Part 5:
Knowledge Representation and Reasoning I:
Propositional Logic

("Aussagenlogik")

Univ.-Prof. Dr. Gerhard Widmer
Department of Computational Perception
Johannes Kepler University Linz

gerhard.widmer@jku.at
http://www.cp.jku.at/people/widmer

Overview

Motivation: The need for knowledge representation and reasoning

The Wumpus World

Propositional (Boolean) Logic

Reasoning in Propositional Logic: Inference Rules and Theorem Proving

A Sound and Complete Inference Procedure Based on a Single Inference Rule: *Resolution*

Efficient Inference in a Restricted Logic: Horn Clause Logic & Forward and Backward Chaining

Limitations of Approaches Considered So Far

Reflex Agent:

- Direct reaction to observations (stimuli)
- Fixed program, inflexible, difficult to write

Search-based Agent:

- Knows about actions and their consequences
- Can make its own plans to reach goals
- But: ignores newly incoming percepts
- Does not incorporate new observations (percepts) into its planning
- In particular: cannot reason about *unobservable* things

Goal: Knowledge-based Agents

Reflex Agent:

- Direct reaction to observations (stimuli)
- Fixed program, inflexible, difficult to write

Search-based Agent:

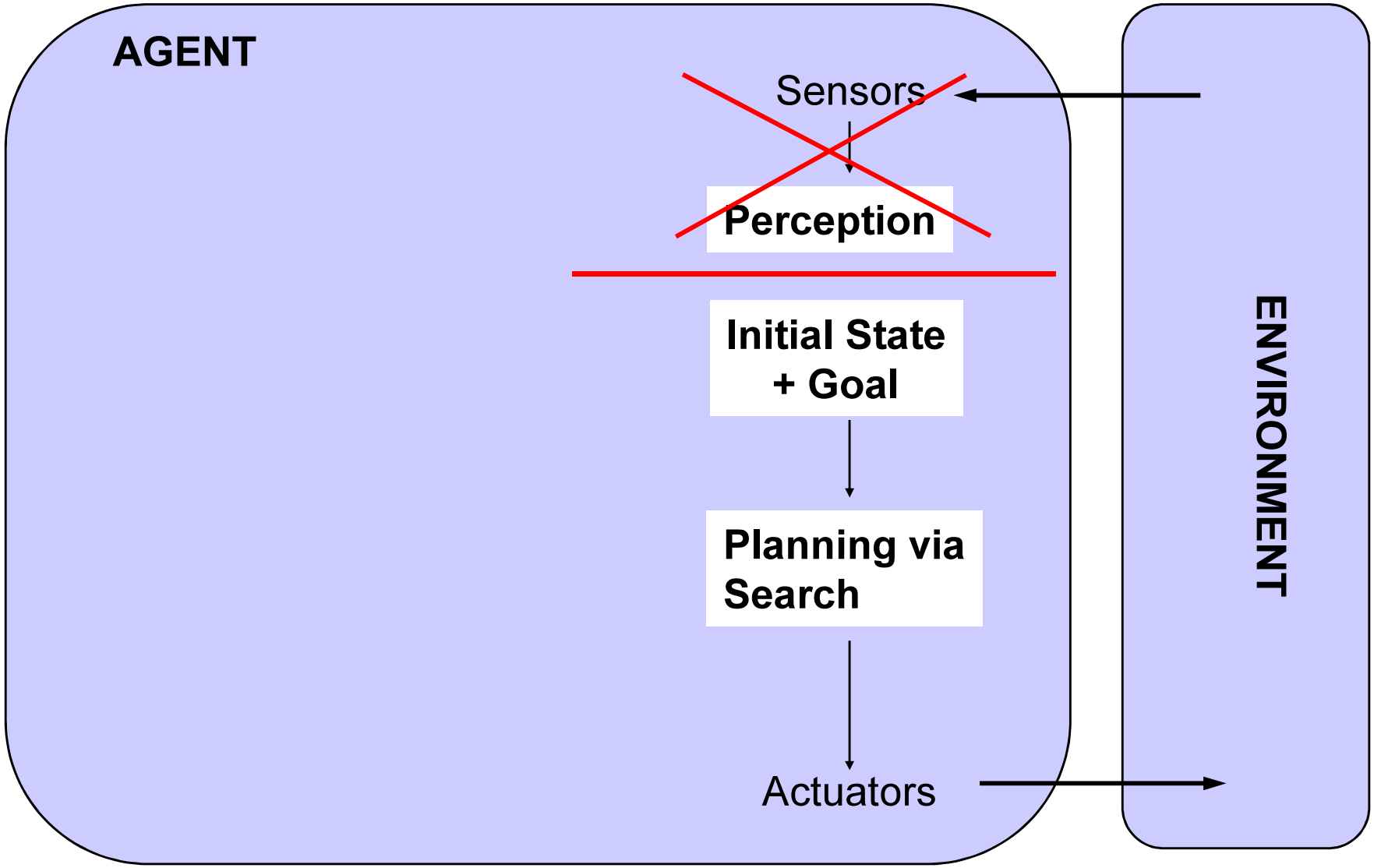
- Knows about actions and their consequences
- Can make its own plans to reach goals
- But: ignores newly incoming percepts
- Does not incorporate new observations (percepts) into its planning
- In particular: cannot reason about *unobservable* things

Knowledge-based Agent:

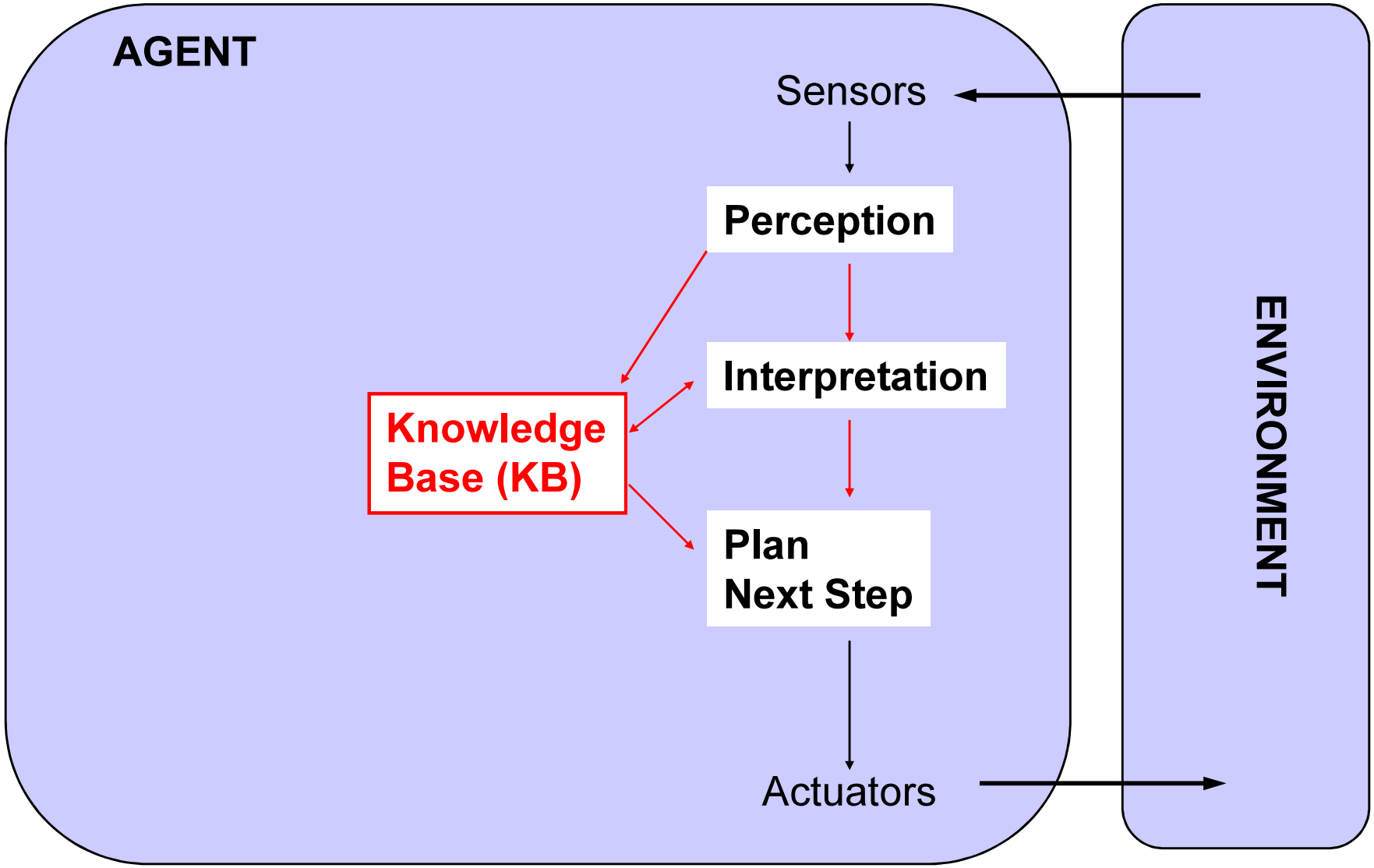
- Agent that can collect information, remember it, combine new information with known information to derive new facts, “explain” why certain things happen, “think about” the consequences of actions, make predictions

