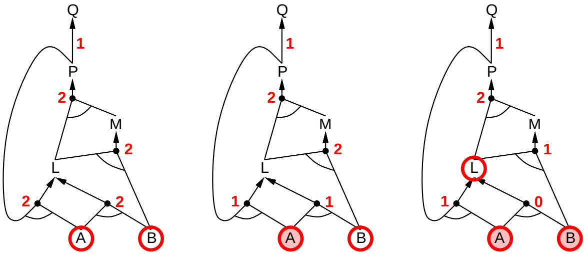
Forward chaining example



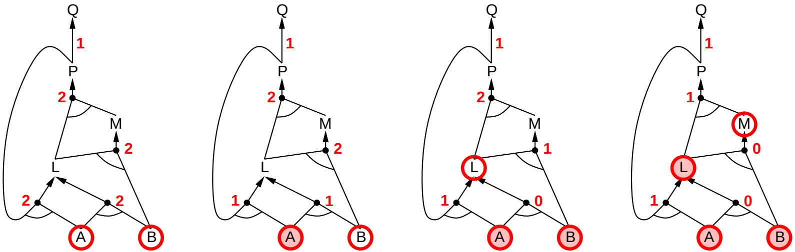
Forward chaining example



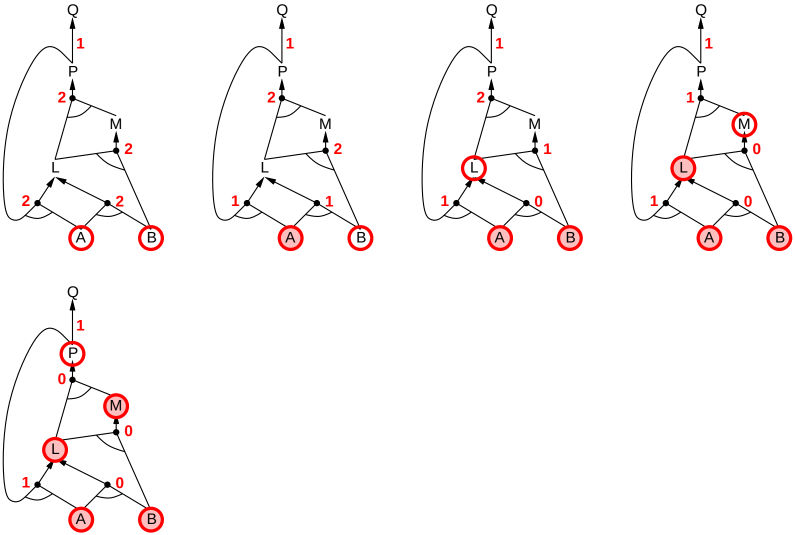
Forward chaining example



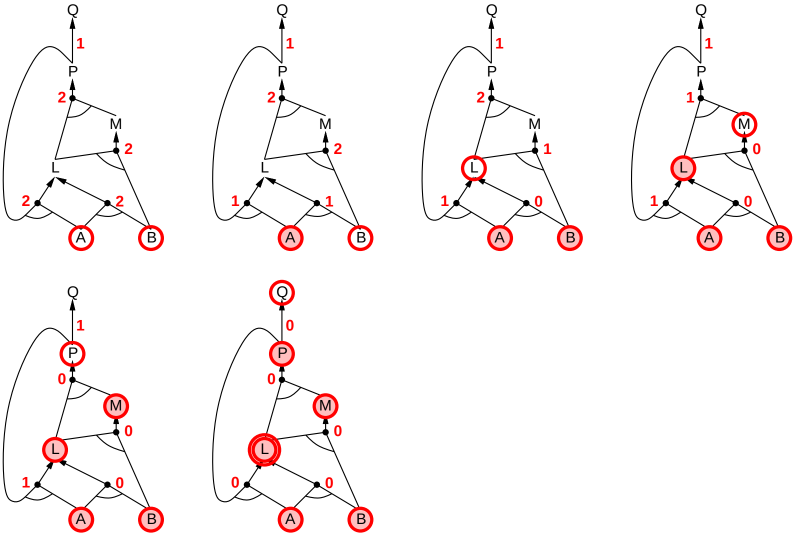
Forward chaining example



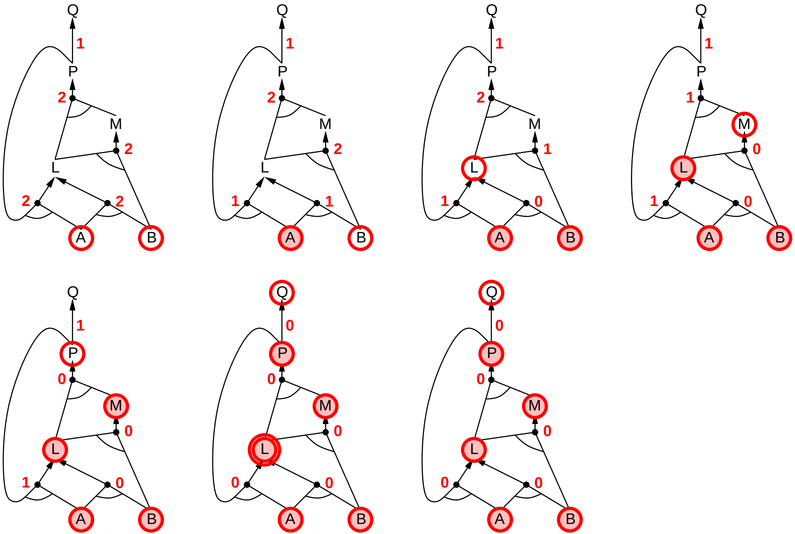
