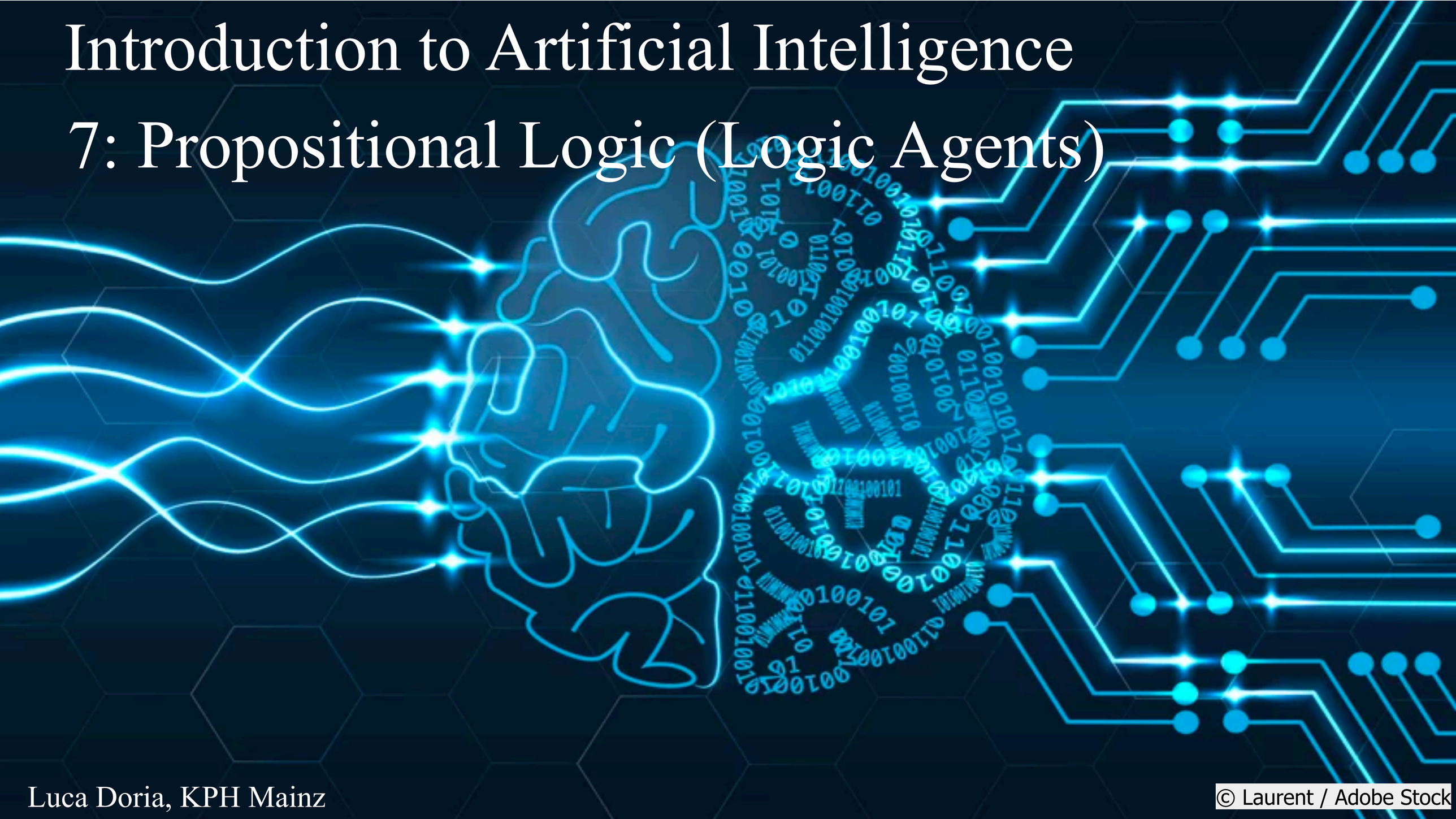
Introduction to Artificial Intelligence

7: Propositional Logic (Logic Agents)



Luca Doria, KPH Mainz

A new kind of agent: logical agents

Intelligent agents until now:

- Limited “knowledge”
- No knowledge of general facts about the problem
- Often forced to generate a large amount of states with fixed rules

Embed agents with logic:

- Ability to employ general rules disconnected to the specific data
- Ability to update their behaviour as new knowledge is acquired
- Ability to perform new tasks

Applications

Logic as universal tool for reasoning:

- Theorem proving
- Software verification
- Detection of unwanted states
- Hardware verification
- Relation to NP-hard problems

Outline:

- Agents thinking rationally
- An example “world”
- Introduction to propositional logic
- Syntax and Semantics
- Entailment and Resolution

Agents thinking rationally

- We pretended that our agents were acting rationally.
- Rationality often requires logical thought.
- For that part of the world where the agent is operating must be represented by a knowledge base (KB).

A KB is composed of sentences in a language together with a truth theory (logic).
We, externally to the agent, can attach a meaning to these sentences as statements about the world (**semantics**).
The sentences can influence the agent's behaviour through their form (**syntax**).

Interaction with the KB through the simplified actions "Ask" and "Tell":
- Ask(KB,P) = "yes" when P follows from KB
- Tell(KB,P) = KB' so that P follows from KB'

The 3 levels of a logical agent (A. Newell, 1982)

We can define 3 levels for giving an abstract description of logical agents:

1: Knowledge Level: This is the most abstract and concerns the total knowledge in the KB. For example, the Lufthansa information system knows the price P of a flight from Frankfurt to Toronto.

2: Logical Level: Encodes the knowledge of a formal language. For example:
Price(Frankfurt, Toronto, P).

3: Implementation Level: The concrete internal representation of the sentences.
For example:
Price(Frankfurt, Toronto, P) represented as a string, as a matrix, hash table,…

If Ask and Tell work correctly, it is possible to remain at Level 1.
the advantage is a simpler user interface.