➔ Need for **knowledge representation and reasoning**

Search-based Agents



Knowledge-based Agents



Problem Solving = Knowledge + Inference …

← generic reasoning algorithms

← problem-dependent content (“knowledge”)

Declarative approach to building an agent:

- Tell it what it needs to know (build **knowledge base**)
- Equip it with general reasoning algorithm (**inference engine**)
- (Ideally: equip it with a **learning algorithm** …)

Knowledge Base (KB) = set of sentences in a **formal language**

Agents can be analysed at the

- **implementation level** (i.e., data structures in KB and algorithms that manipulate them)
- **knowledge level** (i.e., what they know, regardless of how it is implemented)

Basic Concepts

Knowledge

- information that an agent has about its “world” (including itself)

Individual “pieces” of knowledge:

- specific facts (“I am standing in front of a tree”)
- taxonomic knowledge about classes of objects (“a fir is a tree”, “trees are plants”, “plants are objects”)
- general laws (“when an object is immersed in water, it gets wet”)

Knowledge Representation Languages

- formalisms for representing knowledge in a formal way
- well-defined syntax and semantics
- best studied for many centuries: formal logic(s)

“Inference” / “Reasoning”

- drawing conclusions from general knowledge and specific observations
- inferring new information from given information
- explaining new observations by relating them to known facts
- making predictions

The Wumpus World

What the world is like:

- Cave consisting of **rooms** connected by passageways
- Somewhere in the cave is the **wumpus**, who eats anyone who enters its room
- The wumpus can be shot by the agent, but the agent has only **1 arrow**
- Some rooms contain **pits** that trap anyone who enters the room
- Somewhere in the cave is a **heap of gold**
- The agent does not know the configuration of the world (i.e., what is where)

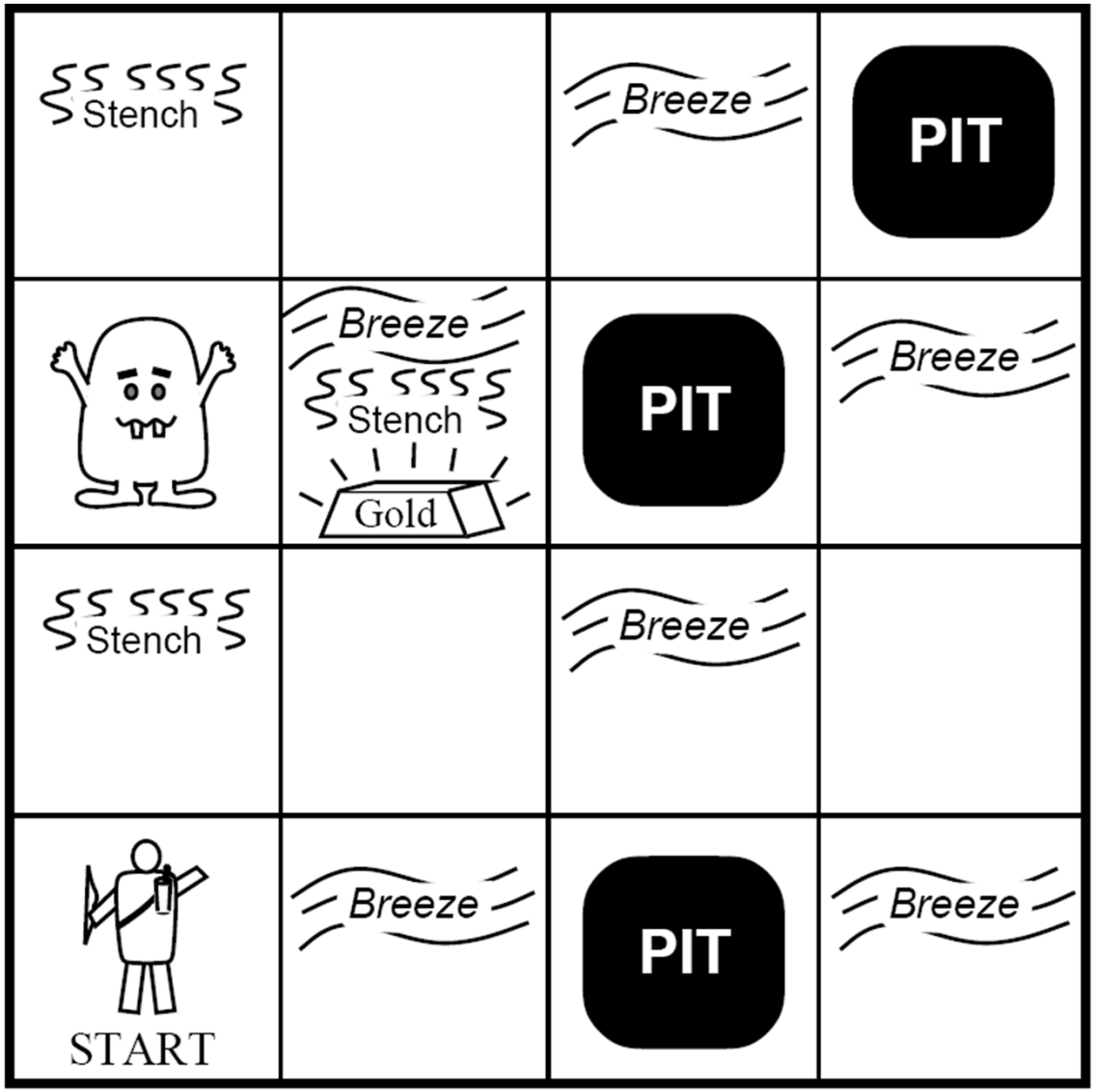
Percepts (what the agent can observe with its sensors):

- In the room containing the **wumpus** and in the directly neighbouring rooms, the agent will perceive a **stench**
- In the rooms directly neighbouring a **pit**, the agent will perceive a **breeze**
- In the room where the **gold** is, the agent will perceive a **glitter**
- When the agent walks into a **wall**, it will perceive a **bump**
- When the **wumpus** is **killed**, the agent will perceive a **scream**

The agent's goal:

- Find the gold and don't get trapped in a pit, or eaten by the wumpus

An Example Wumpus World



The Importance of Logical Reasoning

Challenge:

- Write an agent program that succeeds in arbitrary wumpus worlds

The agent must be able to

- represent (hypothesised) state of the world
- remember new observations and incorporate them into its knowledge
- update internal representation of the world
- reason about hidden properties of the world
- decide on appropriate actions

➔ Need for memory, knowledge representation, and reasoning abilities

Exploring Our Wumpus World

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

A = Agent

B = Breeze
G = Glitter, Gold
P = Pit
S = Stench
W = Wumpus
V = Visited
OK = Safe Room

Exploring Our Wumpus World

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

Percept 1:

[*None*,*None*,*None*,*None*,*None*]

Stench

Breeze

Glitter

Bump

Scream

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

Exploring Our Wumpus World

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 ~~**P?**~~ **OK**	2,1 **B** **A** **OK**	3,1 **P?**	4,1

Percept 2:

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

Stench

Breeze

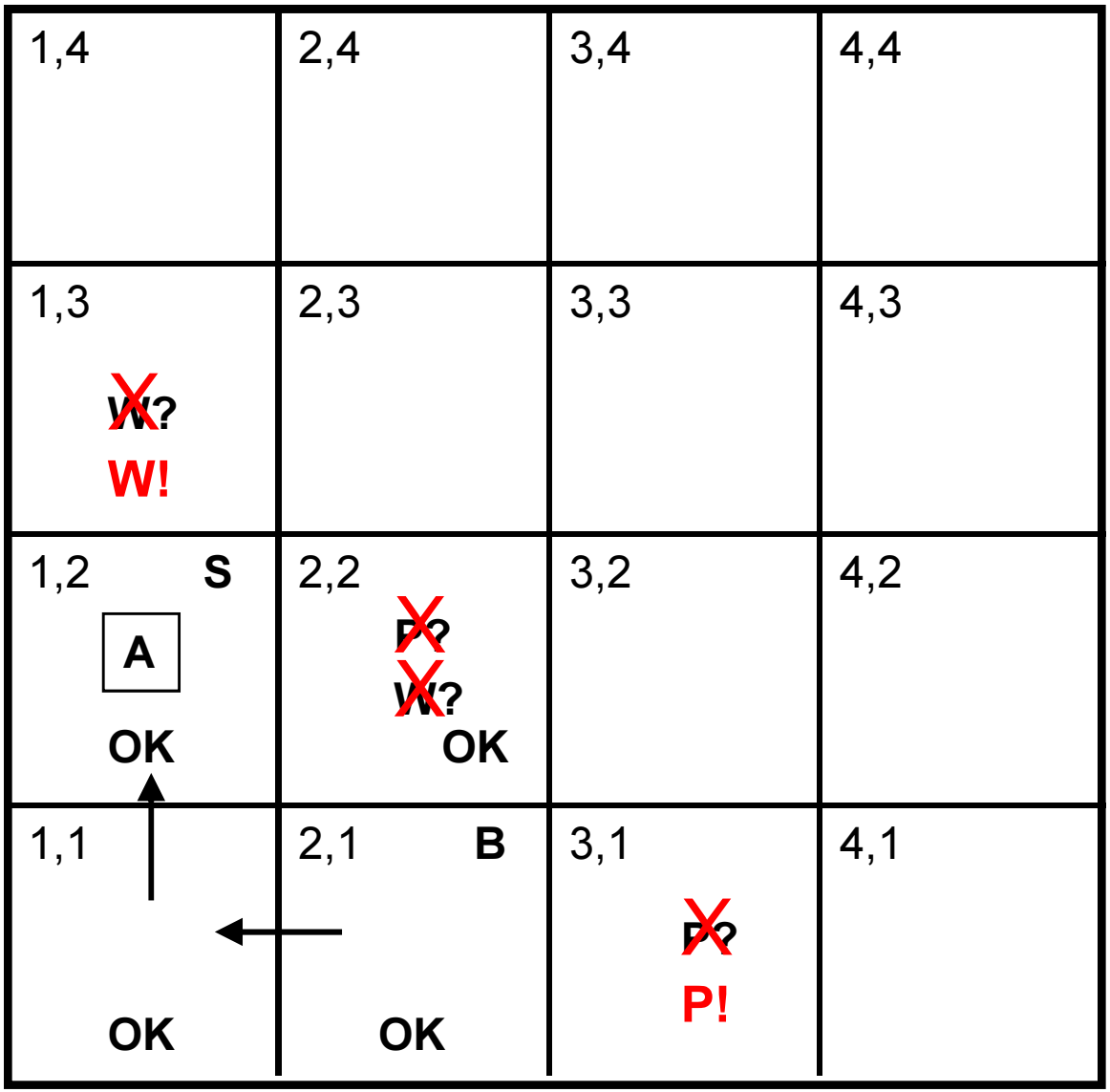
Glitter

Bump

Scream

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

Exploring Our Wumpus World



Percept 3:

[*Stench*, ~~*None*~~, *None*, *None*, *None*]

Stench

Breeze

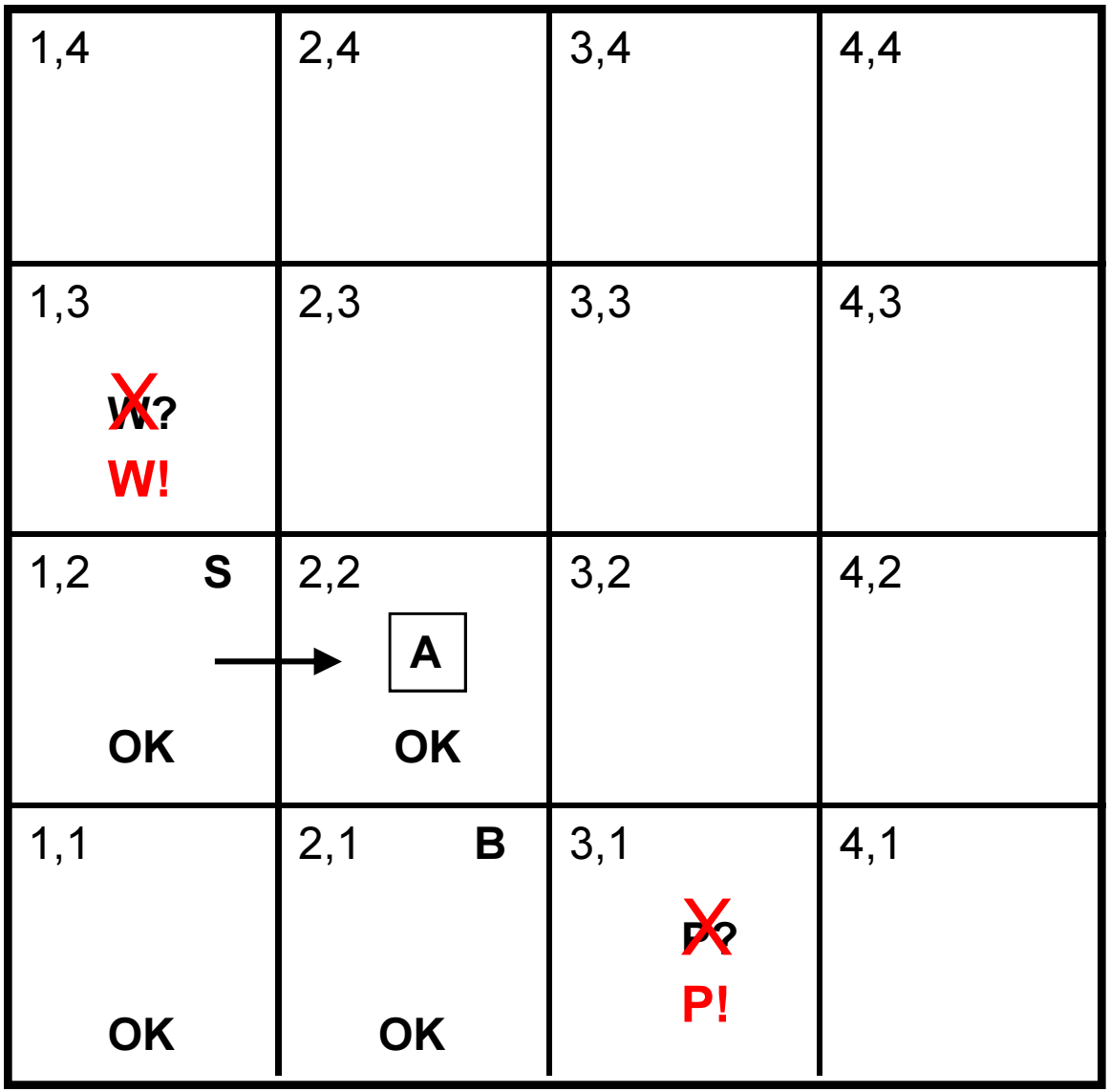
Glitter

Bump

Scream

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

Exploring Our Wumpus World

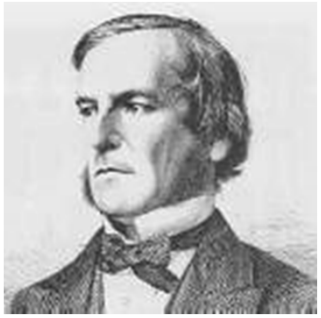


Percept 4:
[*None,None,None,None,None*]

try yourself to sketch how it might go on …

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

Propositional Logic
(Boolean Logic, Aussagenlogik)



George Boole (1816-1864)

George Boole has played a pivotal rôle in the development of the digital computer, for it was he who developed the now familiar 'Boolean Algebra', now a basis of digital manipulations. Although he showed exceptional early promise, his education was curtailed because, on the failure of his father's business, George had to support the family. At the age of 16, he began teaching and before he was 20, he opened a small school of his own and he had to combine his teaching duties with his own study of mathematics.
From 1839 onwards, he did publish his work in the *Cambridge Mathematical Journal and the Philosophical Transactions of the Royal Society.* In 1849, despite his lack of formal university education, he was appointed to the chair of mathematics at Queen's College, Cork.
While there, he published, in 1854, *An Investigation into the Laws of Thought, on which are Founded the Mathematical Theories of Logic and Probabilities.* With this work there began what came to be known as **Boolean Algebra.**
Boole died in 1864 and for many years after his death, his work on logic was regarded as being of theoretical interest but lacking in any practical use.

