Artificial Intelligent

Sem. Ganjil 2023/2024

09. Logical Agent

Lionov

Overview

- Knowledge-Based Agent
- The Wumpus World
- Propositional Logic
- Method Checking
- Theorem Proving
- Intro. to First Order Logic

Preliminaries

Problems with Problem-Solving Agent:

Problems with Problem-Solving Agent:

- “know” very limited thing

Problems with Problem-Solving Agent:

- "know" very limited thing
- requires fully observability and static environment

Preliminaries

Problems with Problem-Solving Agent:

- “know” very limited thing
- requires fully observability and static environment

Human can acquire new information:

Preliminaries

Problems with Problem-Solving Agent:

- “know” very limited thing
- requires fully observability and static environment

Human can acquire new information:

- combine raw knowledge and experience

INFORMATIKA UNPAR

Preliminaries

Problems with Problem-Solving Agent:

- “know” very limited thing
- requires fully observability and static environment

Human can acquire new information:

- combine raw knowledge and experience
- using reasoning

INFORMATIKA UNPAR

Preliminaries

Problems with Problem-Solving Agent:

- “know” very limited thing
- requires fully observability and static environment

Human can acquire new information:

- combine raw knowledge and experience
- using reasoning

example: playing minesweeper

Preliminaries

Problems with Problem-Solving Agent:

- “know” very limited thing
- requires fully observability and static environment

Human can acquire new information:

- combine raw knowledge and experience
- using reasoning

example: playing minesweeper

Knowledge-Based Agent takes actions that

INFORMATIKA UNPAR

Preliminaries

Problems with Problem-Solving Agent:

- “know” very limited thing
- requires fully observability and static environment

Human can acquire new information:

- combine raw knowledge and experience
- using reasoning

example: playing minesweeper

Knowledge-Based Agent takes actions that

- use a process of reasoning: needs knowledge to choose actions

Problems with Problem-Solving Agent:

- "know" very limited thing
- requires fully observability and static environment

Human can acquire new information:

- combine raw knowledge and experience
- using reasoning

example: playing minesweeper

Knowledge-Based Agent takes actions that

- use a process of reasoning: needs knowledge to choose actions
- represent its internal representation of knowledge

Preliminaries

Problems with Problem-Solving Agent:

- “know” very limited thing
- requires fully observability and static environment

Human can acquire new information:

- combine raw knowledge and experience
- using reasoning

example: playing minesweeper

Knowledge-Based Agent takes actions that

- use a process of reasoning: needs knowledge to choose actions
- represent its internal representation of knowledge
- knowledge = *sentences* in a knowledge representation language (formal language).

Preliminaries

Problems with Problem-Solving Agent:

- “know” very limited thing
- requires fully observability and static environment

Human can acquire new information:

- combine raw knowledge and experience
- using reasoning

example: playing minesweeper

Knowledge-Based Agent takes actions that

- use a process of reasoning: needs knowledge to choose actions
- represent its internal representation of knowledge
- knowledge = *sentences* in a knowledge representation language (formal language).
- A sentence is an assertion about the world.

Logical AI

"The idea is that an agent can represent knowledge of its world, its goals and the current situation by sentences in logic and decide what to do by inferring that a certain action or course of action is appropriate to achieve its goals."

John McCarthy

Knowledge-Based Agent

INFORMATIKA
UNPAR

Knowledge-Based Agent

Knowledge-Based Agent is composed of:

Knowledge-Based Agent is composed of:

- Knowledge base or $\mathcal{KB}$ (domain-specific content):

Knowledge-Based Agent is composed of:

- Knowledge base or $\mathcal{KB}$ (domain-specific content):
  - set of senteces (that represent facts/belief about the environment)

Knowledge-Based Agent is composed of:

- Knowledge base or $\mathcal{KB}$ (domain-specific content):
  - set of senteces (that represent facts/belief about the environment)
  - expressed in a *knowledge representation language*

Knowledge-Based Agent is composed of:

- Knowledge base or $\mathcal{KB}$ (domain-specific content):
  - set of senteces (that represent facts/belief about the environment)
  - expressed in a *knowledge representation language*
  - initially contains some background knowledge

Knowledge-Based Agent is composed of:

- Knowledge base or $\mathcal{KB}$ (domain-specific content):
  - set of senteces (that represent facts/belief about the environment)
  - expressed in a *knowledge representation language*
  - initially contains some background knowledge
- Inference mechanism (domain-independent algorithm):

Knowledge-Based Agent is composed of:

- Knowledge base or $\mathcal{KB}$ (domain-specific content):
  - set of senteces (that represent facts/belief about the environment)
  - expressed in a *knowledge representation language*
  - initially contains some background knowledge
- Inference mechanism (domain-independent algorithm):
  - deriving new sentences from old

Knowledge-Based Agent

Knowledge-Based Agent is composed of:

- Knowledge base or $\mathcal{KB}$ (domain-specific content):
  - set of senteces (that represent facts/belief about the environment)
  - expressed in a *knowledge representation language*
  - initially contains some background knowledge
- Inference mechanism (domain-independent algorithm):
  - deriving new sentences from old
  - use declarative approach (instead of procedural)

INFORMATIKA UNPAR

Knowledge-Based Agent is composed of:

- Knowledge base or $\mathcal{KB}$ (domain-specific content):
  - set of senteces (that represent facts/belief about the environment)
  - expressed in a *knowledge representation language*
  - initially contains some background knowledge
- Inference mechanism (domain-independent algorithm):
  - deriving new sentences from old
  - use declarative approach (instead of procedural)
  - add new sentences (TELL)

Knowledge-Based Agent is composed of:

- Knowledge base or $\mathcal{KB}$ (domain-specific content):
  - set of senteces (that represent facts/belief about the environment)
  - expressed in a *knowledge representation language*
  - initially contains some background knowledge
- Inference mechanism (domain-independent algorithm):
  - deriving new sentences from old
  - use declarative approach (instead of procedural)
  - add new sentences (TELL)
  - query what is known (ASK)

Knowledge-Based Agent

Knowledge-Based Agent is composed of:

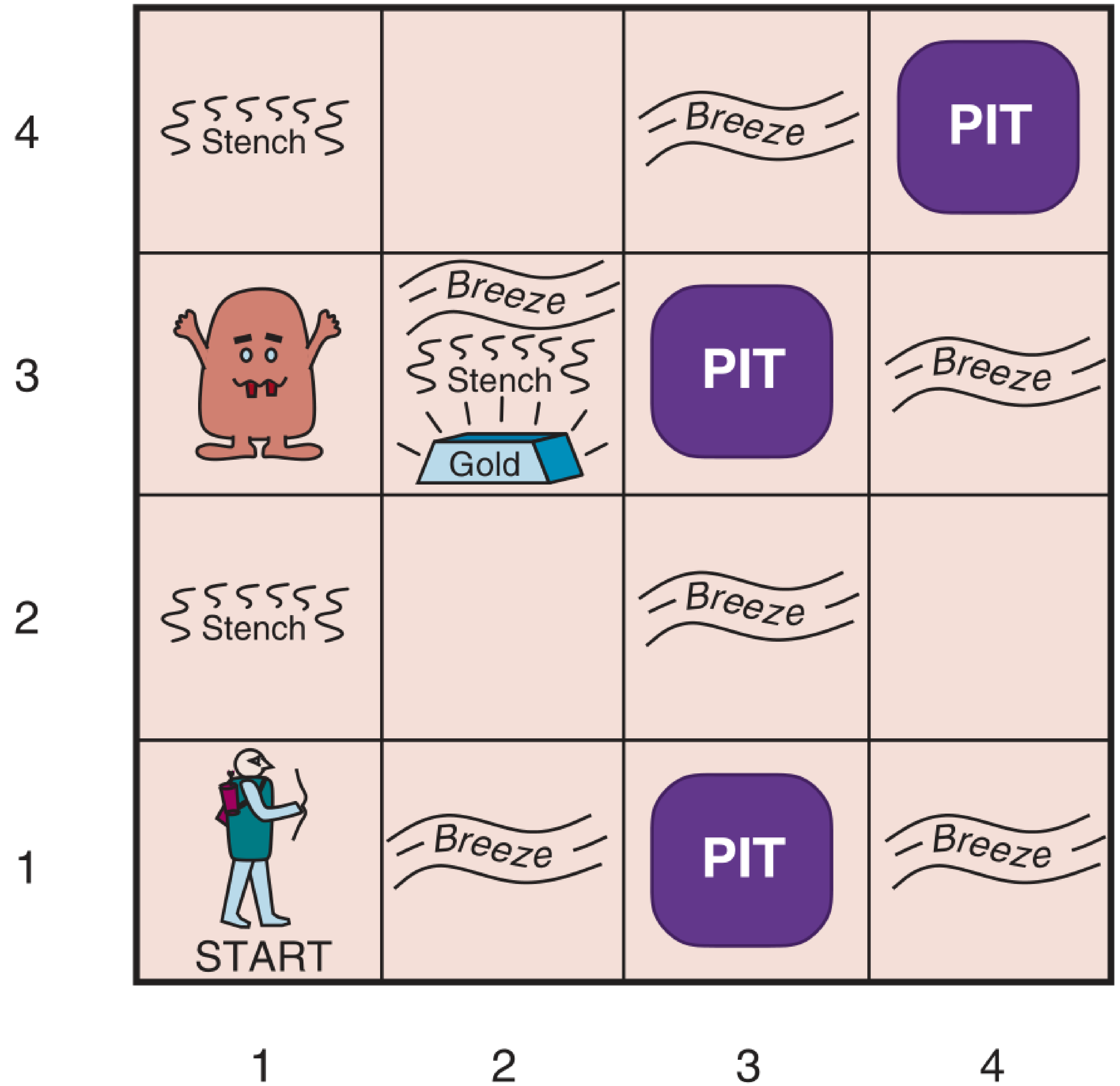
- Knowledge base or $\mathcal{KB}$ (domain-specific content):
  - set of senteces (that represent facts/belief about the environment)
  - expressed in a *knowledge representation language*
  - initially contains some background knowledge
- Inference mechanism (domain-independent algorithm):
  - deriving new sentences from old
  - use declarative approach (instead of procedural)
  - add new sentences (TELL)
  - query what is known (ASK)

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*))
  $t \leftarrow t + 1$
  return *action*

INFORMATIKA UNPAR

The Wumpus World



INFORMATIKA UNPAR

PEAS of the Wumpus world

PEAS of the Wumpus world

Performance: +1000 take the gold and climb out the cave. -1000 for falling into a pit or being eaten by the wumpus. -10 for using the arrow and -1 for each action taken. The game ends either when the agent dies or when the agent climbs out of the cave.

PEAS of the Wumpus world

Performance: +1000 take the gold and climb out the cave. -1000 for falling into a pit or being eaten by the wumpus. -10 for using the arrow and -1 for each action taken. The game ends either when the agent dies or when the agent climbs out of the cave.

Environment: 4x4 grid rooms (walls surrounding the grid), Start in [1,1], facing east. The locations of the gold and the Wumpus are chosen randomly (not at [1,1]) with a uniform distribution. Each square (not at [1,1]) can be a pit with probability 0.2.

PEAS of the Wumpus world

Performance: +1000 take the gold and climb out the cave. -1000 for falling into a pit or being eaten by the wumpus. -10 for using the arrow and -1 for each action taken. The game ends either when the agent dies or when the agent climbs out of the cave.

Environment: 4x4 grid rooms (walls surrounding the grid), Start in [1,1], facing east. The locations of the gold and the Wumpus are chosen randomly (not at [1,1]) with a uniform distribution. Each square (not at [1,1]) can be a pit with probability 0.2.

Sensors: *Stench* and *Breeze*: adjacent to the wumpus and a pit, respectively. *Glitter*: square with gold. *Bump*: walks into a wall. *Scream*: the wumpus is killed. Given to the agent as a list of five symbols, ex.: [Stench, Breeze, None, None, None].

INFORMATIKA UNPAR

PEAS of the Wumpus world

Performance: +1000 take the gold and climb out the cave. -1000 for falling into a pit or being eaten by the wumpus. -10 for using the arrow and -1 for each action taken. The game ends either when the agent dies or when the agent climbs out of the cave.

Environment: 4x4 grid rooms (walls surrounding the grid), Start in [1,1], facing east. The locations of the gold and the Wumpus are chosen randomly (not at [1,1]) with a uniform distribution. Each square (not at [1,1]) can be a pit with probability 0.2.

Sensors: *Stench* and *Breeze*: adjacent to the wumpus and a pit, respectively. *Glitter*: square with gold. *Bump*: walks into a wall. *Scream*: the wumpus is killed. Given to the agent as a list of five symbols, ex.: [Stench, Breeze, None, None, None].

Actuators: *Forward*, *TurnLeft* or *TurnRight* (both $90°$), *Grab*, *Shoot* (used only once), *Climb* (only from [1,1]).