Knowledge-Based Agents

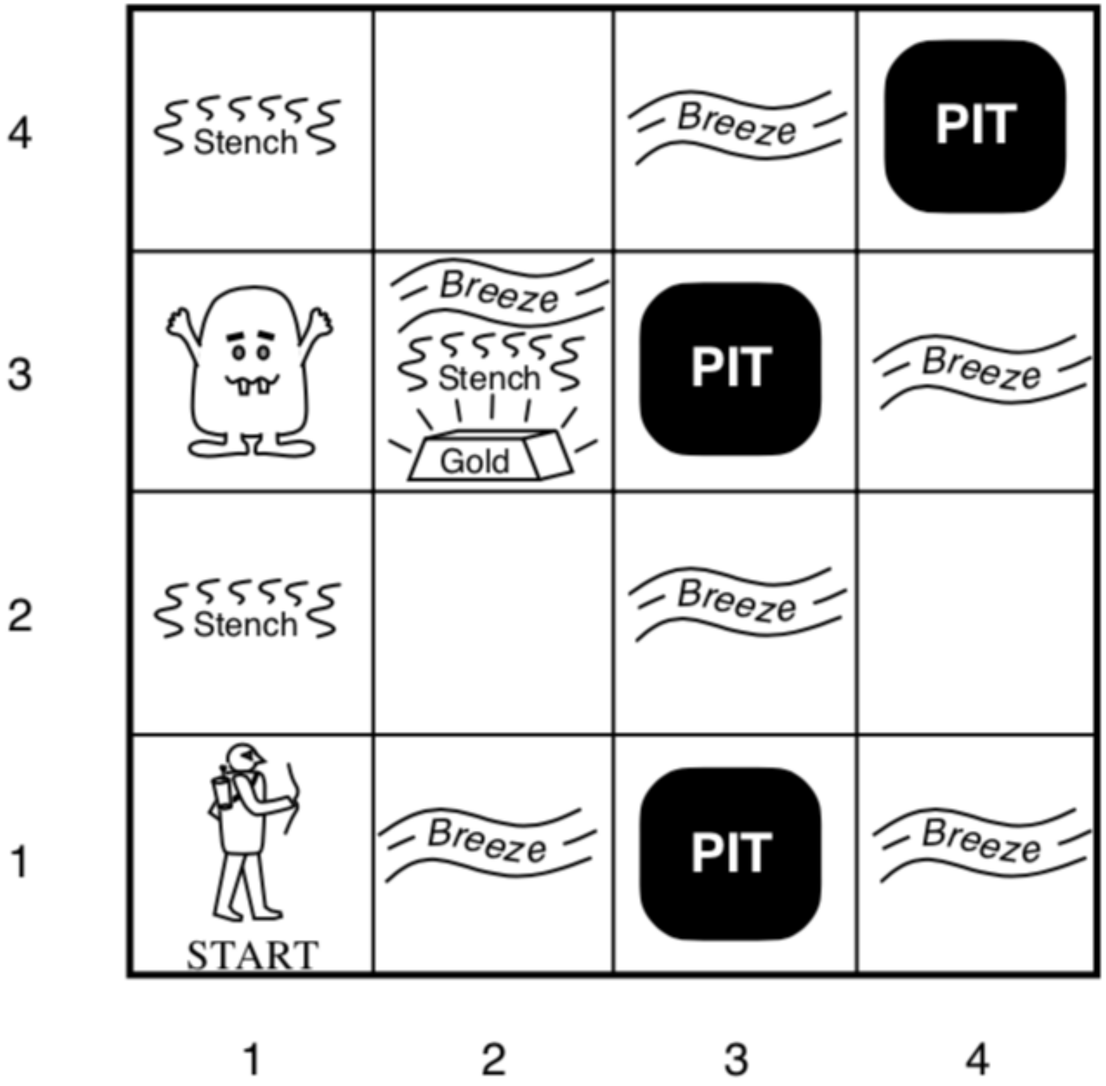
```
function KB-AGENT(percept) returns an action
    persistent: KB, t

    TELL(KB,MAKE-PERCEPT-SENTENCE(percept,t)) //add percept to KB
    action <— ASK(KB,MAKE-ACTION-QUERY(t)) //ask the KB which action to perform
    TELL(KB,MAKE-ACTION-SENTENCE(action,t)) //inform KB that the action
                                                was performed
    t <— t+1
    return action
```

An example world: the “Wumpus World” (G. Yob, 1975)

Description of the world:

- 4x4 grid
- In the Wumpus square and in the adjacent ones the agent perceives a stench.
- In the squares adjacent to a pit, the agent perceives a breeze.
- If the agent hits a wall (bump), he can perceive it.
- When the Wumpus is killed, the scream is heard everywhere.
- Percepts are the 5-tuple. [Stench, Breeze, Glitter, Bump, Scream].

- The agent cannot perceive its own location or look into an adjacent square.
- Only up-down-left-right moves are allowed, plus 90deg turns.
- Somewhere there is a gold treasure: the task is to find it.
- Initial state: agent in square [1,1].
- You have 1 (and only 1) arrow to throw



Let’s start and apply some logic…

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2	3,2	4,2
1,1 A OK	2,1 OK	3,1	4,1

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

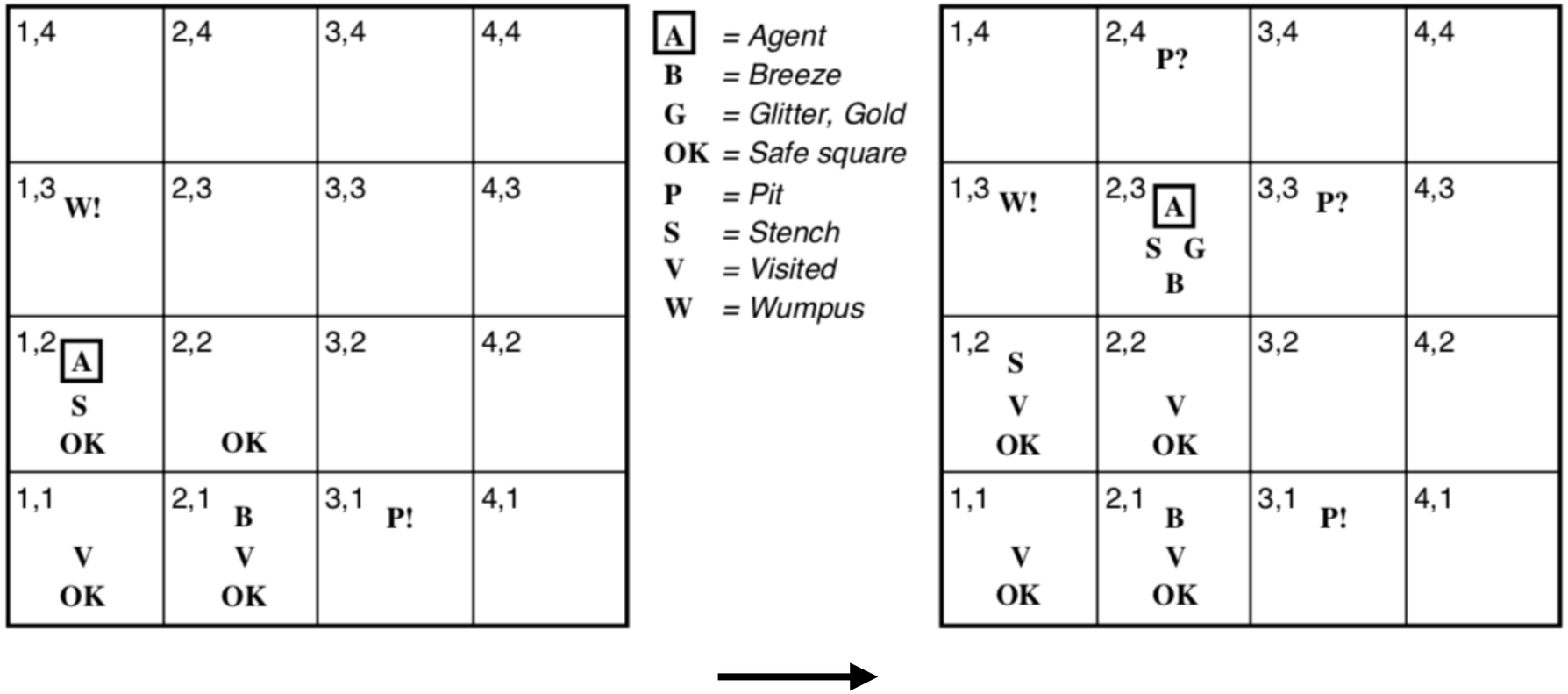
1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2 P?	3,2	4,2
1,1 V OK	2,1 A B OK	3,1 P?	4,1

Move right

Percept = [none, none, none, none, none]

Percept = [none, breeze, none, none, none]

After some steps…



Percept(1,2) = [stench, none, none, none, none]

Percept(2,3) = [stench, breeze, glitter, none, none]

We deduce that the Wumpus must be in (1,3)!