Propositional Logic
(Boolean Logic, Aussagenlogik)

- One of the simplest logics
- Too simple for most AI applications, but illustrates the basic ideas
- Basic building blocks:
  - there are two elementary ***truth values***: true and false
  - ***proposition symbols*** stand for elementary facts (‘propositions’)
  - there is a defined set of ***logical operators*** that can be used to combine propositions and sentences
- Expressions in propositional (and any other) logic are called ***sentences***
- Sentences are constructed from other sentences via logical operators

Propositional Logic
(Boolean Logic, Aussagenlogik)

The ***syntax*** of a logic specifies what sentences are permitted / can be constructed in the logic

Syntax of Propositional Logic (recursive definition):

1. Single **proposition symbols** P, Q, etc. and single **truth values** ($true, false$) are sentences ("atomic sentences")
2. If S is a sentence, $\neg S$ is a sentence (**negation**)
3. If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (**conjunction**)
4. If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (**disjunction**)
5. If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence (**implication**)
6. If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (**biconditional**)

Propositional Logic: Syntax in Backus-Naur Form (BNF)

$$
\begin{array}{rcl}
\mathit{Sentence} & \rightarrow & \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} & \rightarrow & \mathbf{True} \mid \mathbf{False} \mid \mathit{Symbol} \\
\mathit{Symbol} & \rightarrow & \mathbf{P} \mid \mathbf{Q} \mid \mathbf{R} \mid \ldots \\
\mathit{ComplexSentence} & \rightarrow & \neg\, \mathit{Sentence} \\
 & \mid & (\, \mathit{Sentence} \wedge \mathit{Sentence}\,) \\
 & \mid & (\, \mathit{Sentence} \vee \mathit{Sentence}\,) \\
 & \mid & (\, \mathit{Sentence} \Rightarrow \mathit{Sentence}\,) \\
 & \mid & (\, \mathit{Sentence} \Leftrightarrow \mathit{Sentence}\,)
\end{array}
$$

- Parentheses can be omitted if precedence is clear
- Precedence of operators: not, and, or, implication, biconditional

Propositional Logic: Semantics

Basic definitions:

- A **model** is an assignment of specific truth values (true or false) to every proposition symbol in a logical sentence.
  That is, a model specifies which propositions are true in a particular world, and which ones are false.

  Example: three proposition symbols P, Q, R
  ➔ $2^3 = 8$ possible models

- Every **sentence** in propositional logic is either true or false in a given model (that is, if the truth values of all elementary propositions are fixed)

- The ***semantics*** of a logic defines rules for determining the truth of any sentence with respect to a particular model
  (➔ the semantics of a logic defines the meaning of the logical operators "and", "or", "not", …)

Propositional Logic: Semantics

The semantics must specify how to compute the truth value of an arbitrary sentence, given a fixed model

The semantics of propositional logic (recursive definition):

1. Each atomic proposition P is either true or false (given by the model)
2. A sentence $\neg S$ is true iff S is false
3. $S_1 \wedge S_2$ is true iff S_1 is true **and** S_2 is true
4. $S_1 \vee S_2$ is true iff S_1 is true **or** S_2 is true (or both are true)
5. $S_1 \Rightarrow S_2$ is true iff S_1 is false **or** S_2 is true
   (i.e., is **false** iff S_1 is true **and** S_2 is false)
6. $S_1 \Leftrightarrow S_2$ is true iff $S_1 \Rightarrow S_2$ is true **and** $S_2 \Rightarrow S_1$ is true

"iff" = "if and only if"

Propositional Logic: Semantics in the Form of a Truth Table

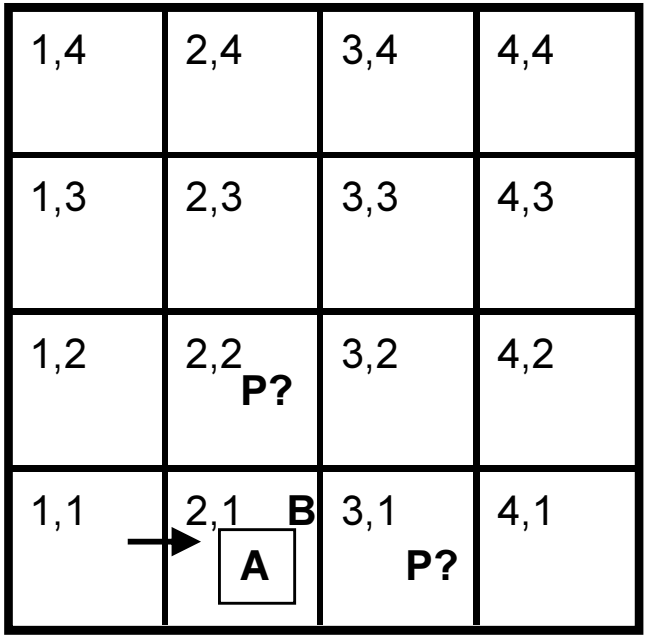
P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

P, Q … arbitrarily complex sentences

To use the table to compute, for example, the value of P => Q when P is *false* and Q is *true*, first look on the left for the row where P is *false* and Q is *true*. Then look in that row unter the P => Q column to see the result: *true*.

Another way to look at this is to think of each row (in the red box) as a **model**, and the entries under each column for that row as saying whether the corresponding sentence is **true in that model**.

Representing the Wumpus World in Propositional Logic: A Few Simple Steps



Goal:

- Construct knowledge base $KB2$ for wumpus world including all percepts after the agent's first move (to room $[2,1]$)
- Model only pits and breezes
- Ignore wumpus & gold and associated percepts (stench etc.)

Step 1: Invent proposition symbols:

- Let P_{ij} = "there is a pit in room $[i,j]$"
- Let B_{ij} = "there is a breeze in room $[i,j]$"

➔ Need 2 x 16 = 32 proposition symbols to represent a 4x4 world with pits and breezes only !

➔ In any specific world, some of these will be true, others false (but the true state of some of these may be unknown to the agent)

Representing the Wumpus World in Propositional Logic: A Few Simple Steps

Step 2: Constructing the knowledge base $KB2$

(as the *set of all sentences that are known to be true*)

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

"There is no pit in $[1,1]$":

$$R_1: \quad \neg P_{1,1} \quad (i.e., P_{1,1} = false)$$

"A square is breezy if and only if there is a pit in a neighbouring square"
(must be explicitly stated for each square …):

$$R_2: \quad B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

$$R_3: \quad B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

.....

First two observations (about breezes) that the agent made:

$$R_4: \quad \neg B_{1,1} \quad (i.e., B_{1,1} = false)$$

$$R_5: \quad B_{2,1} \quad (i.e., B_{2,1} = true)$$

➔ $KB2$ **can be regarded as single true sentence:** $KB2 = R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5$

Entailment and Logical Inference

Logical entailment ("logische Folge / Konsequenz"):

- relation between sentences
- $\alpha \models \beta$ means " α logically entails β" ("α bedingt β ") or "β follows logically from α" ("β folgt aus α")
- Intuitively: $\alpha \models \beta$ means "if α is true, β must also be true"

Inference ("logisches Schlussfolgern") =

- Deriving the truth value of a sentence from other given true sentences
- Deciding whether a sentence follows logically from a knowledge base

Notation:

- $KB \vdash_i \alpha$ means "α is derived (proven to be true) from KB by inference algorithm i "

Note:

- Symbol |- is different from |=
- |= is a general logical relation between sentences
- |- depends on a *specific algorithm*: an algorithm may not be able to derive all the sentences that are logically entailed by a knowledge base

Desirable Properties of Inference Algorithms

Soundness (Correctness):

- An inference algorithm is **sound** if every sentence it derives from KB is indeed entailed by KB, that is:
- Algorithm i is sound if, whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$

Completeness:

- An inference algorithm is **complete** if it can derive any sentence that is entailed by KB:
- i is complete if, whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$

Note:

- Later, we will define a logic (first-order logic) that is expressive enough for almost any relevant problem, and for which there exists a sound and complete inference procedure.
- In other words: the procedure will be able to answer any question whose answer follows logically from what is known by the KB *(if the answer is positive …)*

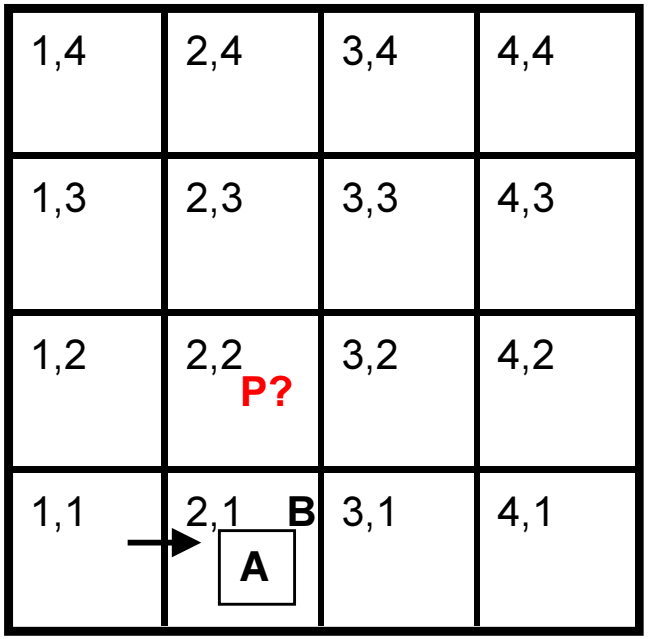
Inference in Propositional Logic

Remember:

Goal of logical inference: to decide whether $KB \models \alpha$ for some α

Example:

$KB2 \models P_{2,2}$? (from what the agent currently knows (KB2), can it conclude that there is a pit in $[2,2]$?)