INFORMATIKA
UNPAR

PEAS of the Wumpus world

Performance: +1000 take the gold and climb out the cave. -1000 for falling into a pit or being eaten by the wumpus. -10 for using the arrow and -1 for each action taken. The game ends either when the agent dies or when the agent climbs out of the cave.

Environment: 4x4 grid rooms (walls surrounding the grid), Start in [1,1], facing east. The locations of the gold and the Wumpus are chosen randomly (not at [1,1]) with a uniform distribution. Each square (not at [1,1]) can be a pit with probability 0.2.

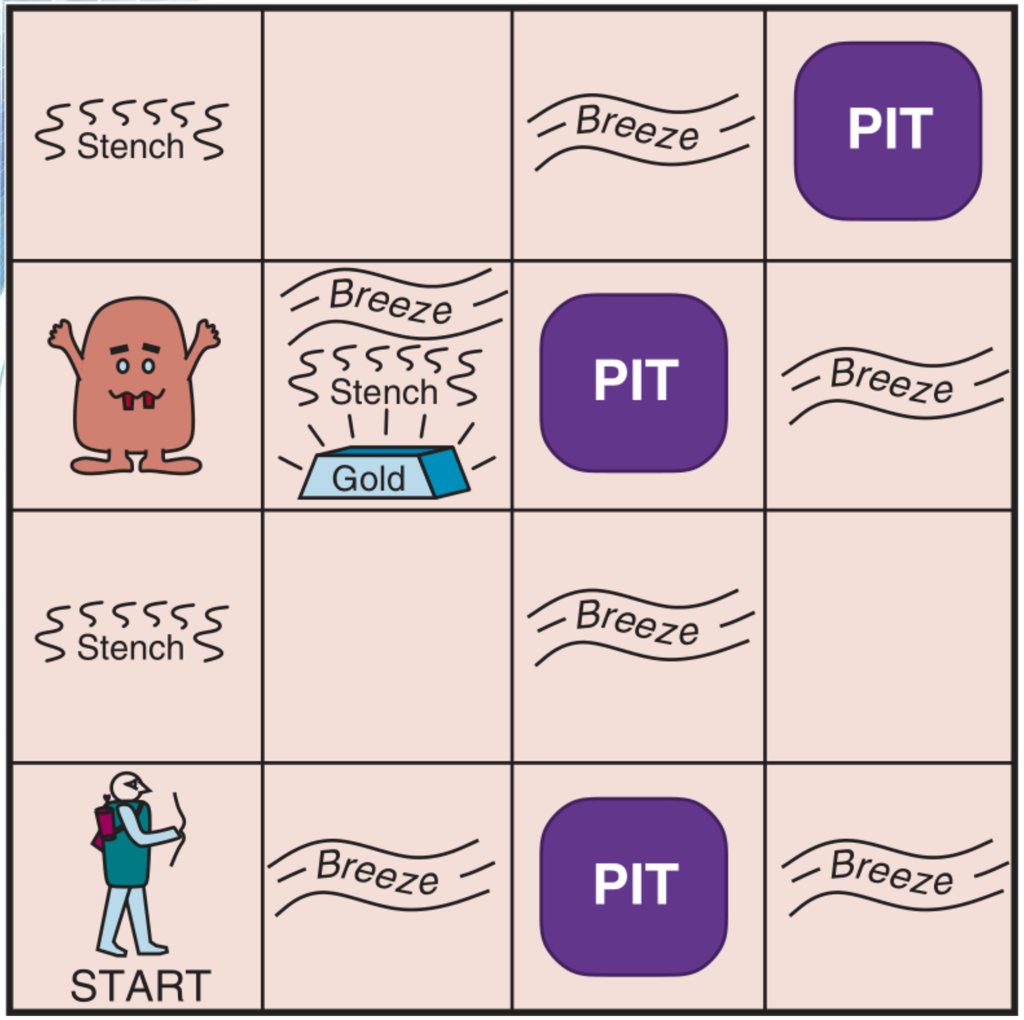
Sensors: *Stench* and *Breeze*: adjacent to the wumpus and a pit, respectively. *Glitter*: square with gold. *Bump*: walks into a wall. *Scream*: the wumpus is killed. Given to the agent as a list of five symbols, ex.: [Stench, Breeze, None, None, None].

Actuators: *Forward*, *TurnLeft* or *TurnRight* (both $90°$), *Grab*, *Shoot* (used only once), *Climb* (only from [1,1]).

Properties: Partially Observable, Static, Discrete, Single-agent, Deterministic, Sequential