Syntax and Semantics

- The KB consists of sentences.
- Sentences are expressed in a language with a syntax
    - Syntax identifies all the sentences which are well-formed
    - Example: arithmetic.

      x+y=4 is well formed, while y4x=+ is not
- A logic defines also the semantics (the meaning) of sentences
    - Defines the truth of sentences with respect to each possible world
    - Example: semantics specifies that x+y=4 is true in a world where x=1 and y=3

Logical Entailment

- If a sentence P is true in a possible world M, we say that M satisfies P or M is a **model** of P.
- The set of all models of P is denoted by M(P)
- Logical entailment:
  - When does a sentence Q follow from another sentence P? Or: **P ⊨ Q** ?
  - P ⊨ Q if and only if (**iff**) in every model where P is true, Q is also true.
  - In symbols: **P ⊨ Q iff M(P) ⊆ M(Q)**
  - P is a stronger assertion than Q, since it rules out more worlds (models).
  - Example from arithmetic:

    x=0 entails xy=0

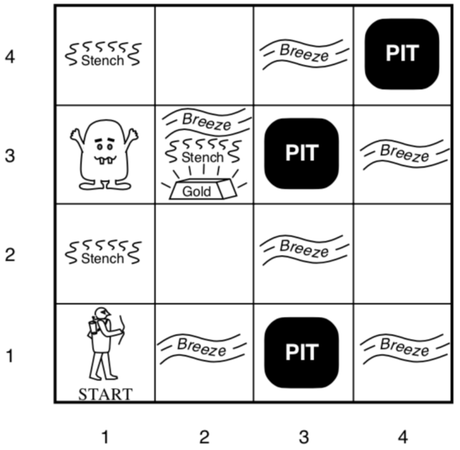
    x=0 rules out the world where {x=1,y=0} but xy=0 does not.

Example from the Wumpus World

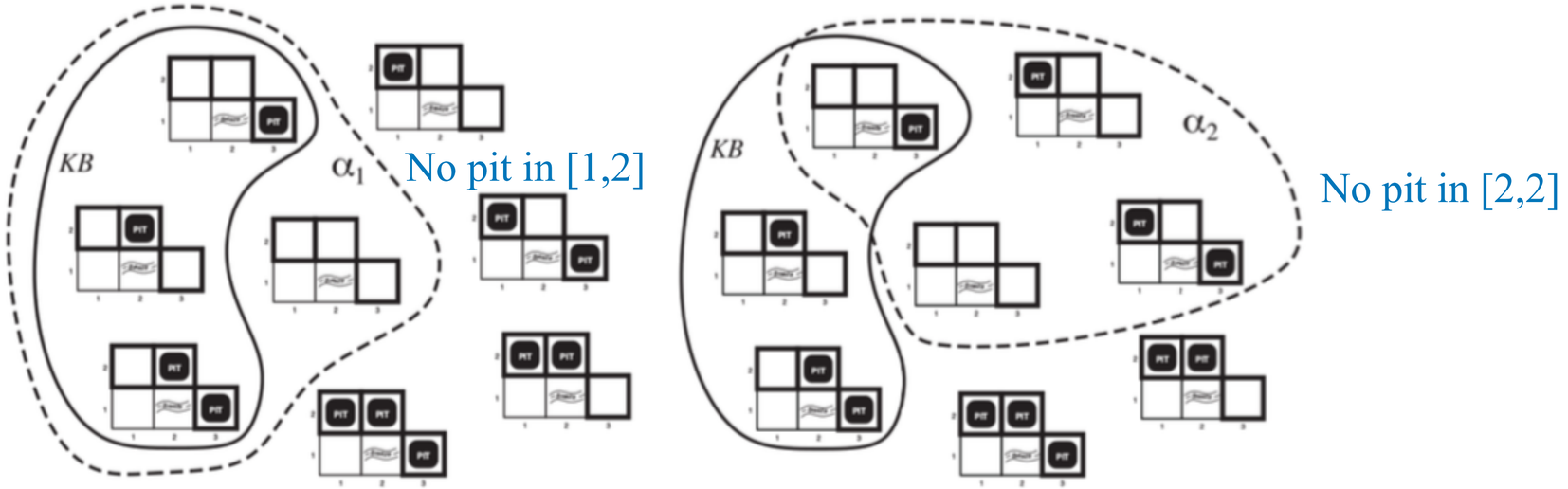
- The agent detected nothing in [1,1] and breeze in [2,1].
- This information is the actual KB, plus the rules of this world.
- Considering only [1,2] , [2,2] , [3,1] : does any of them contain a pit?
- KB returns “false” in any M contradicting what the agent knows:

  Example 1: In a world where [1,2] has a pit (no breeze in [1,1])

  Example 2: In a world where [2,2] and [3,1] do not have a pit (breeze in [2,1])



Breeze detected in [2,1]



Possible models for a pit in [1,2] , [2,2] , [3,1]

Example from the Wumpus World

- Consider the sentences:

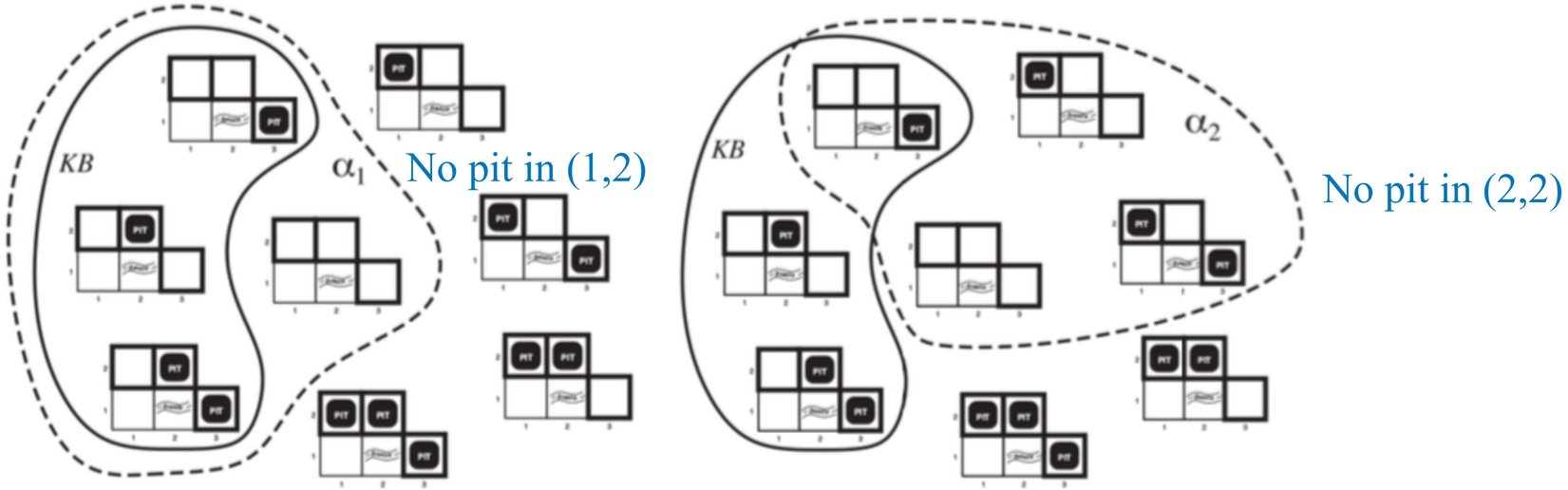
  α_1 =: “There is no pit in [1,2]” (true in the models enclosed with the dashed line, left)