Forward chaining example



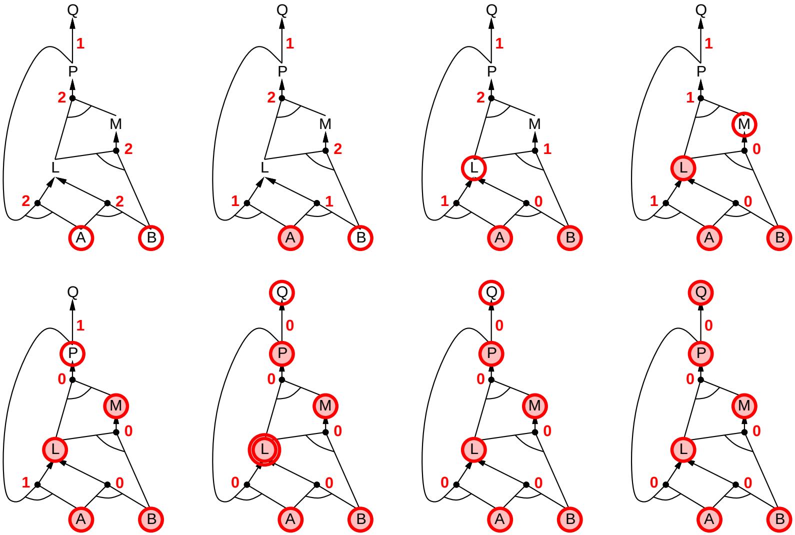
Forward chaining example



Forward chaining example



Forward chaining example

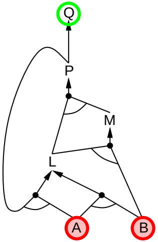


Backward chaining (BC)

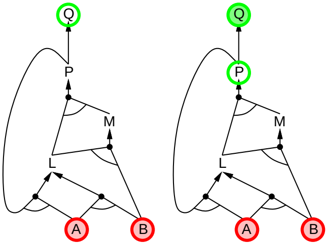
Idea: work backwards from the query q:

- 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

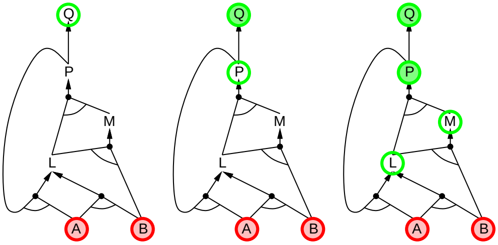
Backward chaining example



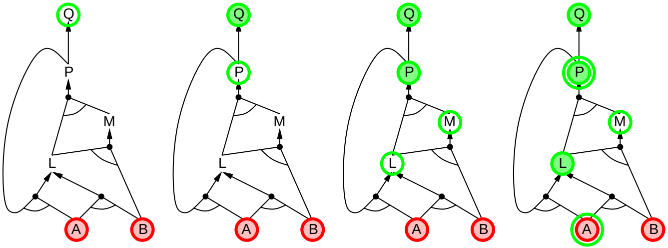
Backward chaining example



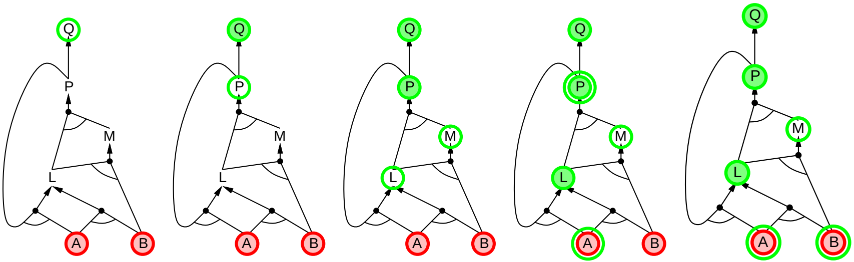
Backward chaining example



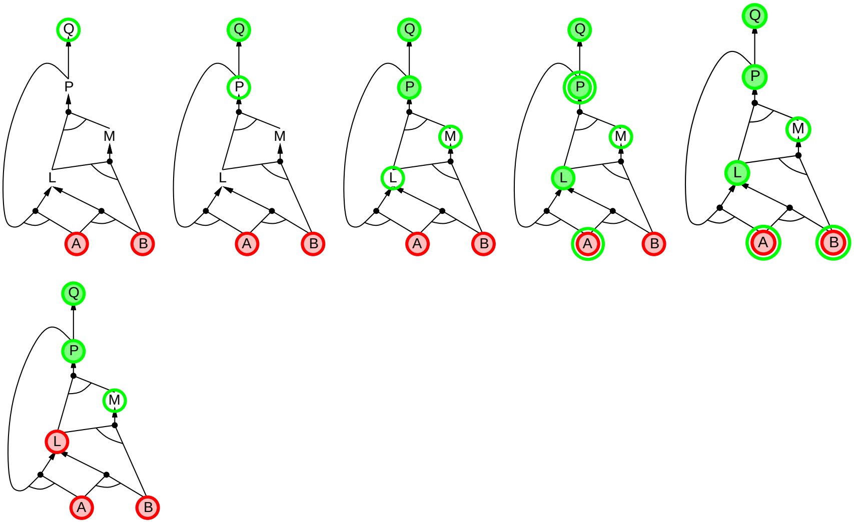
Backward chaining example



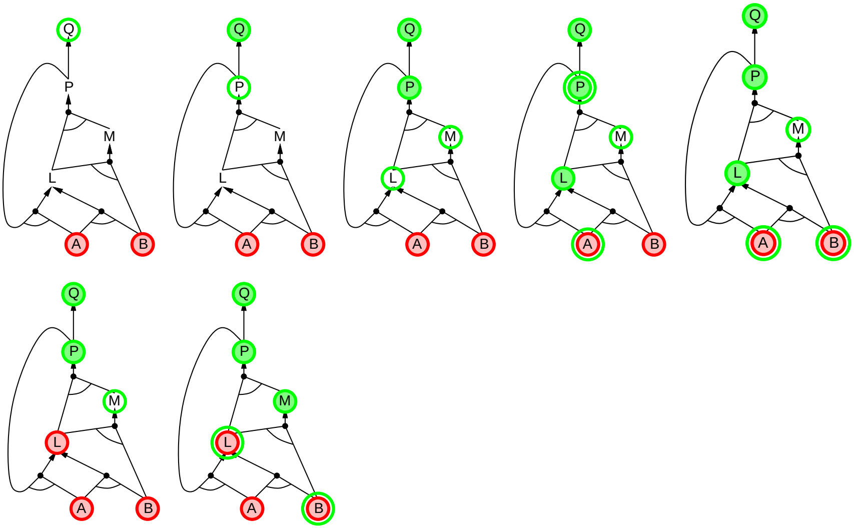
Backward chaining example



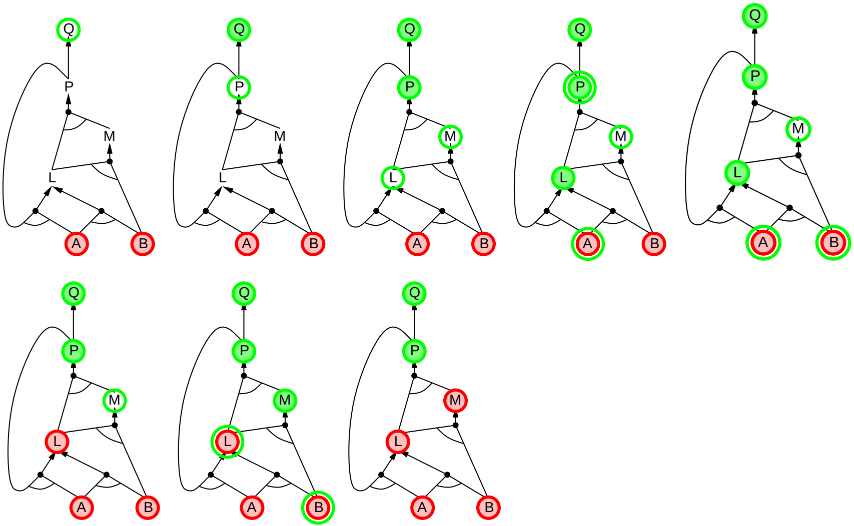
Backward chaining example



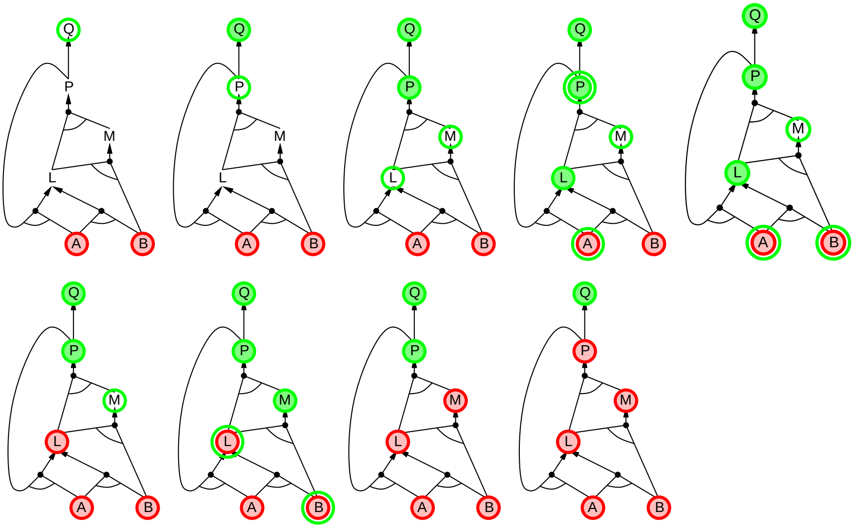
Backward chaining example



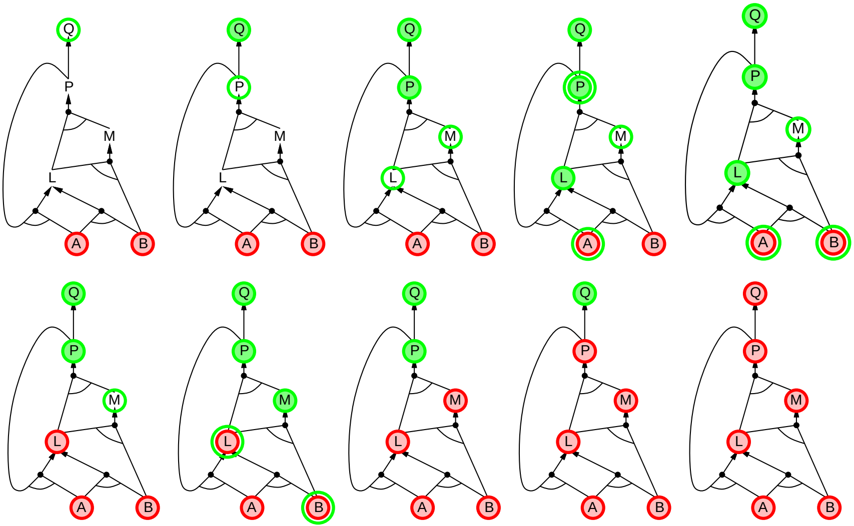
Backward chaining example



Backward chaining example



Backward chaining example



Forward vs. backward chaining

- FC is **data-driven**, cf. 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 Model Checking

- DPLL
- WalkSAT

Efficient propositional inference

Problem

The SAT problem is checking satisfiability of sentence α

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

1. Complete backtracking search algorithms
   - DPLL algorithm (proposed by Davis, Putnam, Logemann and Loveland)
2. Incomplete local search algorithms (hill-climbing)
   - WalkSAT algorithm

- Application of SAT: testing entailment, $\alpha \models \beta$, can be done by testing **unsatisfiability** of $\alpha \wedge \neg\beta$.

The DPLL algorithm

- Determine whether an input propositional logic sentence (**in CNF**) is satisfiable
- A recursive, depth-first enumeration of possible models.
- Improvements over truth table enumeration
  1. Early termination
  2. Pure symbol heuristic
  3. Unit clause heuristic

The DPLL algorithm (cont.)