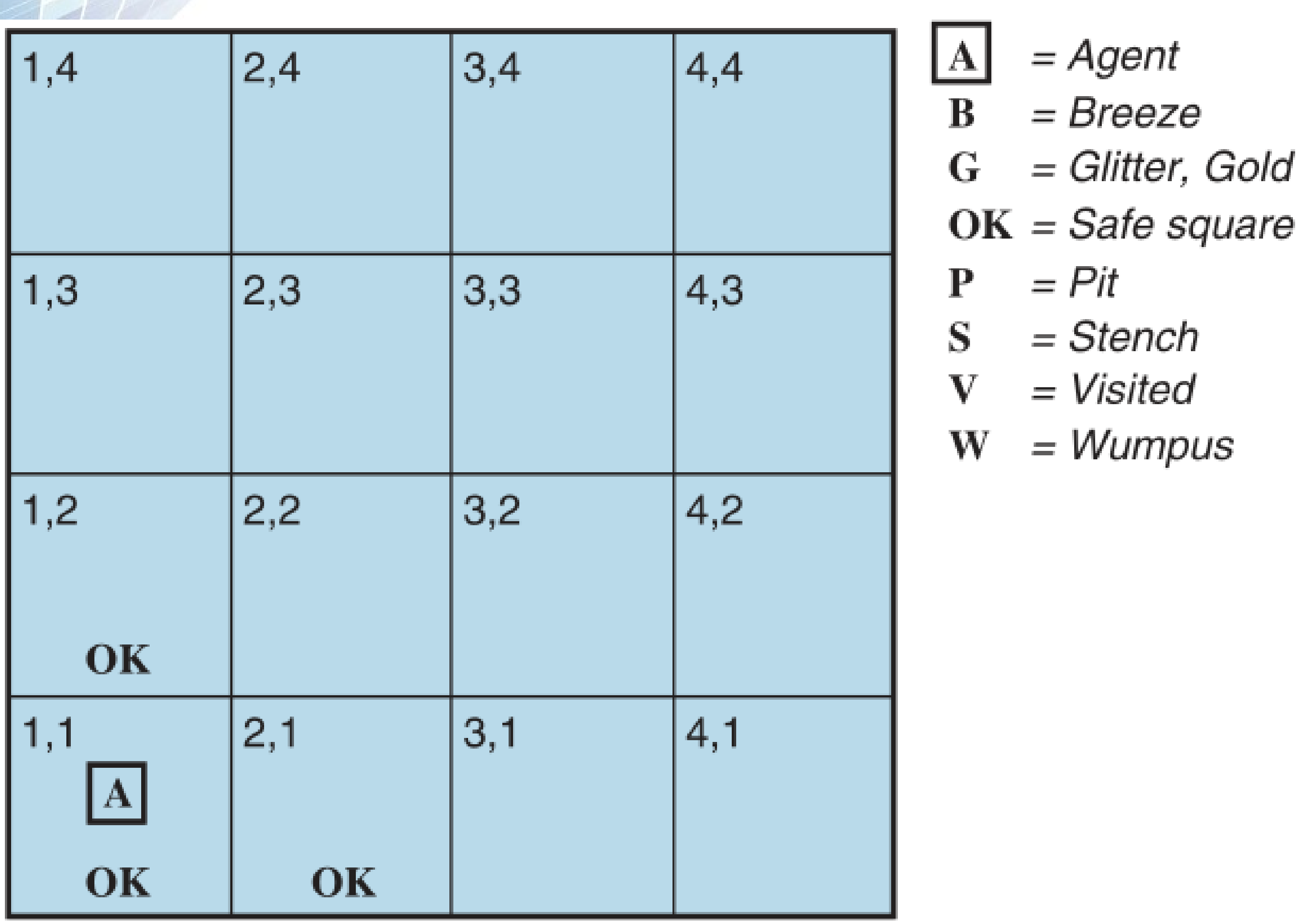
INFORMATIKA UNPAR

Exploring the Wumpus world



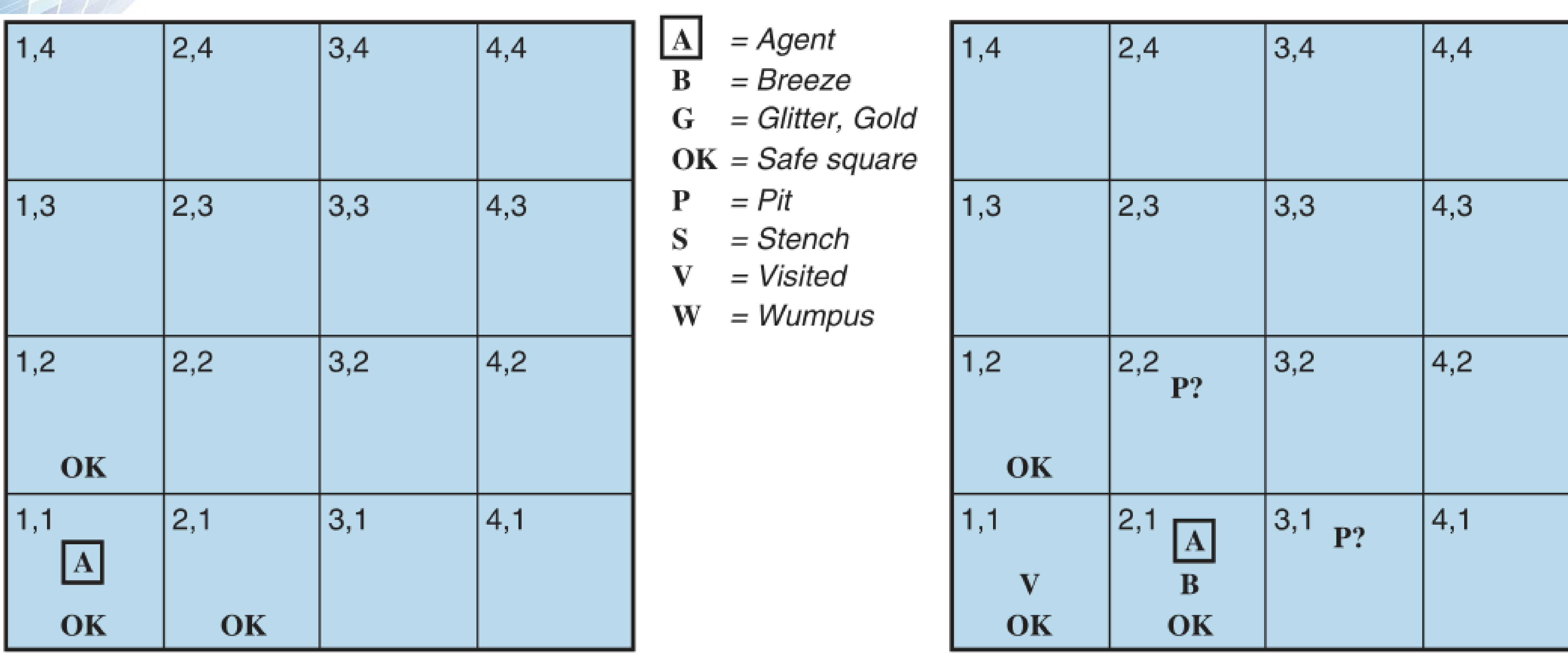
INFORMATIKA UNPAR

Exploring the Wumpus world



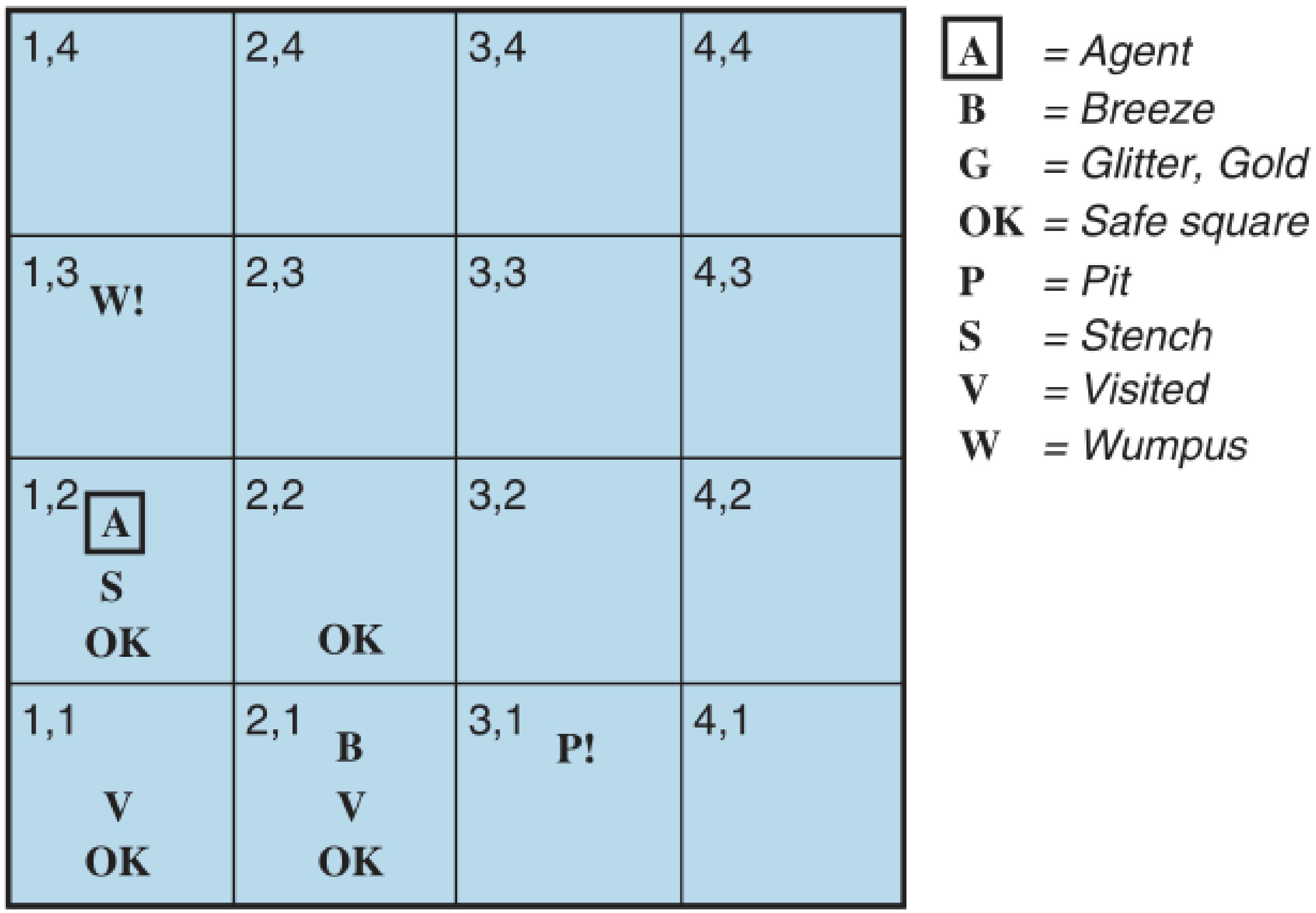
The initial situation: after percept [*None, None, None, None, None*]

Exploring the Wumpus world



After moving to [2,1] and perceiving [*None, Breeze, None, None, None*]

Exploring the Wumpus world



After moving to [1,1] and then [1,2], and perceiving [*Stench, None, None, None, None*]

Exploring the Wumpus world

After moving to [2,2] and then [2,3], and perceiving [*Stench, Breeze, Glitter, None, None*]

Logic

Logics are formal languages for representing knowledge

Logic

Logics are formal languages for representing knowledge

- **Syntax:** defines a well-formed sentence in the language

INFORMATIKA UNPAR

Logic

Logics are formal languages for representing knowledge

- **Syntax:** defines a well-formed sentence in the language
- **Semantic:** defines the meaning of a sentence

Logic

Logics are formal languages for representing knowledge

- **Syntax:** defines a well-formed sentence in the language
- **Semantic:** defines the meaning of a sentence
- **Inference:** rules to derive a new sentence from other sentences

Logic

Logics are formal languages for representing knowledge

- **Syntax**: defines a well-formed sentence in the language
- **Semantic**: defines the meaning of a sentence
- **Inference**: rules to derive a new sentence from other sentences

Logical entailment: *a sentence follows logically from another sentence*

$$\mathcal{KB} \models \boldsymbol{\alpha}$$

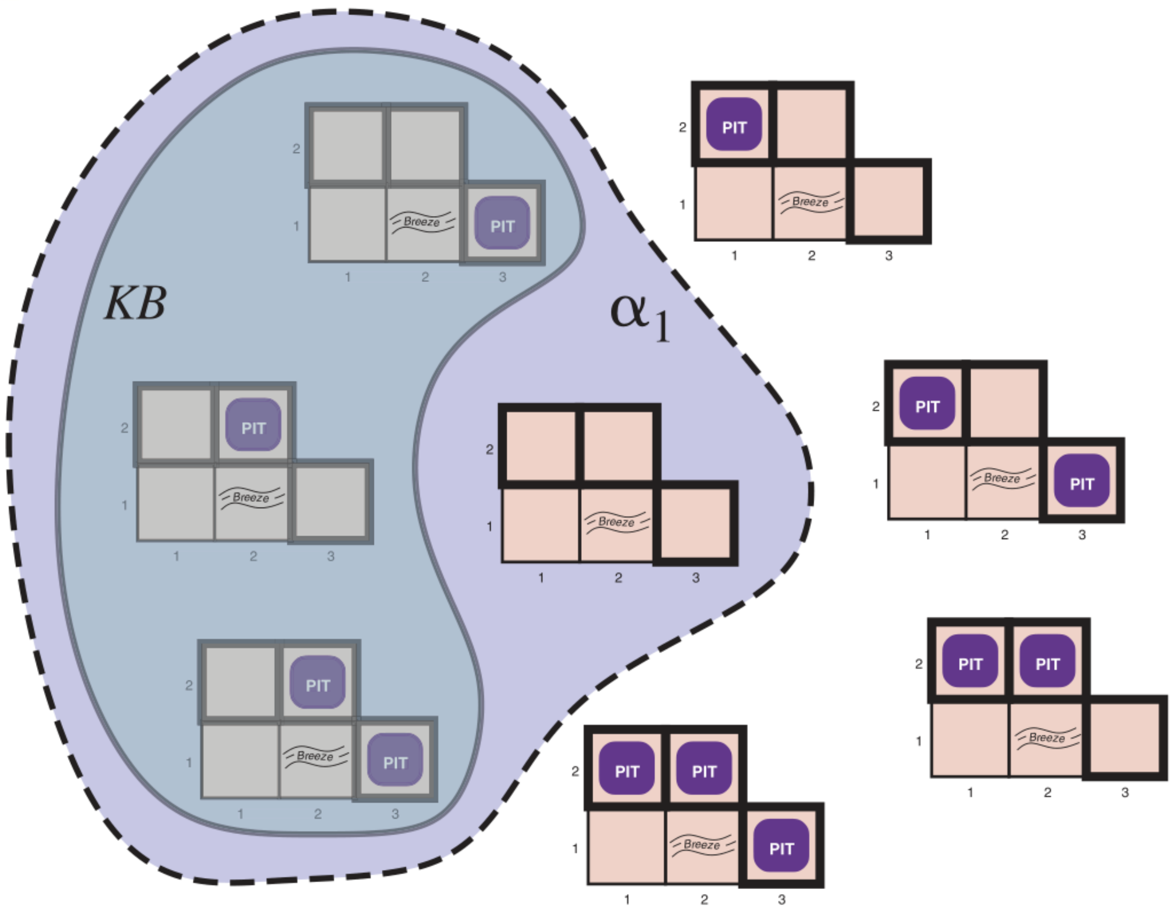
Logic

Logics are formal languages for representing knowledge

- **Syntax:** defines a well-formed sentence in the language
- **Semantic:** defines the meaning of a sentence
- **Inference:** rules to derive a new sentence from other sentences

Logical entailment: *a sentence follows logically from another sentence*

$$\mathcal{KB} \models \alpha$$

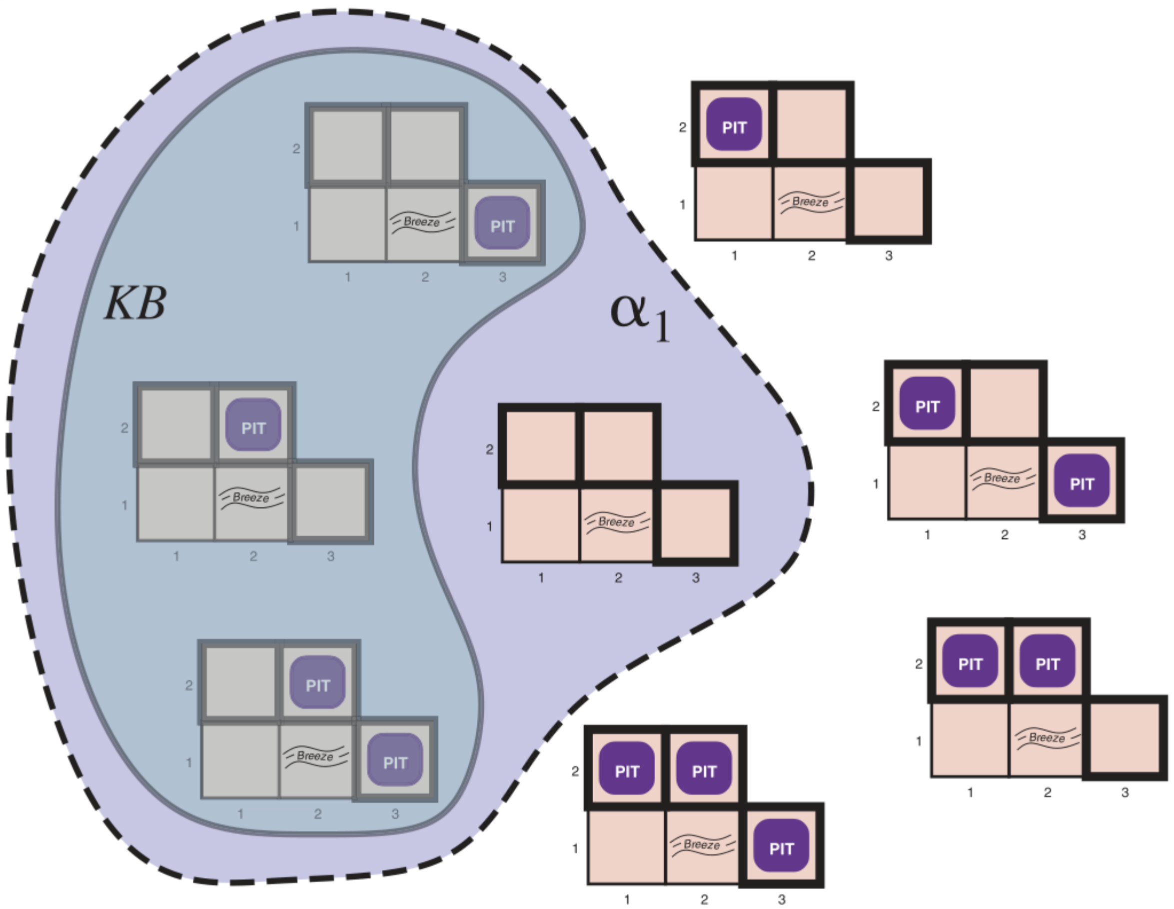


Logics are formal languages for representing knowledge

- **Syntax:** defines a well-formed sentence in the language
- **Semantic:** defines the meaning of a sentence
- **Inference:** rules to derive a new sentence from other sentences

Logical entailment: *a sentence follows logically from another sentence*

$$\mathcal{KB} \models \alpha$$



- $\mathcal{KB}$: nothing in [1,1] and a breeze in [2,1] (from agent's percept)