  α_2 =: “There is no pit in [2,2]” (true in models enclosed with the dashed line, right)

- KB $\models \alpha_1$: by inspection, every model where KB is true, α_1 is also true

- KB $\not\models \alpha_2$: in some models where KB is true, α_2 is false

Breeze detected in (2,1)



Possible models for a pit in (1,2) , (2,2) , (3,1)

Entailment and Inference

- Which inference algorithm did we just use? Exhaustive enumeration!

  Logical entailment: KB $\models \alpha$

  Inference: derive α with an inference algorithm k.

  Formally: KB $\vdash_k \alpha$ (in words: α is derived from KB by algorithm k).

- Our aim is to have an inference algorithm that derives only sentences that are entailed (soundness) and all of them (completeness).

Soundness and Completeness

Soundness: $\vdash P \Rightarrow \models P$

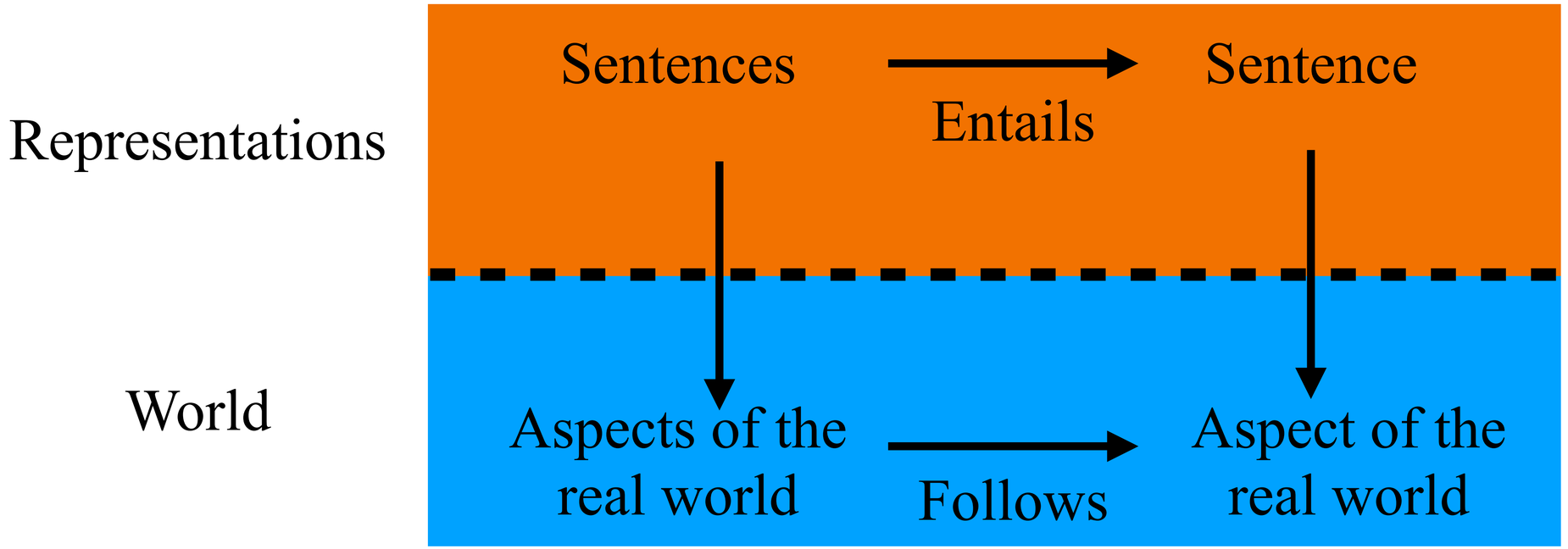
Completeness: $\models P \Rightarrow \vdash P$

1) **Soundness ensures no false positives**: If a proof exists for a statement, then the statement must be true.

2) **Completeness ensures no false negatives**: If a statement is true, then there must be a proof for it.

Ideally, in a logic system, both properties are highly desirable.
We will see how in certain cases this is not the case (Gödel's Theorems)

Entailment and Inference



- Sentences are physical configurations of the agent.
- Reasoning is producing new configurations from old ones.
- Logical reasoning ensures that new configurations respect the aspects of the following world

Declarative Languages

In order to be capable of building a system (agent) which is capable of learning, thinking, planning, … we have to find how to express knowledge.

This can be done only with a precise, declarative language:

Declarative:

We state what we want to compute, not how

The system believes P if and only if (iff) it considers P to be true

Precise:

We must know, which symbols represent sentences, what it means for a sentence to be true, and when a sentence follows from other sentences.

One possibility: **Propositional Logic**

Propositional Logic

Propositions:

The building blocks of propositional logic are indivisible, atomic statements (atomic propositions),

Example: ”The Wumpus is in [1,3]”, expressed, e.g., by the symbol ”$W_{1,3}$”

and the logical connectives ”and”, ”or”, and ”not”, which can be used to build formulae.

Propositional Logic

Interesting questions:

- When is a proposition true?

- When does a proposition follow from a KB? i.e: $KB \models P$

- Can we syntactically define a concept of derivation? i.e: $KB \vdash P$

- Can we make sure that $\models$ and $\vdash$ are equivalent?

This is a concrete implementation of **Ask**.

The Syntax of Propositional Logic

Countable alphabet Σ of atomic propositions P, Q, …

Logical formulae:

$P \in \Sigma$: atomic formula

$\bot$: falseness

$\top$: truth

$\neg P$: negation

$P \wedge Q$: conjunction

$P \vee Q$: disjunction

$P \Rightarrow Q$: implication (equiv. to: $\neg P \vee Q$)

$P \Longleftrightarrow Q$: equivalence (equiv to: $P \Rightarrow Q \wedge Q \Rightarrow P$)

Operator precedence order: $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Longleftrightarrow$ (with brackets if needed)

Atom: atomic formula (P,$\bot$,$\top$)

Literal: (negated) atomic formula

Clause: disjunction of literals (e.g.: $A \vee B \vee C$,…)

Semantics

- Atomic propositions can be true (T) or false (F).
- The truth of a formula follows from the truth of its atomic propositions (truth assignment or interpretation) and the connectives.
- Example: $(P \vee Q) \wedge R$

  If P and Q are false and R is true, what is the value of the formula?
- If P and R are true, the formula is true regardless of what Q is.