Some Standard Inference Patterns in Propositional Logic

"Inference" = chain of (purely syntactic!) logical transformation steps that lead from known facts (KB) to desired conclusion. Each of these steps must be sound (i.e., provably correct).

Standard Reasoning Pattern #1: Modus Ponens:

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

← *If these are known to be true, …*

→ *… then this must also be true*

"Whenever any sentences of the form α => β and α are given, the sentence β can be inferred (and can be added to the knowledge base)"

Example:

given $(WumpusAhead \wedge Shoot) \Rightarrow WumpusDead$ and $(WumpusAhead \wedge Shoot)$
can safely conclude $WumpusDead$

Some Standard Inference Patterns in Propositional Logic

"Inference" = chain of (purely syntactic!) logical transformation steps that lead from known facts (KB) to desired conclusion

Standard Reasoning Pattern #2: AND-Elimination:

$$\frac{\alpha \wedge \beta}{\alpha}$$

Example:

given $(WumpusAhead \wedge WumpusAlive)$, can safely conclude $WumpusAlive$

➔ Both Modus Ponens and And-Elimination are **sound inference rules**

➔ Can be used whenever they apply, without need for enumerating models (and without knowing what α and β "mean" !)

Useful Logical Equivalences

Definition:
Two sentences α and β are **logically equivalent** ($\alpha \equiv \beta$)
if they are true in **exactly the same set of models**

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

Some Standard Reasoning Patterns in Propositional Logic

➔ **All logical equivalences can be used as inference rules: replace a sentence with a logically equivalent one**

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

Example: definition of biconditional elimination gives 2 inference rules:

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)} \qquad \frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}$$

Detecting Pits via Logical Inference (1)

Example: given knowledge base $KB2$:

"There is no pit in $[1,1]$":

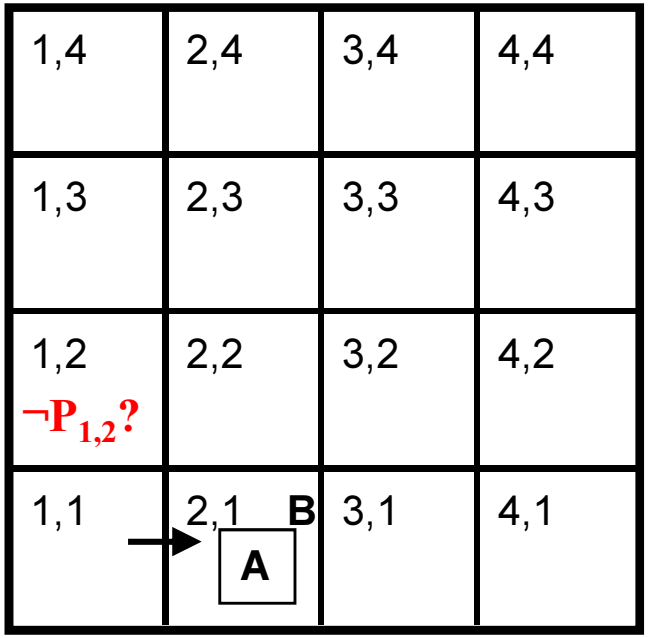
$$R_1 : \quad \neg P_{1,1}$$

"A square is breezy if and only if there is a pit in a neighbouring square":

$$R_2 : \quad B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$
$$R_3 : \quad B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

First two observations (about breezes):

$$R_4 : \quad \neg B_{1,1}$$
$$R_5 : \quad B_{2,1}$$



Question:

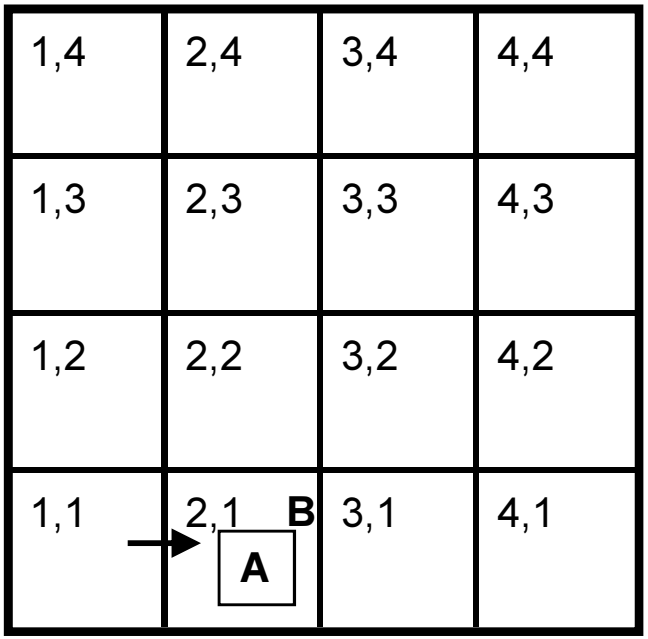
$KB2 \models \neg P_{1,2}$?

(given the state of knowledge about the world contained in $KB2$, can the agent conclude (prove to itself) that there is no pit in $[1,2]$?

Detecting Pits via Logical Inference (2)

$KB2$:
$R_1: \quad \neg P_{1,1}$
$R_2: \quad B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
$R_3: \quad B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
$R_4: \quad \neg B_{1,1}$
$R_5: \quad B_{2,1}$

Question: $KB2 \models \neg P_{1,2}$ **?**



Proof:

1. Apply *biconditional elimination* to R2; result:

$R_6: \quad (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

2. Apply *and-elimination* to R6; result:

$R_7: \quad (P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}$

3. Apply *contraposition* to R7; result:

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

Detecting Pits via Logical Inference (3)

$KB2$:
$R_1: \neg P_{1,1}$
$R_2: B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
$R_3: B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
$R_4: \neg B_{1,1}$
$R_5: B_{2,1}$

Question: $KB2 \models \neg P_{1,2}$ **?**

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
1,1	2,1 **B** **A**	3,1	4,1

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

4. Apply *Modus Ponens* with R8 and R4; result:

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

5. Apply *de Morgan's rule* to R9; result:

$R_{10}: \neg P_{1,2} \wedge \neg P_{2,1}$

6. Apply *and-elimination* to R10; completes the proof:

$R_{11}: \neg P_{1,2}$ ***q.e.d.***

Theorem Proving as Search

Notes:

- Logical derivation $KB \models \beta$ (sequence of logical reasoning steps) is called a **proof** (β is called a **theorem**)
- Proving is a **purely syntactic** process
- Finding a proof is a **search problem** (a hard one …):
  - *Initial state* = current knowledge base
  - *Possible states* = all possible knowledge bases (sets of sentences)
  - *Actions* = successor function = all possible applications of inference rules to any (combination of) sentence(s) in the current KB
    => *large branching factor! (which grows with every added sentence ..)*

➔ Can apply all search algorithms from previous chapters (in principle)

Properties of such a search-based inference algorithm:

- **sound** (if individual inference rules are sound)
- ***complete ?***
  - ➢ only if we use a complete search algorithm (e.g., iterative deepening) AND
  - ➢ only if set of available inference rules is complete!

A Single Complete Inference Rule: Resolution

Resolution =

a **single** inference rule that, when coupled with any complete search algorithm (e.g., iterative deepening depth-first), gives a **complete** and **sound** inference algorithm for propositional logic

Terminology:

- a **literal** is a proposition symbol (P), or a negated proposition symbol ($\neg P$)
- a **clause** is a disjunction of literals $l_1 \vee \cdots \vee l_k$

➔ a literal is a special case of a clause

A Single Complete Inference Rule: Resolution

Special Case: Unit Resolution rule:

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

where

- each l_j is a **literal**
- l_i and m are **complementary literals** (i.e., one is a negation of the other)
- for example: $l_i = \neg P,\ m = P$

➔ Unit resolution takes a clause and a literal and produces a new clause (with k - 1 literals)

Two simple examples of unit resolution:

1. $$\frac{P \vee Q \vee R, \quad \neg Q}{P \vee R}$$

2. $$\frac{\neg P \vee Q, \quad P}{Q}$$ (equivalent to $$\frac{P \Rightarrow Q, \quad P}{Q}$$ i.e., modus ponens!)