- **Early termination**
  - A clause is true if any literal is true.
  - A sentence is false if any clause is false.
  - Avoid examination of entire subtrees in the search space
  - E.g., $(B \vee C) \wedge (B \vee D)$ is true if B is true, regardless C and D

The DPLL algorithm (cont.)

- **Pure symbol heuristic**
  - **Pure symbol**: always appears with the same "sign" in all clauses.
  - Make a pure symbol literal true $\rightarrow$ can never make a clause false
  - For example, given a sentence $(A \vee \neg B), (\neg B \vee \neg C), (A \vee C)$ $\rightarrow$ the symbols A and B are pure, C is impure.
- **Unit clause heuristic**
  - **Unit clause**: only one literal in the clause $\rightarrow$ the only literal in a unit clause must be **true** $\rightarrow$ cause "cascade" of forced assignments (**unit propagation**)
  - For example, given a sentence $B, \neg B \vee \neg C$, if the model contains $B = \mathit{true}$ then $C = \mathit{false}$

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 ≠ ∅ then
    return DPLL(clauses, symbols − P, model ∪ {P = value})
  P, value ← FIND-UNIT-CLAUSE(clauses, model)
  if P ≠ ∅ then
    return DPLL(clauses, symbols − P, model ∪ {P = value})
  P ← FIRST(symbols); rest ← REST(symbols)
  return DPLL(clauses, rest, model ∪ {P = true}) or
         DPLL(clauses, rest, model ∪ {P = false})
```

Success of DPLL

- 1962 – DPLL invented
- 1992 – 300 propositions
- 1997 – 600 propositions (satz)
- 2002 (zChaff) 1,000,000 propositions – encodings of hardware verification problems

The WalkSAT algorithm

- **Evaluation function**: The min-conflict heuristic of minimizing the number of unsatisfied clauses
- Balance between **greediness** and **randomness**
- When the algorithm returns a model
  - The input sentence is indeed satisfiable
- When it returns failure
  - The sentence is unsatisfiable OR we need to give it more time
- WALKSAT cannot always detect unsatisfiability
- It is most useful when a solution is expected to exist. For example,
  - An agent cannot reliably use WALKSAT to prove that a square is safe in the Wumpus world.
  - Instead, it can say, "I thought about it for an hour and couldn't come up with a possible world in which the square isn't safe."

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,
           typically around 0.5
        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
```

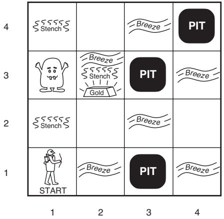
Propositional Logic Based Agent

Propositional Logic Based Agent

- Agent has to act given only local perception
- Agent is installed with two kinds of knowledge base
  - "Hardcode" knowledge base

    IF *glitter* ***THEN*** *grab gold*

    IF *wumpus or pit around* ***THEN*** *avoid it*
  - "Softcode" knowledge base *KB*



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

For a 4×4 wumpus world, the *KB* begin with a total of 155 sentences containing 64 distinct symbols

Algorithm

```
function PL-WUMPUS-AGENT(percept) returns an action
inputs: percept, a list, [stench,breeze,glitter]
static: KB, a knowledge base
        x, y, , the agent's position (initially 1,1)
        orientation, orientation (initially right)
        visited, an array indicating which squares have been visited,
                initially false
        action, the agent's most recent action, initially null
        plan, an action sequence, initially empty
  update x,y,orientation, visited based on action
  if stench then TELL(KB,S_{x,y}) else TELL(KB,¬S_{x,y})
  if breeze then TELL(KB,B_{x,y}) else TELL(KB,¬B_{x,y})
  if glitter then action ← grab
  else if plan ≠ ∅ then action ← POP(plan)
  else if for some fringe square [i,j], ASK(KB,(¬P_{i,j} ∧ ¬W_{i,j})) is true or
          for some fringe square [i,j], ASK(KB,(P_{i,j} ∨ W_{i,j})) is false then
    plan ← A*-GRAPH-SEARCH(ROUTE-PROBLEM([x,y],orientation,[i,j],visited))
    action ← POP(plan)
  else action ← a randomly chosen move
return action
```

References

Goodfellow, I., Bengio, Y., and Courville, A. (2016).
Deep learning.
MIT press.

Lê, B. and Tô, V. (2014).
Cở sở trí tuệ nhân tạo.
Nhà xuất bản Khoa học và Kỹ thuật.

Nguyen, T. (2018).
Artificial intelligence slides.
Technical report, HCMC University of Sciences.

Russell, S. and Norvig, P. (2016).
Artificial intelligence: a modern approach.
Pearson Education Limited.