Semantics

A truth assignment I of the atoms in Σ (or a boolean interpretation I over Σ) is a function

$$I : \Sigma \rightarrow \{T, F\}$$

The **interpretation** I satisfies a **formula** ϕ($I \models \phi$) :

- $I \models \top$
- $I \not\models \bot$
- $I \models P$ iff $P^I = T$
- $I \not\models \neg\phi$ iff $I \models \phi$
- $I \models (\phi \wedge \psi)$ iff $I \models \phi$ and $I \models \psi$
- $I \models (\phi \vee \psi)$ iff $I \models \phi$ or $I \models \psi$
- $I \models (\phi \Rightarrow \psi)$ iff $I \models \phi$ then $I \models \psi$
- $I \models (\phi \Longleftrightarrow \psi)$ iff $I \models \phi$ if and only if $I \models \psi$

Note 1: if Σ contains N propositions, then there are 2^N possible interpretations.

Note 2: I(⊥)=F for whatever I

Note 3: ϕ is a **tautology** if it is always true independently from the interpretation I.

Example: A ∨¬A

I satisfies ϕ(I $\models \phi$) or is true under I, when I(ϕ)=T. I can be seen as a **possible world**.

Semantics: Truth table representation

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

Notes:

- Here we do not list the XOR ("aut") operator, with values (false, true, true, false)
- Observe that the implication is false only when P is true and Q is false.

  On the other hand, if P is false and Q true, P⇒Q is true.

  This can be translated with: "if P is false, I make no claim"…

Concrete Example

Truth assignment:

$$I : \begin{cases} P \mapsto T \\ Q \mapsto T \\ R \mapsto F \\ S \mapsto F \\ \dots \end{cases}$$

Formula: $\phi = ((P \vee Q) \Leftrightarrow (R \vee S)) \wedge (\neg(P \wedge Q) \wedge (R \wedge \neg S))$

Question: $I \vDash \phi$?

The Wumpus World translated in propositional logic

Symbols: B_{11}, B_{12}, …,S_{12}, …W_{11}, …

Meaning: B = “breeze”, B_{ij} = “breeze in square (i,j)” …

Rules:

R_1: $B_{11} \Longleftrightarrow (P_{12} \vee P_{21})$

R_2: $B_{21} \Longleftrightarrow (P_{11} \vee P_{22} \vee P_{31})$

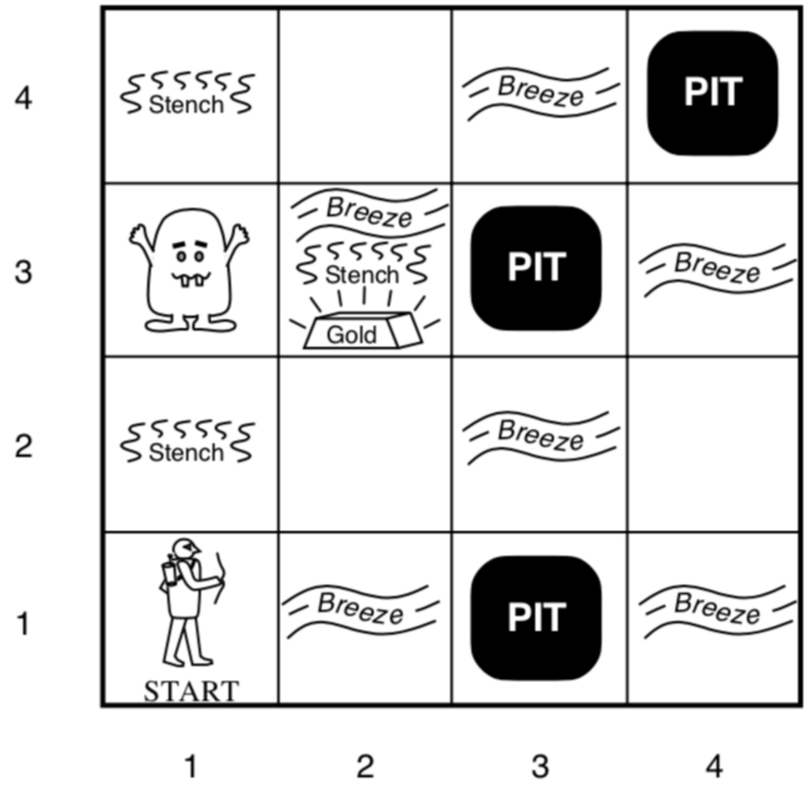
…

Facts:

F_1: $\neg P_{11}$

F_2: $\neg B_{11}$

…



Logical Entailment

An interpretation I is a **model** of ϕ if I $\models \phi$

I is a model of a set of formulae if it fulfils all the formulae of the set.

A formula ϕ is

satisfiable of there exists I that satisfies it

unsatisfiable if ϕ is not satisfiable

falsifiable if there exists I that doesn't satisfy ϕ

valid (tautology) if I $\models \phi$ holds for all I

Relation among formulae:

Two formulae are logically equivalent if (I $\models \phi$ iff I $\models \psi$) holds for all I

Logical Equivalences

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

Truth Table Method

Generating a truth table is a method for deciding if a formula is satisfiable.

Example: $\phi = ((P \vee H) \wedge \neg H) \Rightarrow P$

P	H	$P \vee H$	$(P \vee H) \wedge \neg H$	$((P \vee H) \wedge \neg H) \Rightarrow P$
F	F	F	F	T
F	T	T	F	T
T	F	T	T	T
T	T	T	F	T

The formula is true for all the possible combinations (I) therefore ϕ it is valid.

The Wumpus World Knowledge Base

Sentences:

R_1: $\neg P_{11}$ Initial observation

R_2: $B_{11} \Longleftrightarrow P_{12} \vee P_{21}$

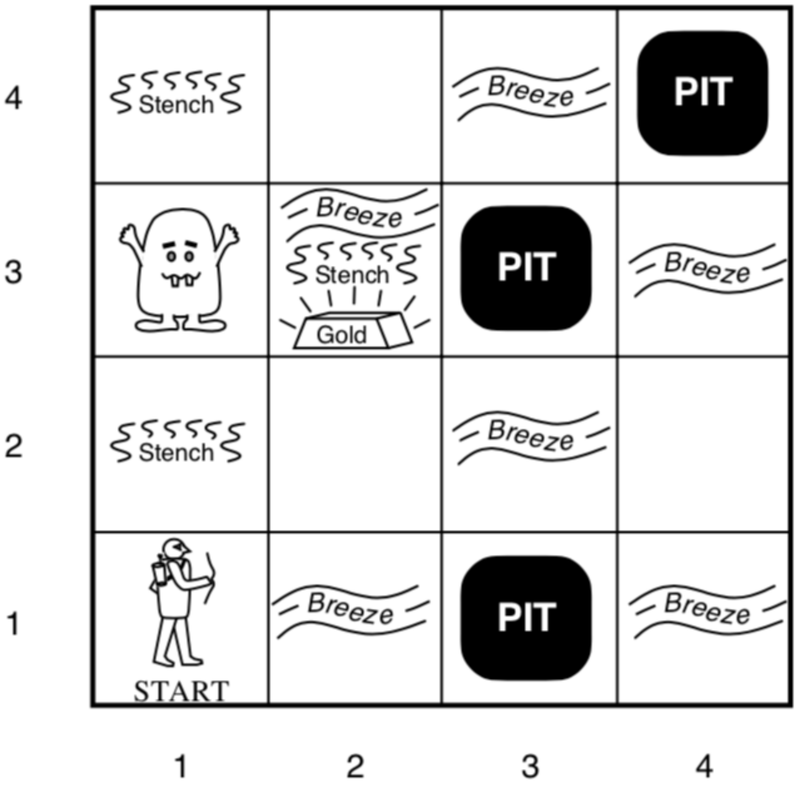
R_3: $B_{21} \Longleftrightarrow P_{11} \vee P_{22} \vee P_{31}$

} True in every world

R_4: $\neg B_{11}$

R_5: B_{21}

} Derived after first two moves



<u>They are sufficient to derive: $\neg P_{12}$</u>

Inference by Enumeration

We would like to decide whether KB $\models \phi$ for some sentence ϕ.