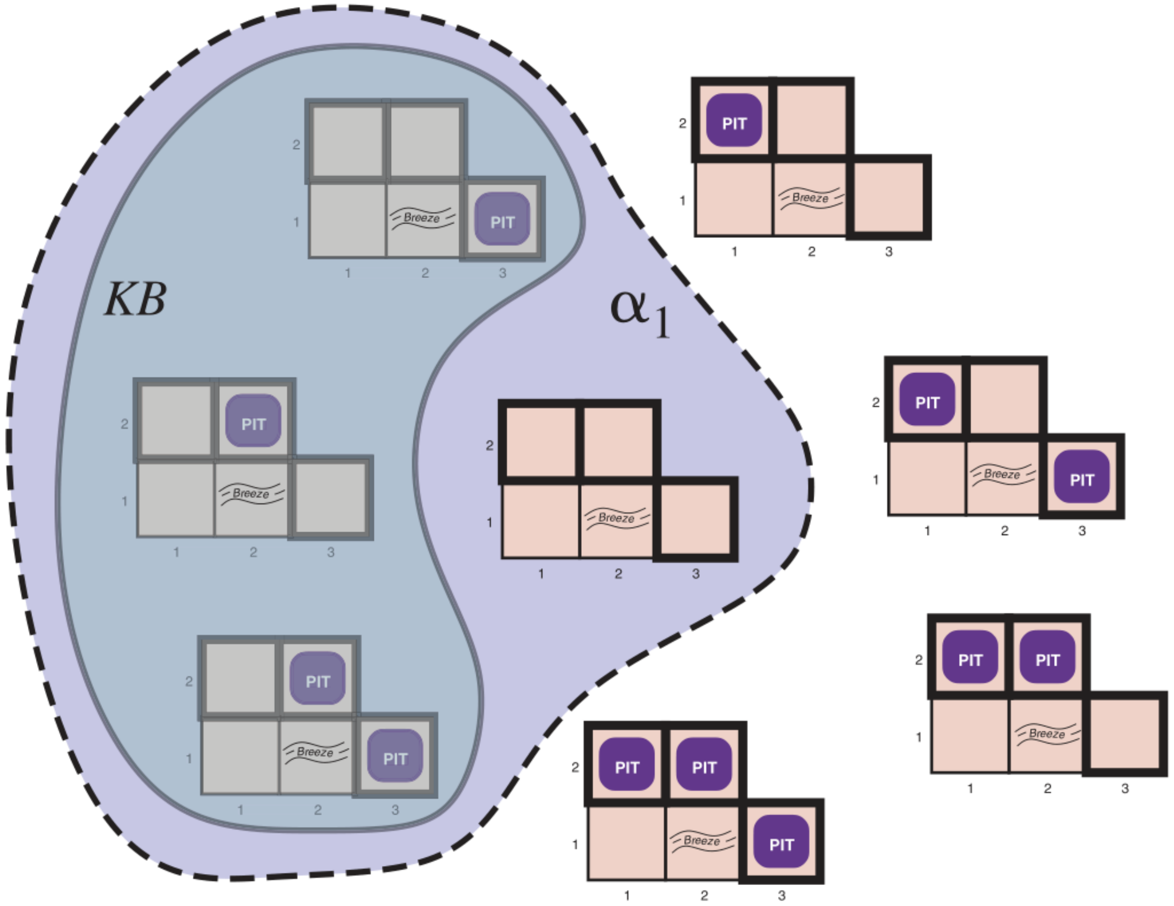
Logic

Logics are formal languages for representing knowledge

- **Syntax:** defines a well-formed sentence in the language
- **Semantic:** defines the meaning of a sentence
- **Inference:** rules to derive a new sentence from other sentences

Logical entailment: *a sentence follows logically from another sentence*

$$\mathcal{KB} \models \alpha$$



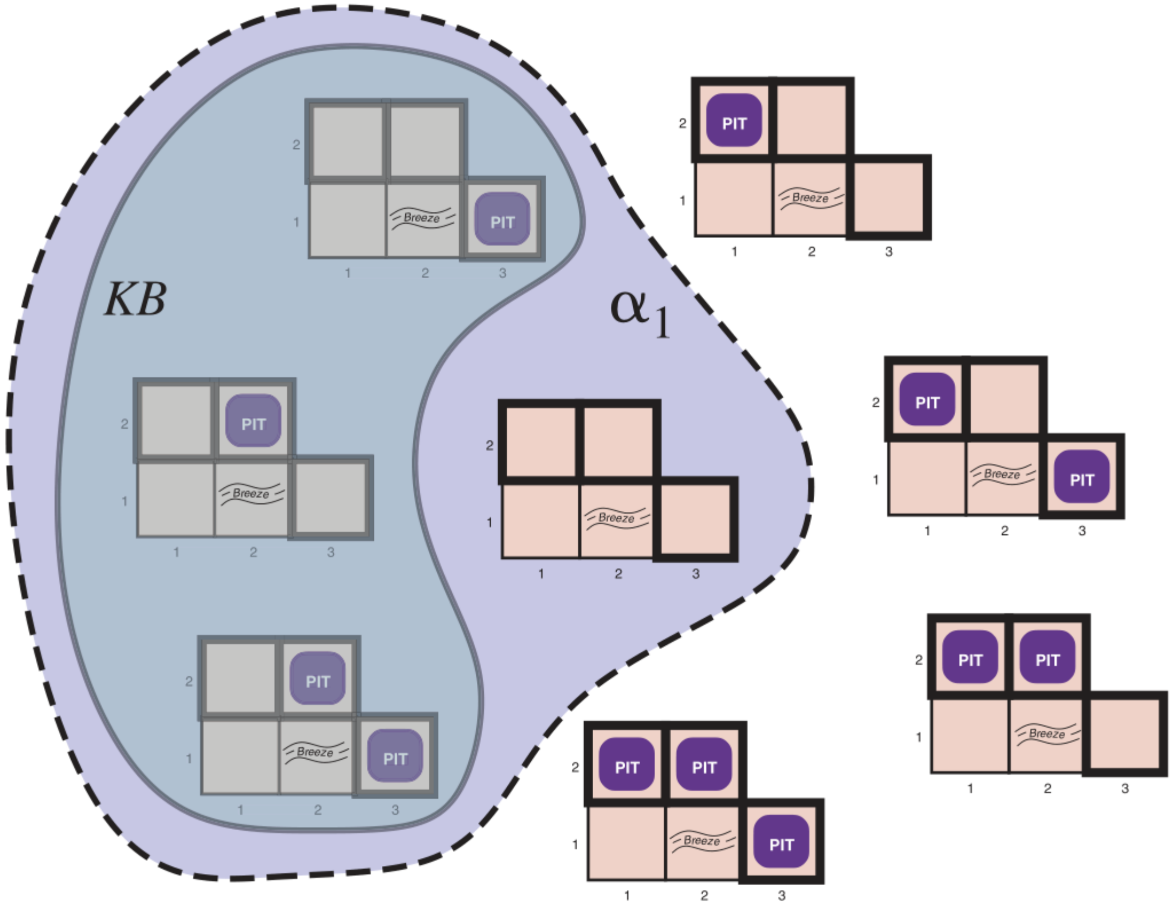
- $\mathcal{KB}$: nothing in [1,1] and a breeze in [2,1] (from agent's percept)
- $\mathcal{KB}$ is false where [1,2] contains a pit. There is no breeze in [1,1].

Logics are formal languages for representing knowledge

- **Syntax:** defines a well-formed sentence in the language
- **Semantic:** defines the meaning of a sentence
- **Inference:** rules to derive a new sentence from other sentences

Logical entailment: *a sentence follows logically from another sentence*

$$\mathcal{KB} \models \boldsymbol{\alpha}$$



- $\mathcal{KB}$: nothing in [1,1] and a breeze in [2,1] (from agent's percept)
- $\mathcal{KB}$ is false where [1,2] contains a pit. There is no breeze in [1,1].
- α_1 = "There is no pit in [1,2]"

INFORMATIKA UNPAR

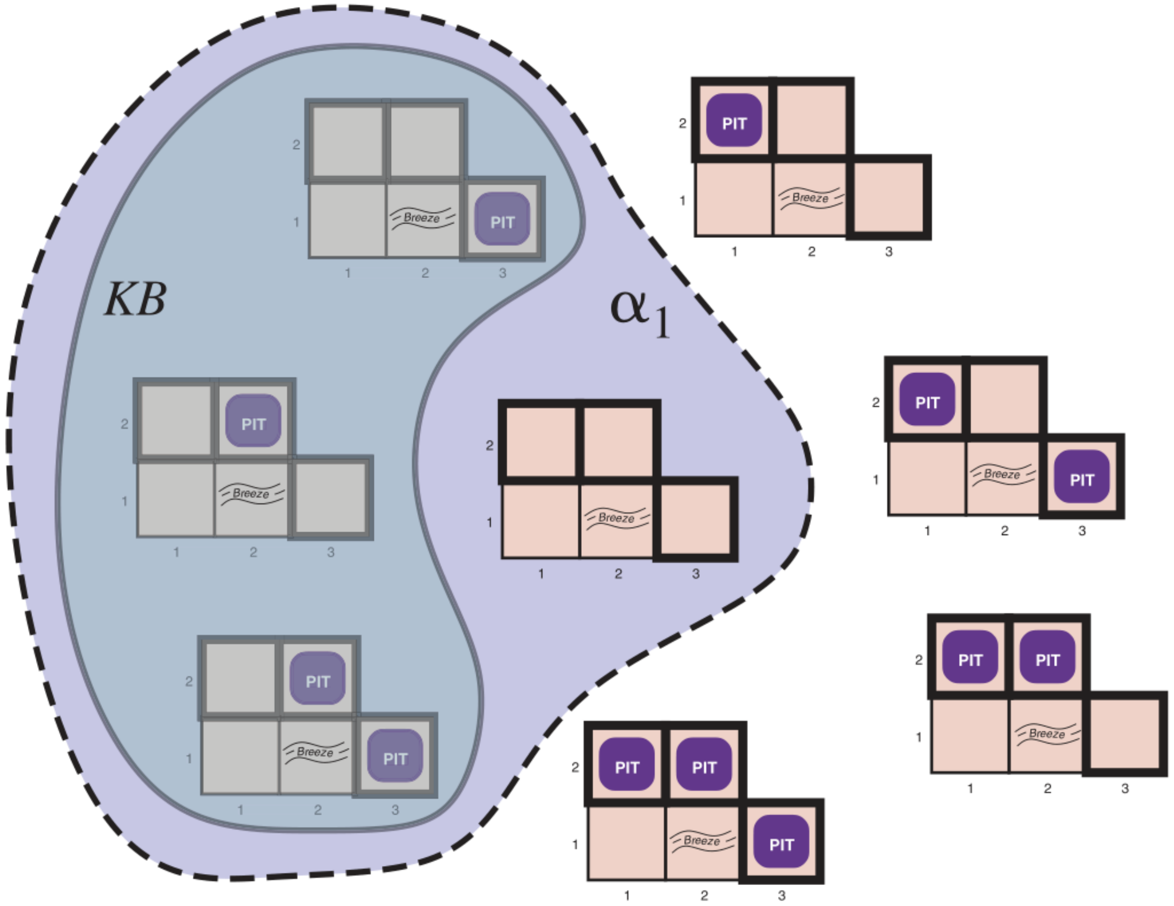
Logic

Logics are formal languages for representing knowledge

- **Syntax:** defines a well-formed sentence in the language
- **Semantic:** defines the meaning of a sentence
- **Inference:** rules to derive a new sentence from other sentences

Logical entailment: *a sentence follows logically from another sentence*

$$\mathcal{KB} \vDash \boldsymbol{\alpha}$$



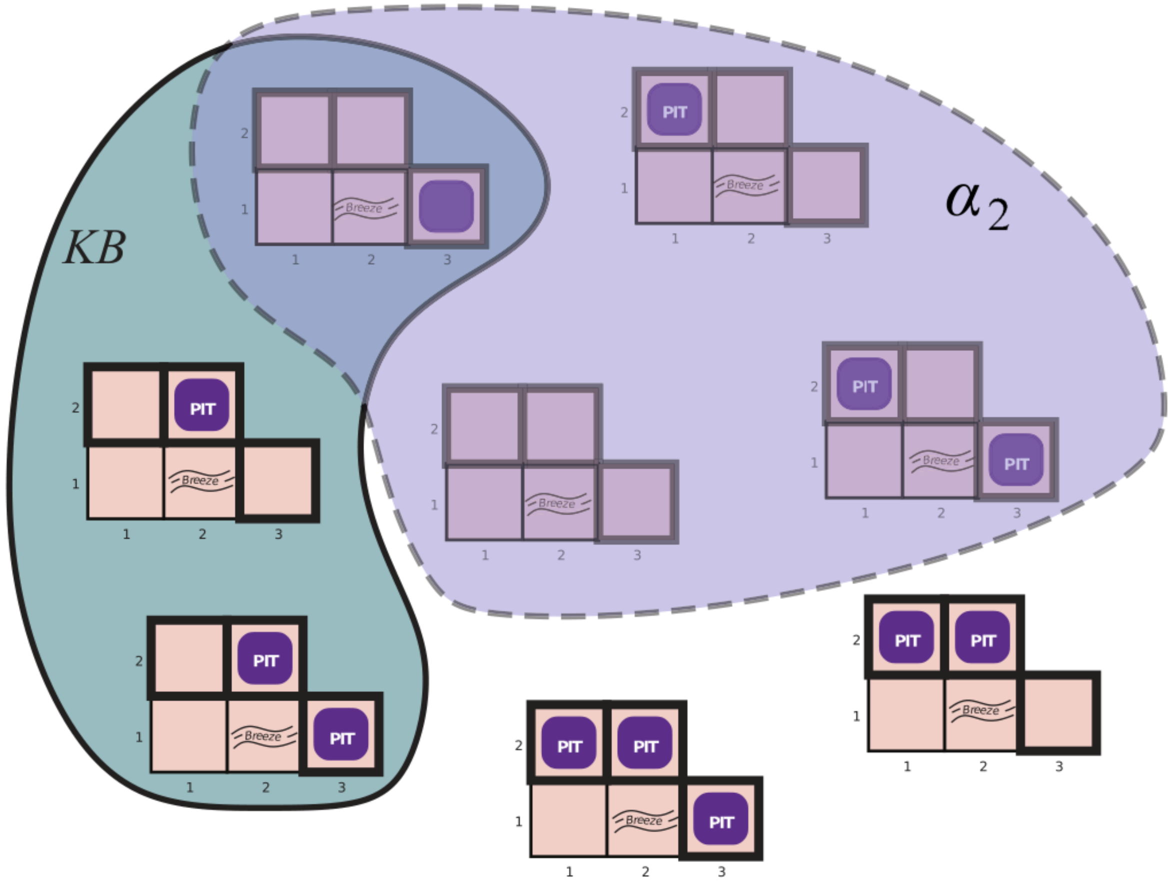
- $\mathcal{KB}$: nothing in [1,1] and a breeze in [2,1] (from agent's percept)
- $\mathcal{KB}$ is false where [1,2] contains a pit. There is no breeze in [1,1].
- α_1 = "There is no pit in [1,2]"
- in every model in which $\mathcal{KB}$ is true, so does α_1
- hence, $\mathcal{KB} \vDash \alpha_1$