A Single Complete Inference Rule: Resolution

The General Resolution Rule:

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

where

- all l and m are **literals**
- l_i and m_j are **complementary literals** (i.e., one is a negation of the other)

➔ Resolution takes two clauses and produces a new clause
(with $k + n - 2$ literals, and the pair of complementary literals dropped)

A Single Complete Inference Rule: Resolution

A simple example of resolution:

$$\frac{P \vee Q, \quad \neg Q \vee R}{P \vee R}$$

Intuitively clear:

- Q cannot be true and false at the same time, thus either Q or $\neg Q$ is false
- if Q is false => P must be true (otherwise $P\ OR\ Q$ could not be true)
- if Q is true => $NOT\ Q$ is false => R must be true
- Q is either true or false => one of the above conclusions must hold
  => $P\ \ OR\ \ R$ must be true

Alternative view:

$(P \vee Q) \equiv (\neg P \Rightarrow Q),\ (\neg Q \vee R) \equiv (Q \Rightarrow R)$, thus

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

➔ in this case, resolution implements transitivity of implication …

Properties of the Resolution Rule

1. Resolution is **sound**

(intuitively clear, and easy to prove – consider simple example above)

Theorem (J. A. Robinson, 1965):

Any complete search algorithm, applying only the resolution rule, can derive any conclusion entailed by any knowledge base in propositional logic.

➔ 2. Resolution is **complete**

Conjunctive Normal Form (CNF)

Question:

The resolution rule applies only to disjunctions of literals (i.e., clauses); so how can it be applied to arbitrarily complex sentences?

Answer:

Every sentence α in propositional logic can be transformed into a conjunction of disjunctions of literals that is logically equivalent ($\equiv$) to α

➔ **Every knowledge base can be represented as a set of clauses!**

Definition:

A sentence expressed as a conjunction of disjunctions of literals:

$$(l_{11} \vee \cdots \vee l_{1k}) \wedge \cdots \wedge (l_{n1} \vee \cdots \vee l_{nm})$$

is said to be in **Conjunctive Normal Form (CNF)**

Conversion of Propositional Sentences into CNF

Example:

$$(P \Leftrightarrow (Q \vee R))$$

Step 1: Eliminate $\Leftrightarrow$, replace $\alpha \Leftrightarrow \beta$ with
$(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$

$$(P \Rightarrow (Q \vee R)) \wedge ((Q \vee R) \Rightarrow P)$$

Step 2: Eliminate $\Rightarrow$, replace $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$

$$(\neg P \vee (Q \vee R)) \wedge (\neg(Q \vee R) \vee P)$$

Step 3: Move negation inwards, using the following equivalences:

$\neg(\neg\alpha) \equiv \alpha$ double-negation elimination
$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$ De Morgan
$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$ De Morgan

$$(\neg P \vee Q \vee R) \wedge ((\neg Q \wedge \neg R) \vee P)$$

Step 4: Apply distributivity of OR over AND: $(\neg P \vee Q \vee R) \wedge (\neg Q \vee P) \wedge (\neg R \vee P)$

An Algorithm for Theorem Proving with the Resolution Rule in Propositional Logic

General Principle: Proof by Contradiction:

To show that $KB \models \alpha$, we try to show that $(KB \wedge \neg\alpha)$ is **unsatisfiable (false)**.

More precisely: show that from $(KB \wedge \neg\alpha)$, we can derive the **empty clause** (*false*, contradiction)

Algorithm:

- Convert $(KB \wedge \neg\alpha)$ into CNF
- **Loop:** Apply resolution rule to resulting clauses (for any pair that can be resolved); add new clauses to the set
- **End condition:**
  - if an application of the resolution rule derives the empty clause, then KB entails α ➔ ***proof successful***
  - if no more new clauses can be added, then KB does ***not*** entail α

Note: The empty clause is equivalent to *false* because (1) a disjunction requires at least one disjunct to be true; (2) the empty clause results (only) from resolving two contradictory unit clauses (e.g., P and $\neg P$)

A Resolution Algorithm for Propositional Logic

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

Note: Function PL-RESOLVE returns the set of all possible clauses obtained by resolving its two inputs.

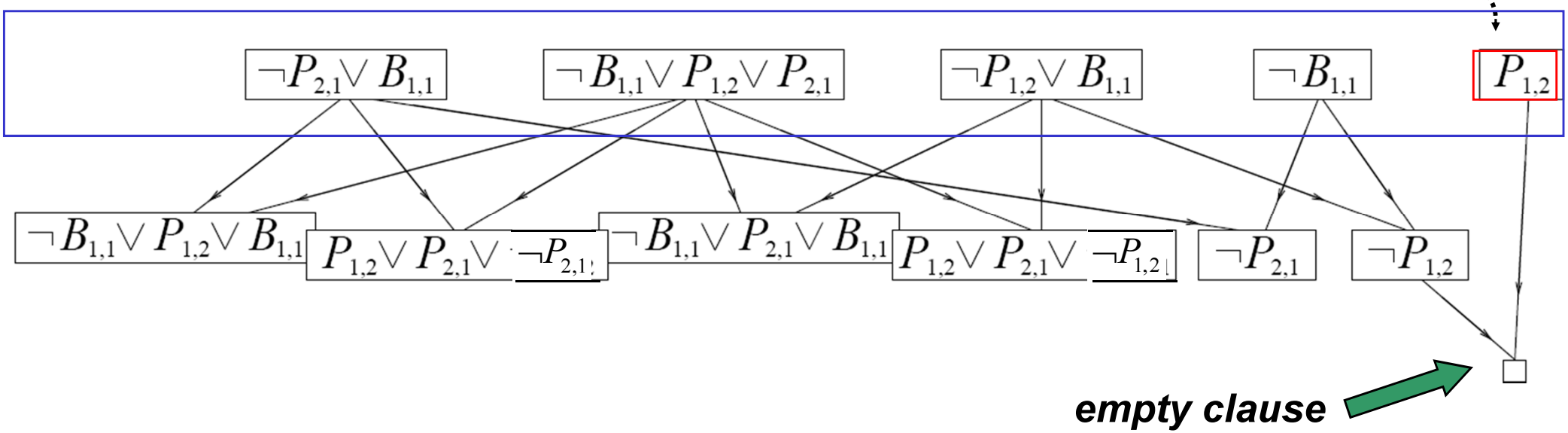
A Simple Example from the Wumpus World

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

To be proved: $\alpha = \neg P_{1,2}$

$\neg\alpha$

Proof by resolution:



Problem: Number of clauses that can be derived can be exponential

➔ **Again:** Logical inference is an exponential problem, in the worst case

Efficient Reasoning in Horn Clause Logic

Problem: Complexity of logical inference is exponential, in the worst case.
(Will not prove this in this lecture, but easy to show)

Idea: Reduce complexity of inference by restricting it to a simpler (limited) kind of logic.

Horn Clause Logic is a specific logic with restricted syntax and expressiveness

Terminology:

- a **literal** is a proposition symbol, or a negated proposition symbol
- a **clause** is a disjunction of literals

Efficient Reasoning in Horn Clause Logic

Definition:

A **Horn Clause** is a disjunction of literals of which **at most one** is positive (i.e., without negation)

Examples:

$(\neg P \vee \neg Q \vee R)$	is a Horn clause
$(\neg P \vee Q \vee R)$	is not a Horn clause
P	is a Horn clause (a "fact")
$\neg P$	is a Horn clause

➔ **Propositional Horn Clause Logic** is a logic where only propositional Horn clauses are permitted

Efficient Reasoning in Horn Clause Logic

Relevance of Horn clauses:

1. Every Horn clause can be written as an implication with a single positive literal to the right:

$$(\neg P \vee \neg Q \vee R) \equiv (\neg(P \wedge Q) \vee R) \equiv ((P \wedge Q) \Rightarrow R)$$

   Implications of this form are an intuitive way of formalising knowledge **("IF-THEN rules")**

2. Inference with Horn clauses can be done via **Forward Chaining** and **Backward Chaining** (very natural and intuitive forms of inference)

3. Deciding entailment with Horn clauses can be done in $O(n)$ time!

BUT: **Horn clauses are severely limited: cannot represent everything that propositional logic can represent !** (simple example: $P \vee Q$)

Forward Chaining

Linear time algorithm for deriving the truth of a **literal** Q from a KB

Basic idea:

- Work forward from known facts
- When all conditions of an implication (rule) are known to be true, add the conclusion of the rule (another fact) to the KB
- Continue until Q is added (➔ Q proved),
  or no more inferences are possible (➔ Q cannot be derived)

➔ successive applications of Modus Ponens in forward direction
➔ a kind of “data-driven” or “fact-driven” reasoning

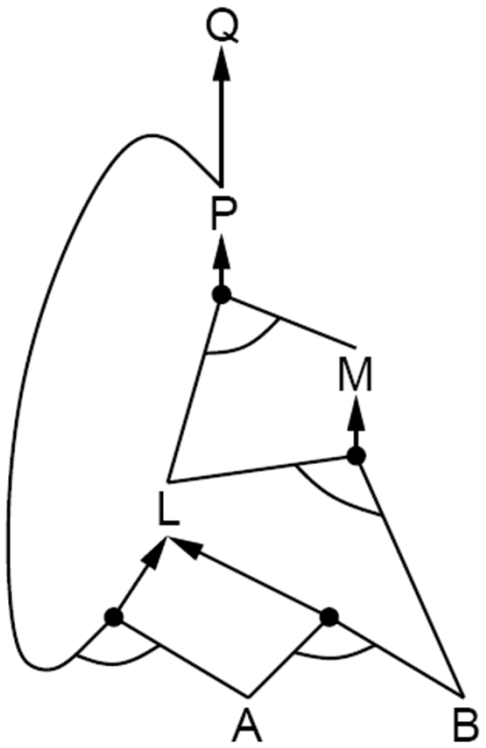
Forward Chaining: An Example

KB (2 facts, 5 rules):