Example: $\phi = \neg P_{12}$.

Direct algorithm: enumerate the models and check that ϕ is true in every model where KB is true.

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
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*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
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*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Model Checking vs Theorem Proving

- **Model Checking**: Enumerate models and show the sentence holds everywhere

- **Theorem Proving**: Apply rules of inference to the sentences in the KB to prove a sentence without consulting models.

If the number of models is large with respect to the length of a proof, then theorem proving can bring a computational advantage.

Theorem Proving

Starting point: logical equivalence:

$P \equiv Q$ iff $P \models Q$ and $Q \models P$

Validity:

P is valid if true in all models (it is a tautology. Example: $P \vee \neg P$)

Deduction Theorem:

For any sentences P and Q, $P \models Q$ iff $P \Rightarrow Q$ is valid.

Satisfiability:

A sentence P is satisfiable, if it is true in some model.

In Logic (and Computer Science!) this problem is better known as **SAT**, the paradigmatic **NP-complete** problem.

Theorem Proving: Inference Rules

Modus Ponens

$$\frac{P \Rightarrow Q, P}{Q}$$

$P \Rightarrow Q, P$ → Given

Q → Inferred

Modus Tollens

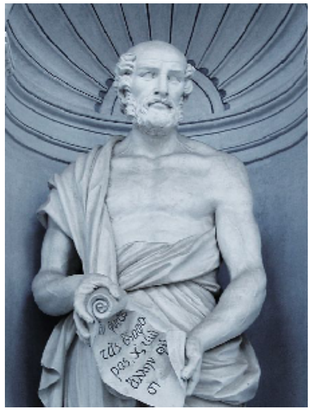
$$\frac{P \Rightarrow Q, \neg P}{\neg Q}$$

AND-elimination

$$\frac{P \wedge Q}{P}$$

Reductio ad Absurdum

$$\frac{P \Rightarrow Q, P \Rightarrow \neg Q}{\neg P}$$



Theophrastus (from Wikipedia)

Biconditional Elimination

$$\frac{P \iff Q}{(P \Rightarrow Q) \wedge (Q \Rightarrow P)}$$

$$\frac{(P \Rightarrow Q) \wedge (Q \Rightarrow P)}{P \iff Q}$$

Wumpus World Example

1: Apply biconditional elimination to R_2:

R_6: $B_{11} \Rightarrow (P_{12} \vee P_{21}) \wedge (P_{12} \vee P_{21}) \Rightarrow B_{11}$

2: Apply AND-elimination to R_6:

R_7: $(P_{12} \vee P_{21}) \Rightarrow B_{11}$

3: Logical equiv. for contrapositives:

R_8: $\neg B_{11} \Rightarrow \neg(P_{12} \vee P_{21})$

4: Modus ponens to R_8 and use R_4

R_9: $\neg(P_{12} \vee P_{21})$

5: Apply De Morgan's rule to R_9

$\neg P_{12} \wedge \neg P_{21}$ ⟶ No pit in [1,2] and [2,1]

Remember the KB:

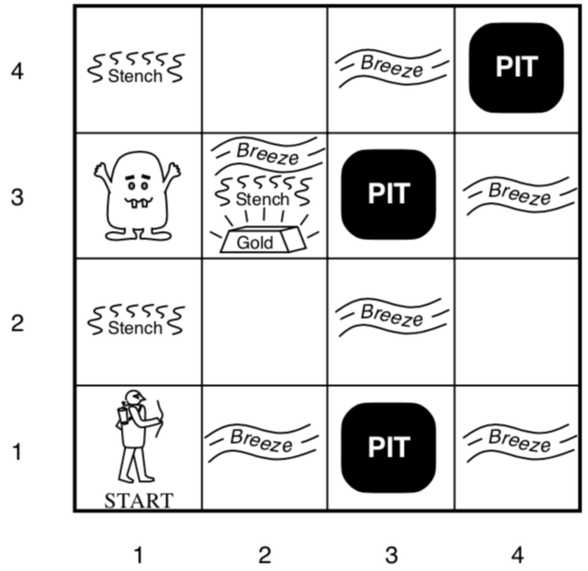
R_1: $\neg P_{11}$

R_2: $B_{11} \Leftrightarrow P_{12} \vee P_{21}$

R_3: $B_{21} \Leftrightarrow P_{11} \vee P_{22} \vee P_{31}$

R_4: $\neg B_{11}$

R_5: B_{21}



Resolution

We present with an example an inference technique called **Resolution**:

Suppose that in the Wumpus world we have R: $P_{11} \vee P_{22} \vee P_{31}$
and we add the fact $\neg P_{22}$ (no pit in [2,2]).

We say that P_{22} resolves R giving the resolvent $P_{11} \vee P_{31}$

This means: “If there is a pit in [1,1] or [2,2] or [3,1]” and it is not in [2,2] then it is in either [1,1] or [3,1]…
This is an example of unit resolution inference rule

Unit Resolution

Let $\mathbf{l_i}$ be literals (i=1,..,k) and

let **m** be complementary literal (negation) of l_i,

then the **unit resolution** formula is:

$$\frac{l_1 \vee \ldots \vee l_k, \quad m}{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k}$$

Generalised Resolution Rule

$$\frac{l_1 \vee \ldots \vee l_k\ ,\quad m_1\ \vee \ldots \vee m_n}{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

where $\mathbf{l_i}$ and $\mathbf{m_k}$ are complementary literals.

Note that the rule eliminates the complementary literal but adds all the rest of the additional information in the literals m.

Important: the resulting clause(*) must contain only one copy of each literal.