Logics are formal languages for representing knowledge

- **Syntax:** defines a well-formed sentence in the language
- **Semantic:** defines the meaning of a sentence
- **Inference:** rules to derive a new sentence from other sentences

Logical entailment: *a sentence follows logically from another sentence*

$$\mathcal{KB} \models \boldsymbol{\alpha}$$



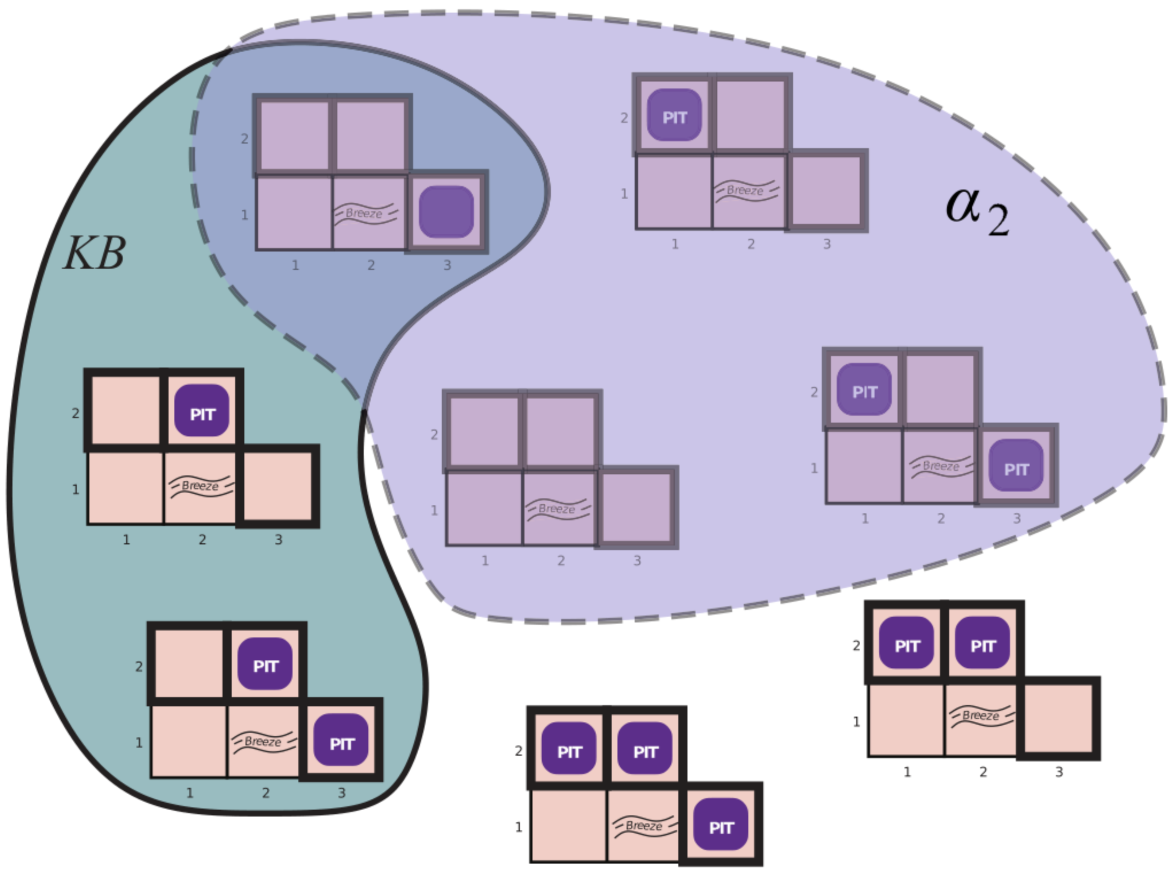
Logic

Logics are formal languages for representing knowledge

- **Syntax:** defines a well-formed sentence in the language
- **Semantic:** defines the meaning of a sentence
- **Inference:** rules to derive a new sentence from other sentences

Logical entailment: *a sentence follows logically from another sentence*

$$\mathcal{KB} \models \boldsymbol{\alpha}$$



- α_2 = "There is no pit in [2,2]"

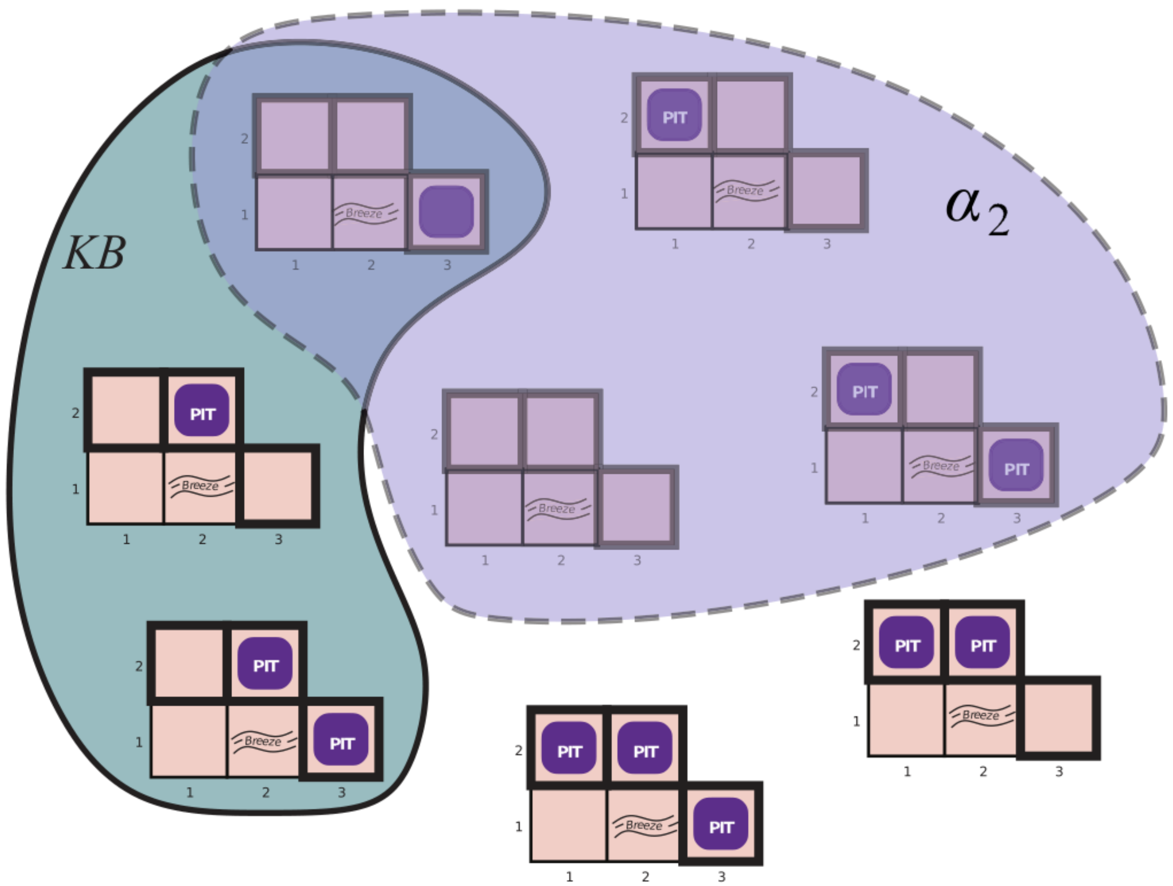
Logic

Logics are formal languages for representing knowledge

- **Syntax**: defines a well-formed sentence in the language
- **Semantic**: defines the meaning of a sentence
- **Inference**: rules to derive a new sentence from other sentences

Logical entailment: *a sentence follows logically from another sentence*

$$\mathcal{KB} \models \boldsymbol{\alpha}$$



- α_2 = "There is no pit in [2,2]"
- in some model in which $\mathcal{KB}$ is true, α_2 is false

INFORMATIKA UNPAR

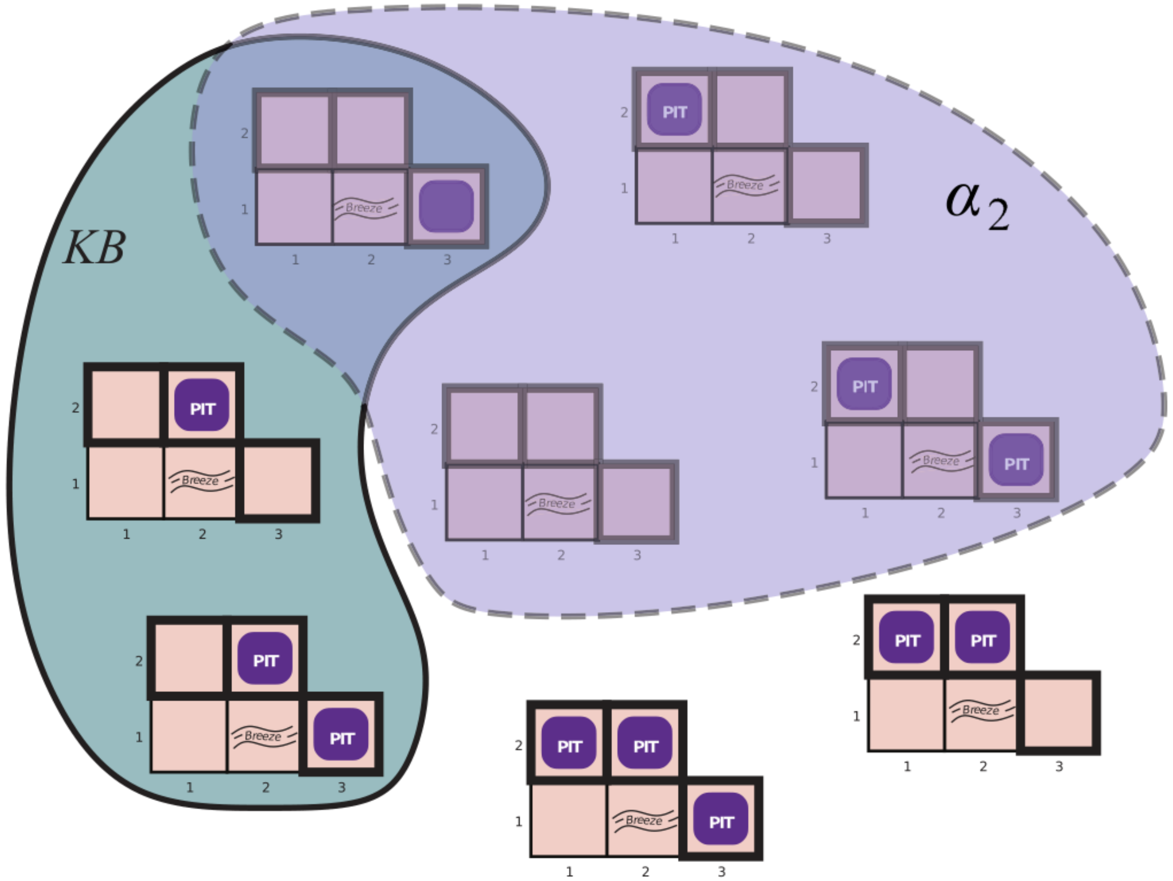
Logic

Logics are formal languages for representing knowledge

- **Syntax:** defines a well-formed sentence in the language
- **Semantic:** defines the meaning of a sentence
- **Inference:** rules to derive a new sentence from other sentences

Logical entailment: *a sentence follows logically from another sentence*

$$\mathcal{KB} \models \boldsymbol{\alpha}$$



- α_2 = "There is no pit in [2,2]"
- in some model in which $\mathcal{KB}$ is true, α_2 is false
- hence, $\mathcal{KB}$ does not entail α_2

Logical inference is performed using

Logical Inference

Logical inference is performed using

- **Model Checking**: entailment through semantics, enumerate all models and show that the sentence α must hold in all models, $\mathcal{KB} \models \alpha$; or