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

Goal: prove that Q is true

Note: Rules in the KB imply an **AND-OR-Graph**:



Proof by Forward Chaining: An Example

$P \Rightarrow Q$ (5)

$L \wedge M \Rightarrow P$ (3)

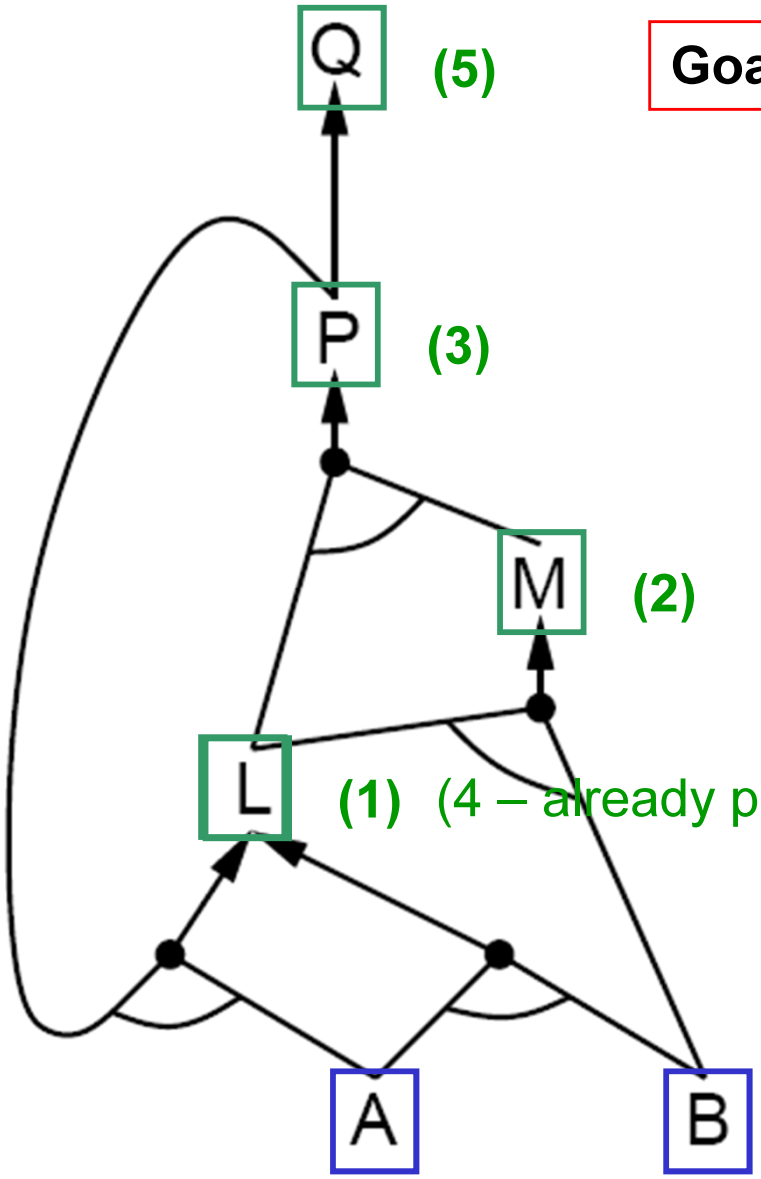
$B \wedge L \Rightarrow M$ (2)

$A \wedge P \Rightarrow L$ (4)

$A \wedge B \Rightarrow L$ (1)

A

B



Goal: prove that Q is true

Backward Chaining

Linear time algorithm for deriving a **literal** Q from a KB

Basic idea:

- Work backwards from the "goal" Q
- If Q is known (i.e., is in the KB) ➔ stop; proof successful
- Otherwise: find those rules in KB that would conclude Q (i.e., implicationss of the form $l_1 \wedge \ldots \wedge l_n \Rightarrow Q$)
- For each of these, try to prove all preconditions l_i (one by one, recursively, via backward chaining)
- If all preconditions l_i of a rule $l_1 \wedge \ldots \wedge l_n \Rightarrow Q$ are proved, add its conclusion Q to KB

➔ a kind of "goal-directed reasoning" ➔ more focused

Note: Backward chaining (combined with *first-order* Horn clauses) is the basic proof strategy underlying the programming language PROLOG. (First-order Horn clauses will be introduced in next lecture ...)

Proof by Backward Chaining: An Example

$P \Rightarrow Q$ (1)

$L \wedge M \Rightarrow P$ (2)

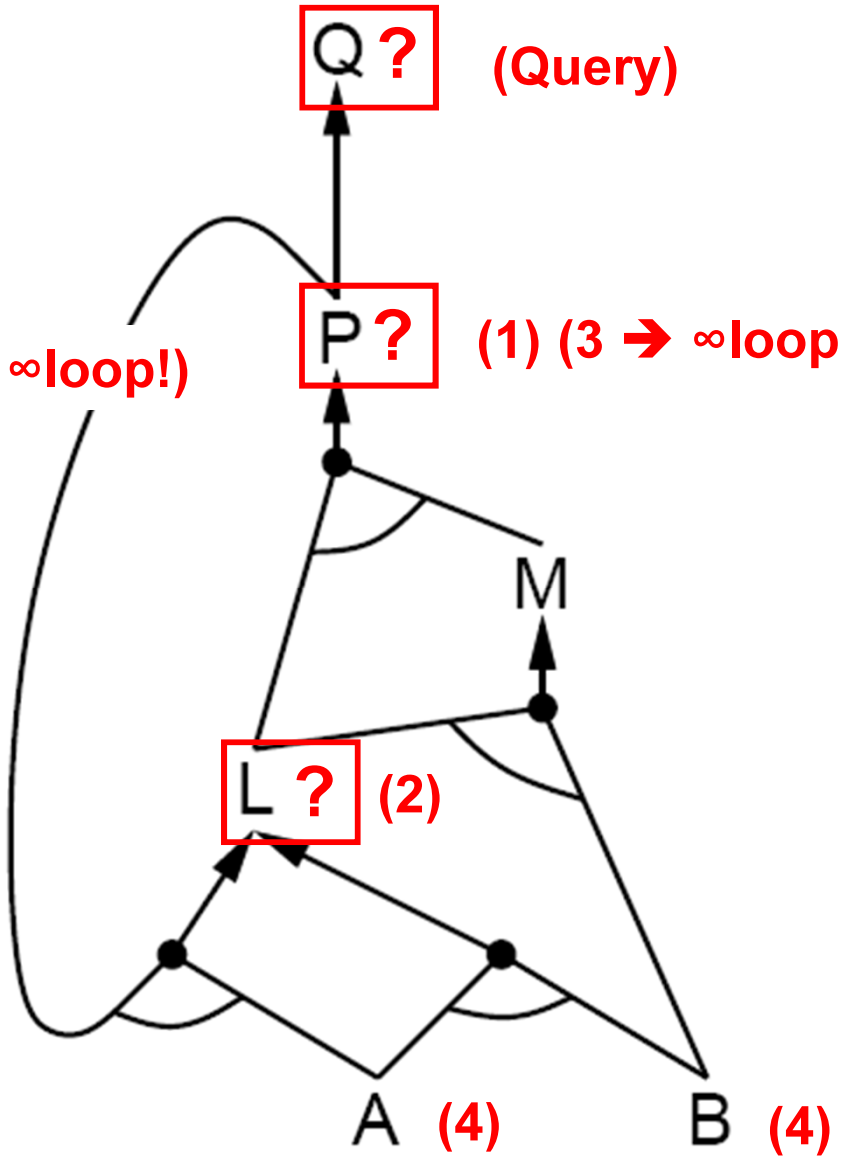
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$ (3 ➔ ∞loop!)