Copy removal is called factoring, e.g.: $A \vee A \rightarrow A$

(*) A **clause** is a conjunction of literals

Conjunctive Normal Form

Theorem: every sentence of propositional logic is logically equivalent to a conjuction of clauses, called conjunctive normal form (CNF)

Example: convert $P \iff (Q \vee R)$ in CNF

1) Biconditional elimination: $P \Rightarrow (Q \vee R) \wedge (Q \vee R) \Rightarrow P$
2) Implication elimination: $(\neg P \vee Q \vee R) \wedge (\neg (Q \vee R) \vee P)$
3) De Morgan law: $(\neg P \vee Q \vee R) \wedge [(\neg Q \wedge \neg R) \vee P]$
4) Distributive law: $\underbrace{(\neg P \vee Q \vee R)}_{\text{Clause}} \wedge \underbrace{(\neg Q \vee P)}_{\text{Clause}} \wedge \underbrace{(\neg R \vee P)}_{\text{Clause}}$ ⟶ CNF

Resolution Algorithm

The algorithm is based in the proof by contradiction, i.e. in order to show that KB⊨P, we show that (KB ∧ ¬A) cannot be satisfied.

The steps of the algorithms are:

1) Convert (KB ∧ ¬A) in CNF
2) Apply the resolution rule to the resulting clauses
3) Stop when:
    a) No new clauses to be added: then KB does not entail A
    b) Two clauses resolve in the empty clause then KB entails A

Empty clause = disjunction of no disjuncts: this is always false.

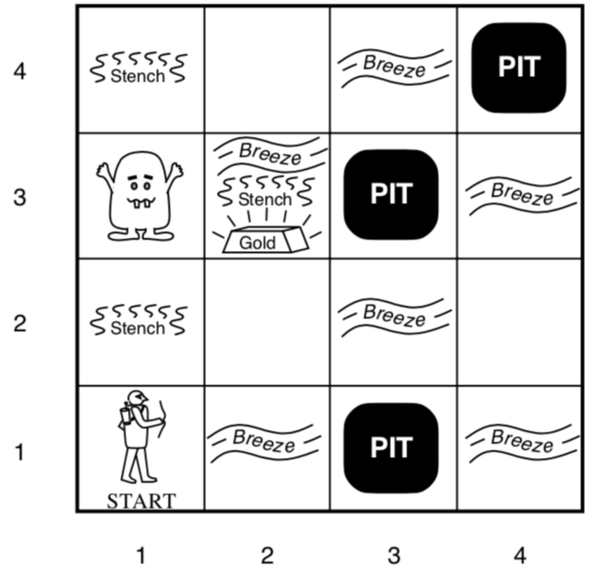
Example

If the agent is in [1,1], there is no breeze so there are no pits in [1,2] and [2,1]. The corresponding KB is:

$R_2 \wedge R_4 : (B_{11} \iff (P_{21} \vee P_{12})) \wedge \neg B_{11}$.

We would like to prove that there is no pit in [1,2], or $\neg P_{12}$.

For proving KB$\models \neg P_{12}$ we prove by contradiction KB$\wedge P_{12}$.

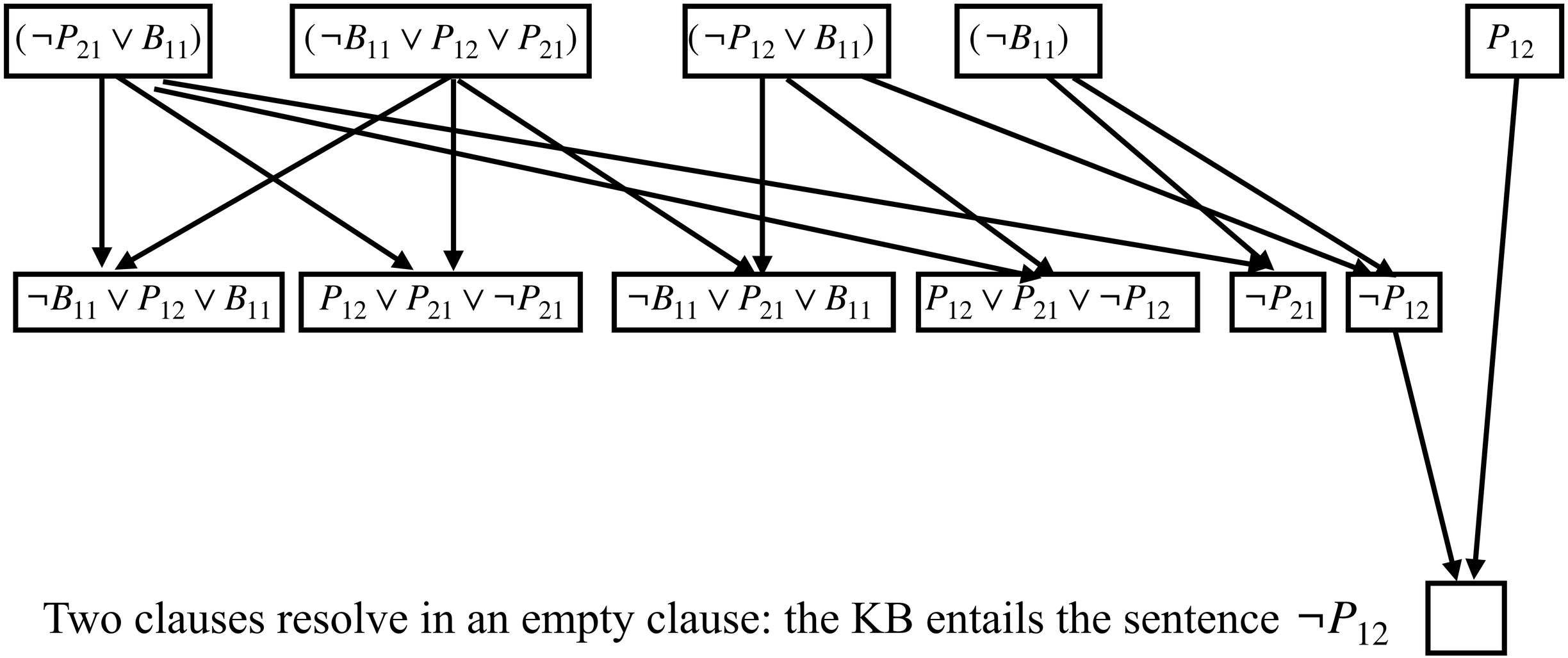


Reducing the last formula in CNF we have:

$(\neg P_{21} \vee B_{11}) \wedge (\neg B_{11} \vee P_{12} \vee P_{21}) \wedge (\neg P_{12} \vee B_{11}) \wedge (\neg B_{11}) \wedge P_{12}$

Now we apply **resolution** to the 5 clauses we found…

Example (cont'd)



Two clauses resolve in an empty clause: the KB entails the sentence $\neg P_{12}$

Resolution Algorithm's Code

```
function RESOLUTION(KB, A) returns true or false
    clauses <— set of clauses in the CNF of KB ∧ ¬A
    new <— { }
    while true do
        for each pair of clauses Ci,Cj in clauses do
            resolvents <— RESOLVE(Ci,Cj)
            if resolvents contains an empty clause then return true
            new <— new ∪ resolvents
        if new ⊆ clauses then return false
        clauses <— clauses ∪ new
```

Soundness and Completeness

Reminder:

A procedure (algorithm) is **sound** (or "truth-preserving") if derives only entailed sentences. In other words, it never produces wrong statements.

Completeness means that an algorithm can derive any entailed sentence. For a finite number of consequences, this can be verified in a certain time. For infinite consequences, this property can be tricky…

Resolution is sound and complete.