INFORMATIKA UNPAR

Logical Inference

Logical inference is performed using

- **Model Checking**: entailment through semantics, enumerate all models and show that the sentence α must hold in all models, $\mathcal{KB} \models \alpha$; or
- **Theorem Proving**: entailment through syntax, apply rules of inference to $\mathcal{KB}$ to build a proof of α without enumerating and checking all models; $\mathcal{KB} \vdash \alpha$

$\mathcal{KB} \vdash_i \alpha$ denotes inference algorithm i derives α from $\mathcal{KB}$

INFORMATIKA UNPAR

Logical Inference

Logical inference is performed using

- **Model Checking**: entailment through semantics, enumerate all models and show that the sentence α must hold in all models, $\mathcal{KB} \vDash \alpha$; or
- **Theorem Proving**: entailment through syntax, apply rules of inference to $\mathcal{KB}$ to build a proof of α without enumerating and checking all models; $\mathcal{KB} \vdash \alpha$

  $\mathcal{KB} \vdash_i \alpha$ denotes inference algorithm i derives α from $\mathcal{KB}$

Inference algorithms should hold these two properties:

INFORMATIKA UNPAR

Logical inference is performed using

- **Model Checking**: entailment through semantics, enumerate all models and show that the sentence α must hold in all models, $\mathcal{KB} \models \alpha$; or
- **Theorem Proving**: entailment through syntax, apply rules of inference to $\mathcal{KB}$ to build a proof of α without enumerating and checking all models; $\mathcal{KB} \vdash \alpha$

  $\mathcal{KB} \vdash_i \alpha$ denotes inference algorithm i derives α from $\mathcal{KB}$

Inference algorithms should hold these two properties:

- **Sound**(logically valid): derives only entailed sentences, does not infer false formula

$$\{\boldsymbol{\alpha} | \mathcal{KB} \vdash \boldsymbol{\alpha}\} \subseteq \{\mathcal{KB} \models \boldsymbol{\alpha}\}$$

Logical Inference

Logical inference is performed using

- **Model Checking**: entailment through semantics, enumerate all models and show that the sentence α must hold in all models, $\mathcal{KB} \vDash \alpha$; or
- **Theorem Proving**: entailment through syntax, apply rules of inference to $\mathcal{KB}$ to build a proof of α without enumerating and checking all models; $\mathcal{KB} \vdash \alpha$
  $\mathcal{KB} \vdash_i \alpha$ denotes inference algorithm i derives α from $\mathcal{KB}$

Inference algorithms should hold these two properties:

- **Sound**(logically valid): derives only entailed sentences, does not infer false formula
$$\{\boldsymbol{\alpha} | \mathcal{KB} \vdash \boldsymbol{\alpha}\} \subseteq \{\mathcal{KB} \vDash \boldsymbol{\alpha}\}$$
- **Complete**: derives all entailed sentences
$$\{\boldsymbol{\alpha} | \mathcal{KB} \vdash \boldsymbol{\alpha}\} \supseteq \{\mathcal{KB} \vDash \boldsymbol{\alpha}\}$$

INFORMATIKA
UNPAR

Logical Inference

Logical inference is performed using

- **Model Checking**: entailment through semantics, enumerate all models and show that the sentence α must hold in all models, $\mathcal{KB} \vDash \alpha$; or
- **Theorem Proving**: entailment through syntax, apply rules of inference to $\mathcal{KB}$ to build a proof of α without enumerating and checking all models; $\mathcal{KB} \vdash \alpha$
  $\mathcal{KB} \vdash_i \alpha$ denotes inference algorithm i derives α from $\mathcal{KB}$

Inference algorithms should hold these two properties:

- **Sound**(logically valid): derives only entailed sentences, does not infer false formula
  $$\{\boldsymbol{\alpha} | \mathcal{KB} \vdash \boldsymbol{\alpha}\} \subseteq \{\mathcal{KB} \vDash \boldsymbol{\alpha}\}$$
- **Complete**: derives all entailed sentences
  $$\{\boldsymbol{\alpha} | \mathcal{KB} \vdash \boldsymbol{\alpha}\} \supseteq \{\mathcal{KB} \vDash \boldsymbol{\alpha}\}$$

The basic rules (background knowledge) can be produced from *learning*

Propositional Logic

..... is the simplest logic

Propositional Logic

..... is the simplest logic

Syntax of PL: defines the allowable sentences or propositions

Propositional Logic

..... is the simplest logic

Syntax of PL: defines the allowable sentences or propositions

Sentence/Definition/Proposition: a declarative statement, either TRUE OR FALSE

Propositional Logic

..... is the simplest logic

Syntax of PL: defines the allowable sentences or propositions

Sentence/Definition/Proposition: a declarative statement, either TRUE OR FALSE

Atomic sentence: a single sentence symbol

Propositional Logic

..... is the simplest logic

Syntax of PL: defines the allowable sentences or propositions

Sentence/Definition/Proposition: a declarative statement, either TRUE OR FALSE

Atomic sentence: a single sentence symbol

Compound sentence: formed from atomic sentence, parentheses, & logical connectives

Propositional Logic

..... is the simplest logic

Syntax of PL: defines the allowable sentences or propositions

Sentence/Definition/Proposition: a declarative statement, either TRUE OR FALSE

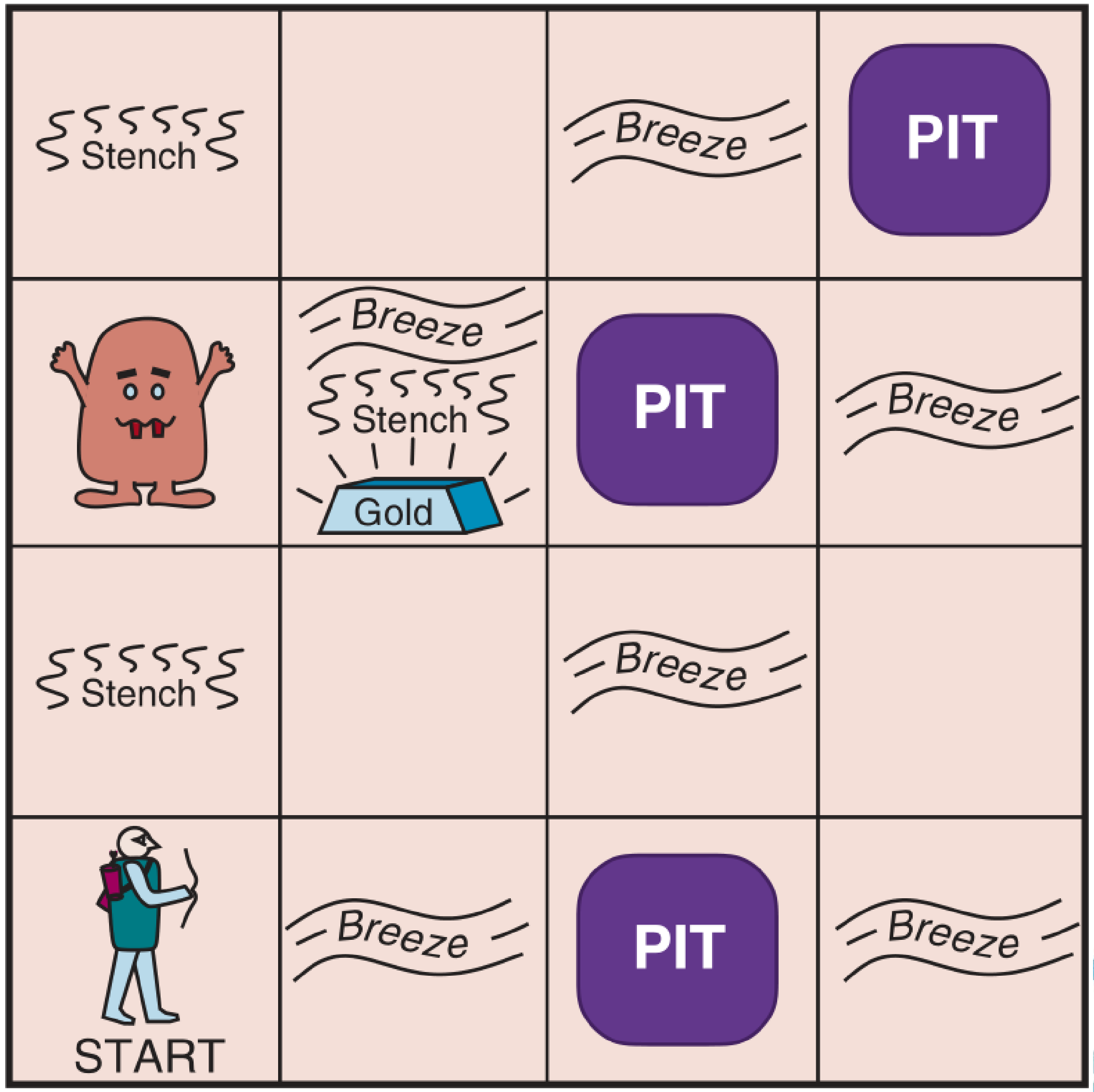
Atomic sentence: a single sentence symbol

Compound sentence: formed from atomic sentence, parentheses, & logical connectives

A BNF (Backus-Naur Form) grammar of sentences in propositional logic:

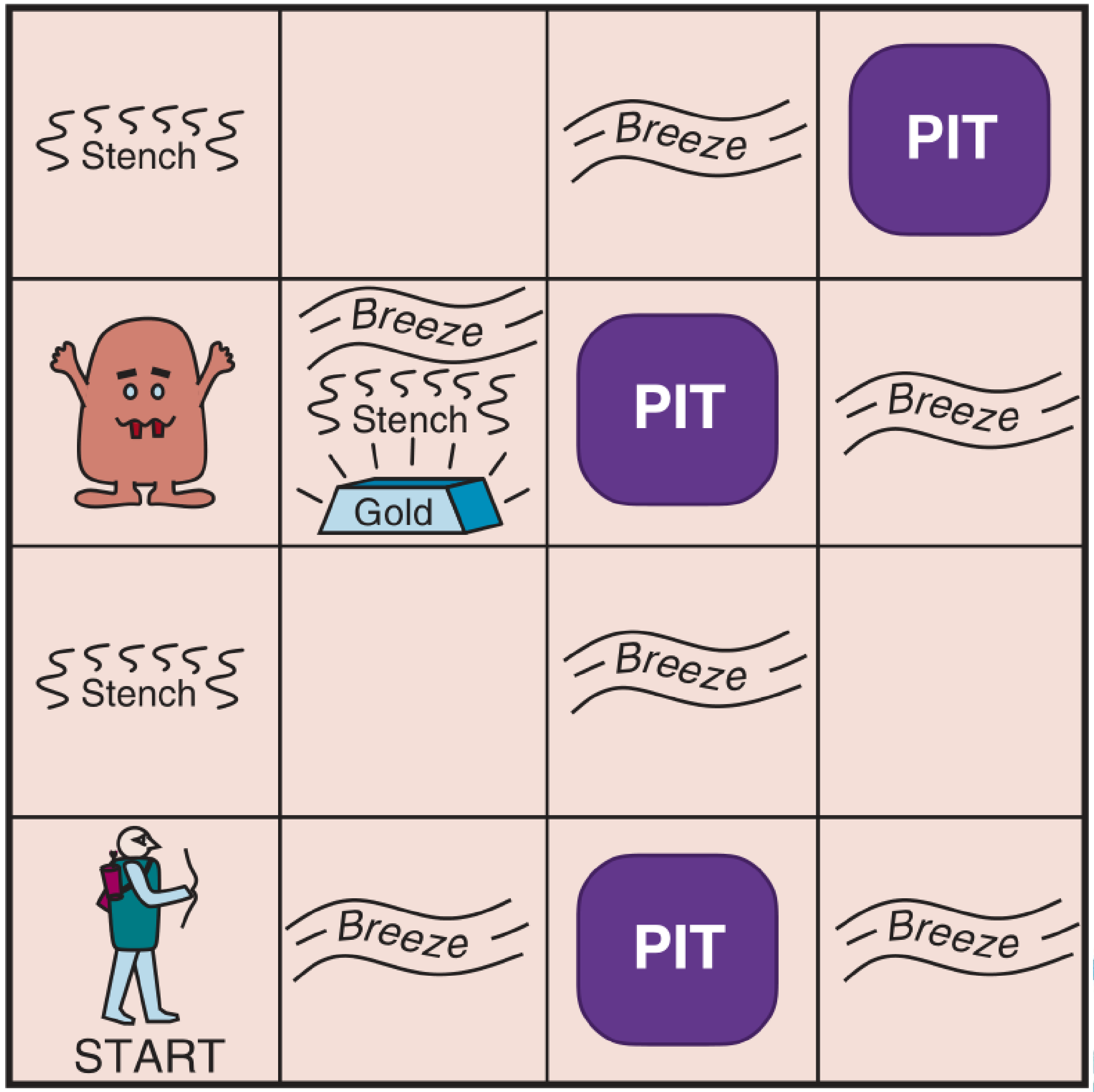
$$
\begin{array}{rcl}
Sentence & \rightarrow & AtomicSentence \mid ComplexSentence \\
AtomicSentence & \rightarrow & True \mid False \mid P \mid Q \mid R \mid \ldots \\
ComplexSentence & \rightarrow & (\ Sentence\ ) \\
 & \mid & \neg\ Sentence \\
 & \mid & Sentence \land Sentence \\
 & \mid & Sentence \lor Sentence \\
 & \mid & Sentence \Rightarrow Sentence \\
 & \mid & Sentence \Leftrightarrow Sentence \\
 & & \\
\text{OPERATOR PRECEDENCE} & : & \neg, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}
$$