$A \wedge B \Rightarrow L$ (4)

A

B



Proof by Backward Chaining: An Example

$P \Rightarrow Q$ (1)

$L \wedge M \Rightarrow P$ (2)

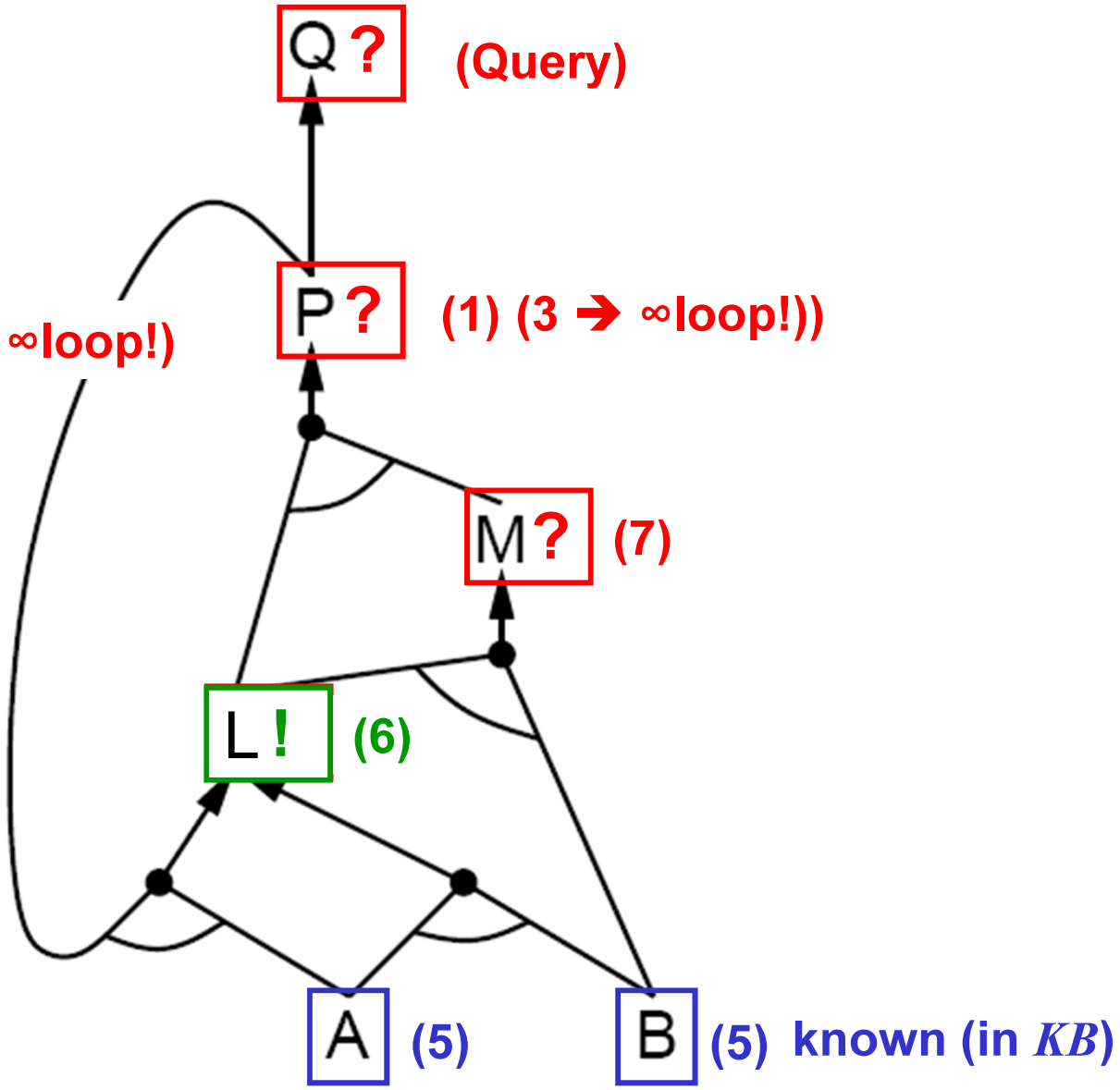
$B \wedge L \Rightarrow M$ (7)

$A \wedge P \Rightarrow L$ (3 ➔ ∞loop!)

$A \wedge B \Rightarrow L$ (6)

A (5)

B (5)



Proof by Backward Chaining: An Example

$P \Rightarrow Q$ (9)

$L \wedge M \Rightarrow P$ (8)

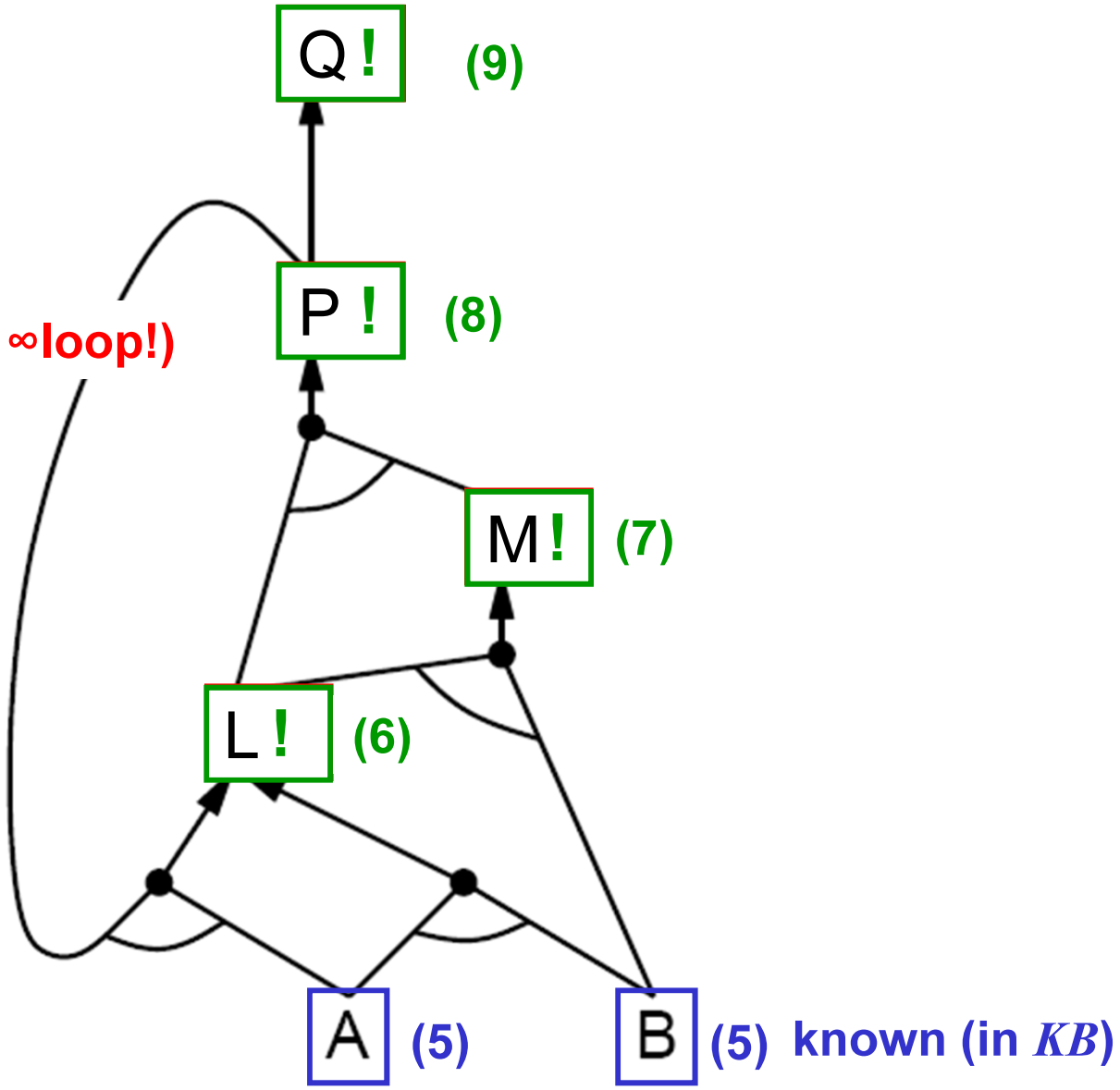
$B \wedge L \Rightarrow M$ (7)

$A \wedge P \Rightarrow L$ (3 ➔ ∞loop!)

$A \wedge B \Rightarrow L$ (6)

A (5)

B (5)



Summary

- **Logical agents** apply inference algorithms (domain-independent) to a knowledge base (domain-dependent), to derive new information
- **Basic concepts of logic:**
  - ***Syntax:*** defines formal structure of sentences
  - ***Semantics:*** truth of sentences with respect to models
  - ***Entailment:*** necessary truth of one sentence given another
  - ***Inference:*** procedure of deciding if a sentence is entailed by other sentences (the knowledge base)
  - ***Soundness:*** inference procedure produces only entailed sentences
  - ***Completeness:*** inference procedure produces all entailed sentences
- **Resolution** is a **sound and complete inference procedure** for propositional logic
- Worst-case **complexity** of resolution (and propositional inference in general): **exponential** in size of model (i.e., number of proposition symbols)
- **Forward chaining** and **backward chaining** are linear-time algorithms (sound and complete) for a specific, restricted sub-language (**Horn clauses**)