Horn Clauses

- Horn clauses are a relevant special case since they lead to polynomial scaling.
- Definition (Horn Clause): it is a clause with maximally one positive literal.
- Example 1: $\neg A_1 \vee \ldots \vee \neg A_n \vee B$
- Example 2: $\neg A_1 \vee \ldots \vee \neg A_n$
- Equivalent representation: $\boxed{\neg A_1 \vee \ldots \vee \neg A_n \vee B} \Longleftrightarrow \boxed{\bigwedge_i A_i \Rightarrow B.}$
- Horn clauses are closed under resolution: resolving two Horn clauses generate a Horn clause.
- This is the basis of logic programming (e.g with languages like PROLOG)

Forward Chaining Algorithm

1. Initialization: Start with an initial set of known facts and a set of rules in the form:

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n \Rightarrow B$$

2. Matching: Find all the rules whose premises (A_1,..,A_n) are satisfied by the current set of known facts.

3. Inference: For each rule that matches, deduce the conclusion B and add it to the set of known facts if it is not already present.

4. Iteration: Repeat the matching and inference steps until one of the following conditions is met: a) the goal proposition is derived, b) No new facts are being added

FC Example 1

Goal: derive Q

$P \Rightarrow Q$

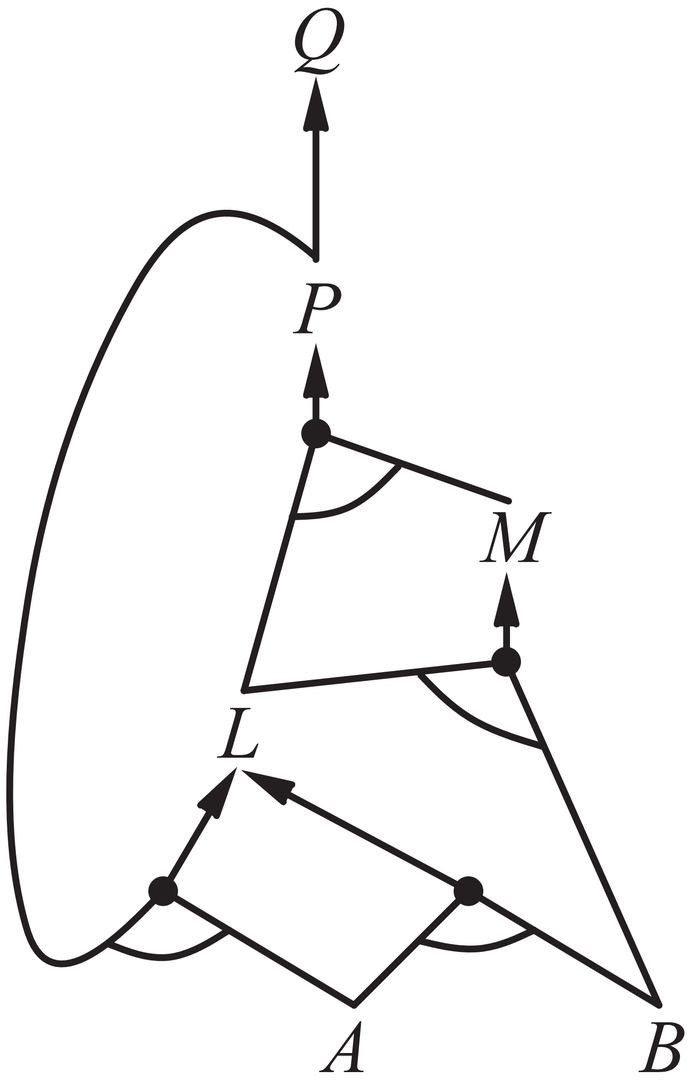
$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge B \Rightarrow L$

A

B



from: P. Norvig-S. Russell

AND-OR Diagram notation:

Edges joined with an arc: AND

Edges with no arc: OR.

The algorithm propagates through the graph and when a conjunction is encountered, it is resolved only when all the inputs are known.

FC Example 2

Known Facts: {A}

Rules: $A \Rightarrow B$

$B \Rightarrow C$

$C \Rightarrow D$

Goal: D

FC Example 2

Known Facts: {A,B} B added to the known facts

Rules: A⇒B Matching: derive B

B⇒C

C⇒D

Goal: D

FC Example 2

Known Facts: {A,B,C} C added to the known facts

Rules: $A \Rightarrow B$

$B \Rightarrow C$ Matching: derive C

$C \Rightarrow D$

Goal: D

FC Example 2

Known Facts: {A,B,C,D} D added to the known facts

Rules: A⇒B

B⇒C

C⇒D Matching: derive D —> D is the goal —> return.

Goal: D

FC Algorithm

```
function FC(KB, q) returns true or false
    count <— a table where count[c] is initially the number of symbols in clause c's premise.
    inferred <— a table where inferred[s] is initially false for all symbols.
    queue <— queue of symbols, initially all the ones known in the KB
    while queue is not empty do
        p <— POP(queue)
        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
                count[c] = count[c] - 1
                if count[c] = 0 then add c.CONCLUSION to queue
     return false
```

Backward Chaining

- FC is **data-driven**: can be used from an agent for updating his KB as new information comes in (new percepts). It is sound and complete.
- Backward chaining (BC) starts from the query q and then works backwards.
- With reference to the previous graph, BC works from Q to A or B.
- BC is a **goal-directed** algorithm and tries to answer to questions like “What should I do now?” looking into the KB for known facts/actions.
- BC corresponds to a search algorithm on AND-OR graphs.

Backward Chaining Algorithm

1. Initialization: Start with an goal-proposition G to prove.

2. Matching: Look into the KB for rules where G is a consequence:

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n \Rightarrow G$$

3. Sub-goal formation: For each rule that matches, the premises A_i become sub-goals. The algorithm tries to match all these sub-goals.

4. Recursion: Matching is repeated for each sub-goal recursively until: a) a sub-goal matches with a KB known fact or b) no rules can be found matching the sub-goal.

BC Example

Known Facts: {A, B}

Rules: $A \wedge B \Rightarrow C$

$C \Rightarrow D$

$B \Rightarrow E$

Goal: D

BC Example

Known Facts: {A, B}

Rules: $A \wedge B \Rightarrow C$

$C \Rightarrow D$ Matching: find a rule where D is the consequence

$B \Rightarrow E$ Sub-goal: prove C

Goal: D

BC Example

Known Facts: {A, B}

Rules: $A \wedge B \Rightarrow C$ Matching: find a rule where C (sub-goal) is the consequence

$C \Rightarrow D$ Sub-goal: prove A and B

$B \Rightarrow E$

Goal: D

BC Example

Known Facts: {A, B}

Rules: $A \wedge B \Rightarrow C$

$C \Rightarrow D$

$B \Rightarrow E$

Goal: D

Matching: A and B are known facts

back recursion: A and B prove C that proves D —> return.

Summary

- Rational agents require knowledge of their world in order to make rational decisions.
- With the help of a declarative (knowledge-representation) language, this knowledge is represented and stored in a knowledge base.
- Propositional logic is a possible way to achieve this.
- Logical implication is key.
- Logical implication can be automated using inference rules, e.g, **resolution**.
- Propositional logic becomes impractical when the world becomes too large (or infinite).