The Wumpus world $\mathcal{KB}$



The Wumpus world $\mathcal{KB}$

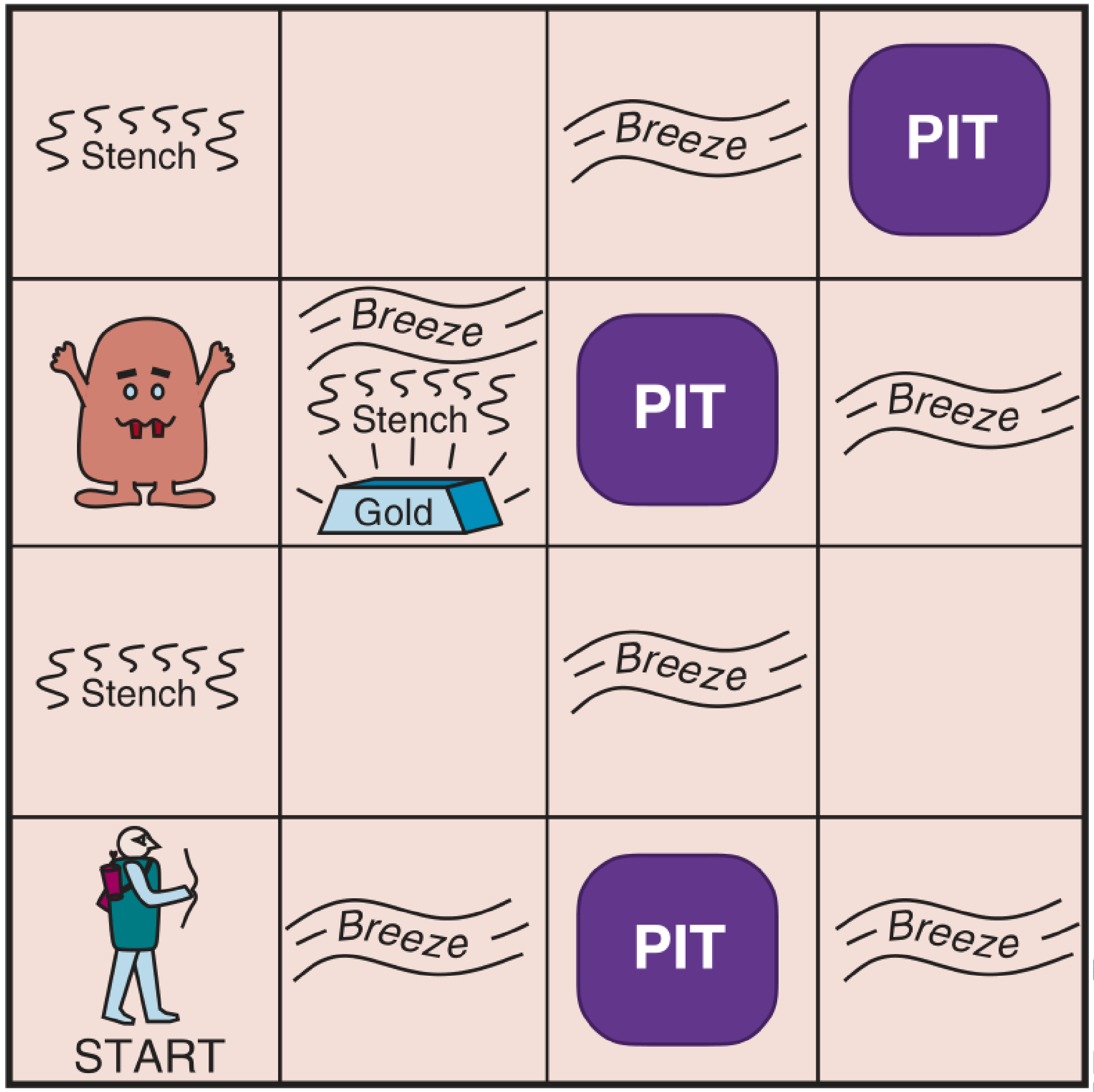
Symbols for each $[x, y]$:



The Wumpus world $\mathcal{KB}$

Symbols for each $[x, y]$:

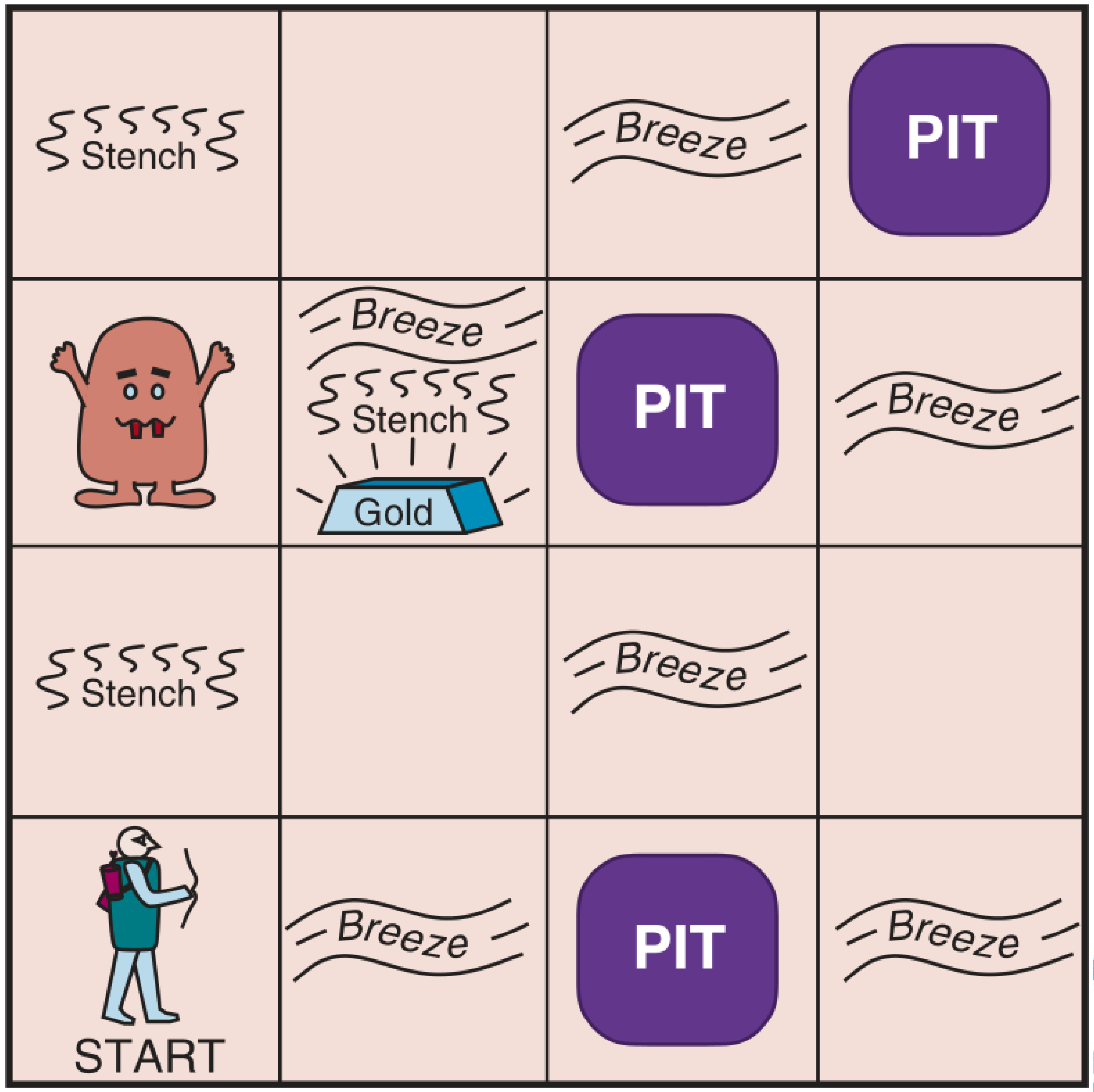
- $P_{i,j}$ is true if there is a pit in $[i, j]$



The Wumpus world $\mathcal{KB}$

Symbols for each $[x, y]$:

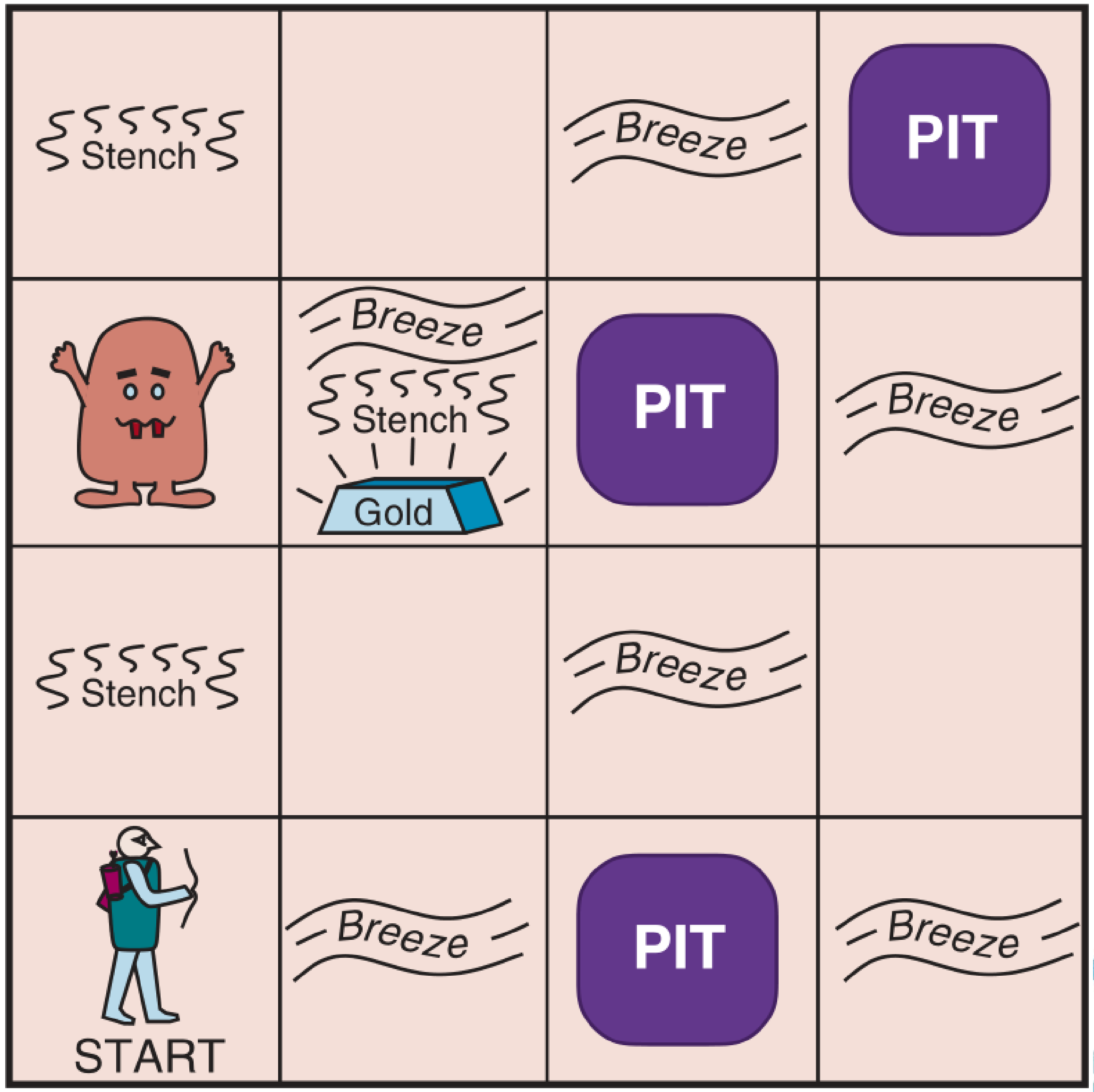
- $P_{i,j}$ is true if there is a pit in $[i, j]$
- $B_{i,j}$ is true if there is a breeze in $[i, j]$



The Wumpus world $\mathcal{KB}$

Symbols for each $[x, y]$:

- $P_{i,j}$ is true if there is a pit in $[i, j]$
- $B_{i,j}$ is true if there is a breeze in $[i, j]$
- and the same for W (wumpus), S (stench), and L (agent's location)



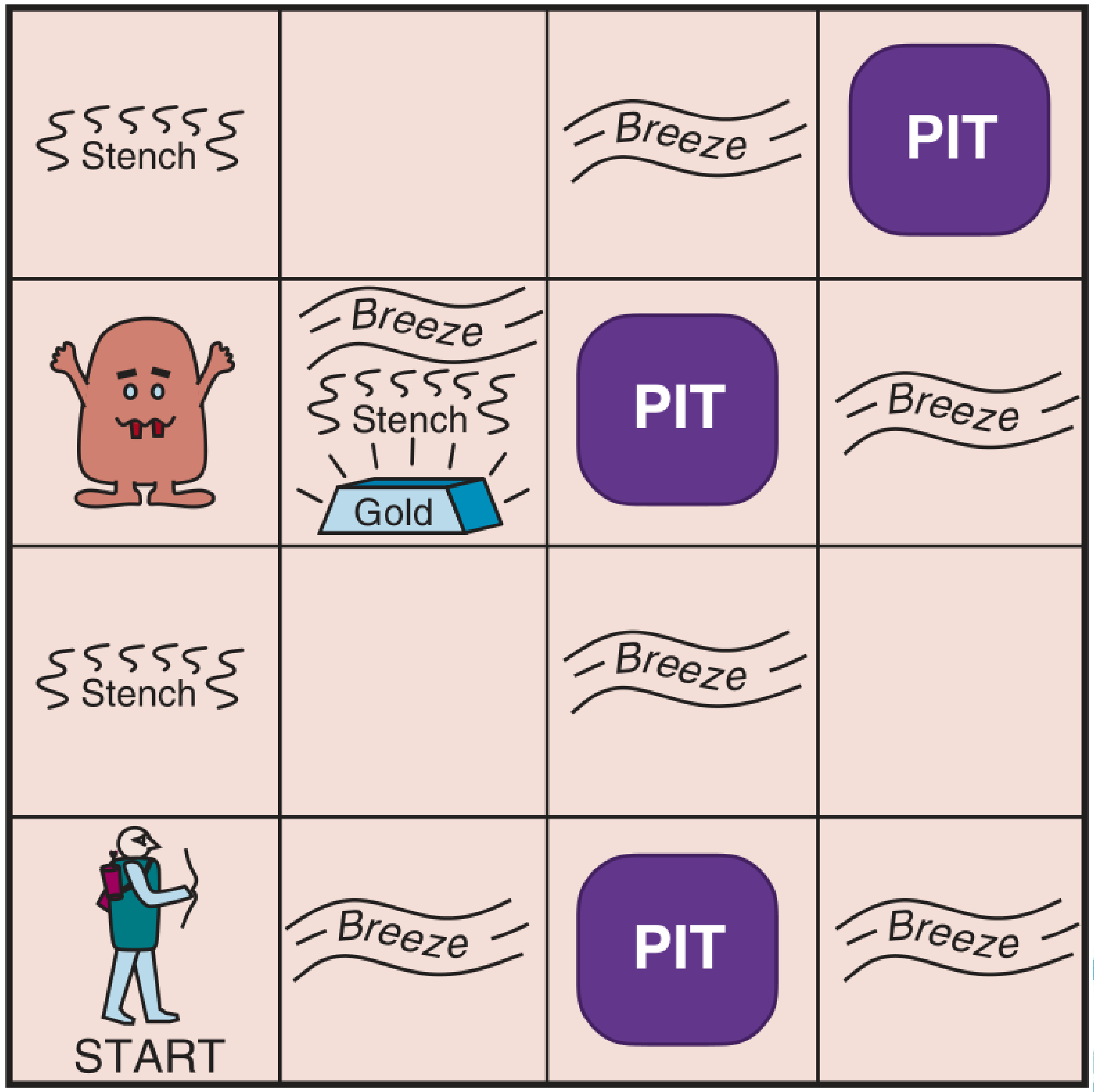
INFORMATIKA UNPAR

The Wumpus world $\mathcal{KB}$

Symbols for each $[x, y]$:

- $P_{i,j}$ is true if there is a pit in $[i, j]$
- $B_{i,j}$ is true if there is a breeze in $[i, j]$
- and the same for W (wumpus), S (stench), and L (agent's location)

$\mathcal{KB}$ for the *reduced* Wumpus world:



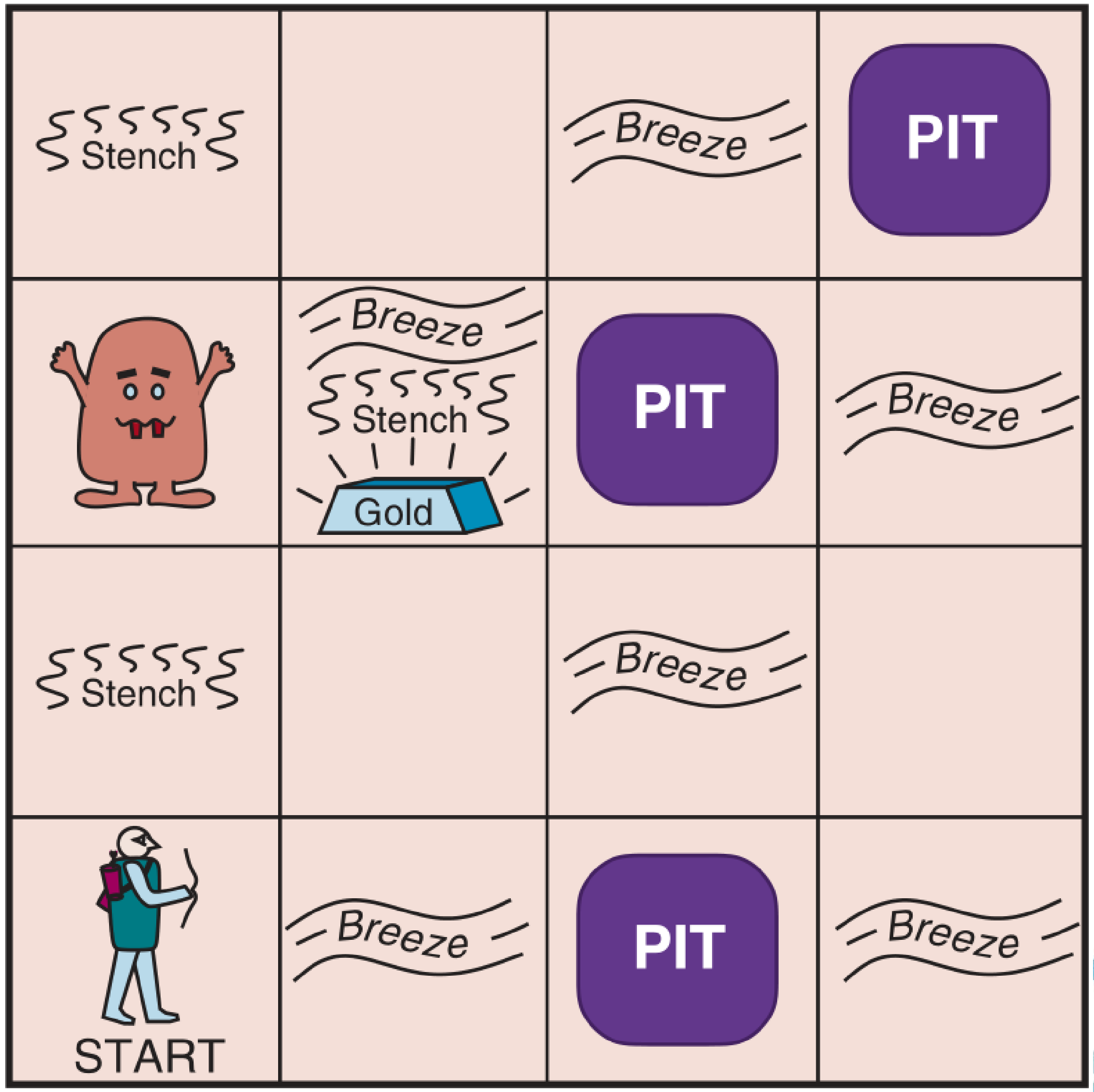
The Wumpus world $\mathcal{KB}$

Symbols for each $[x, y]$:

- $P_{i,j}$ is true if there is a pit in $[i, j]$
- $B_{i,j}$ is true if there is a breeze in $[i, j]$
- and the same for W (wumpus), S (stench), and L (agent's location)

$\mathcal{KB}$ for the *reduced* Wumpus world:

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



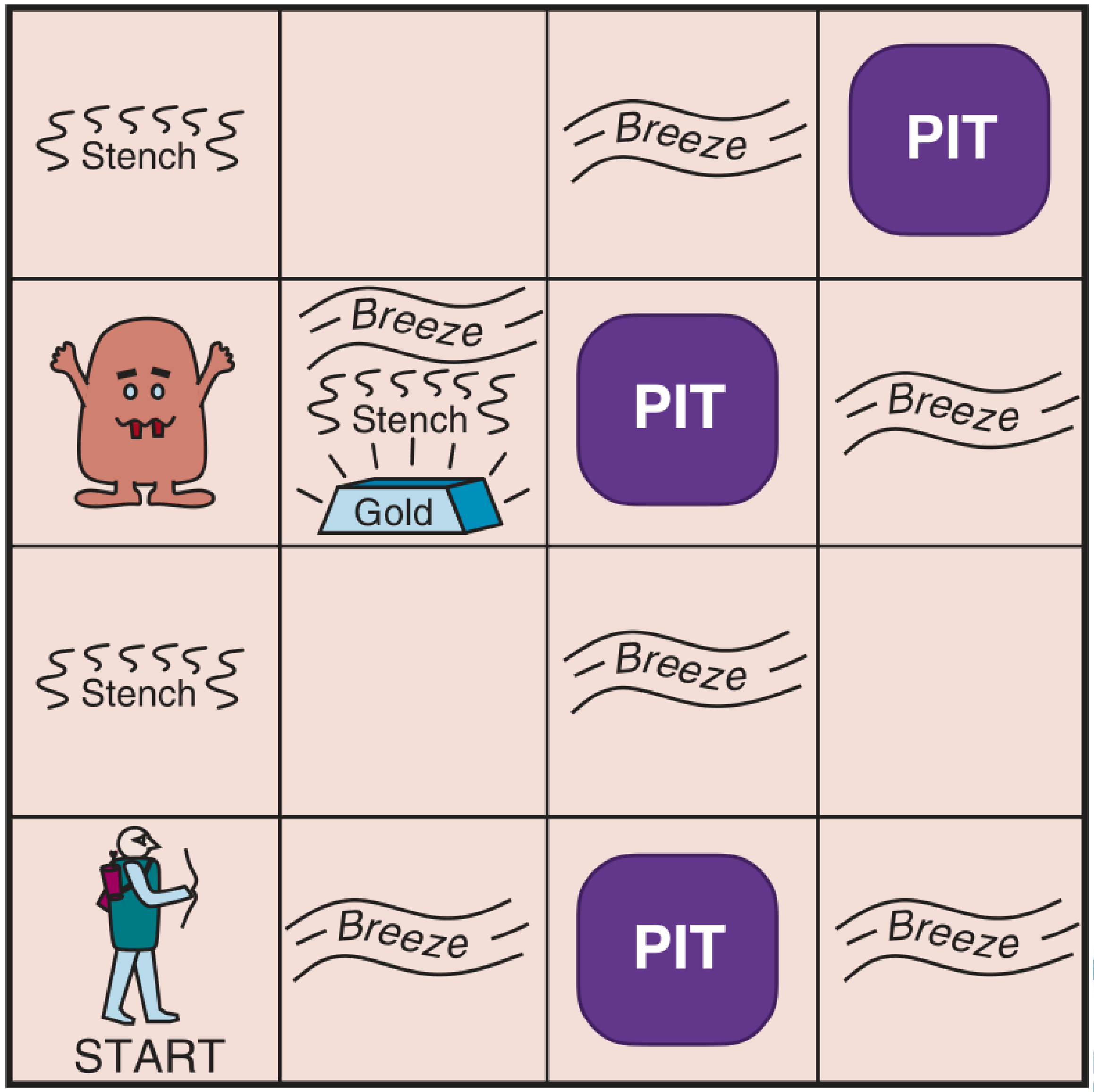
The Wumpus world $\mathcal{KB}$

Symbols for each $[x, y]$:

- $P_{i,j}$ is true if there is a pit in $[i, j]$
- $B_{i,j}$ is true if there is a breeze in $[i, j]$
- and the same for W (wumpus), S (stench), and L (agent's location)

$\mathcal{KB}$ for the *reduced* Wumpus world:

- $R_1 : \neg P_{1,1}$
- $R_2 : B_{1,1} \iff P_{1,2} \vee P_{2,1}$



INFORMATIKA UNPAR

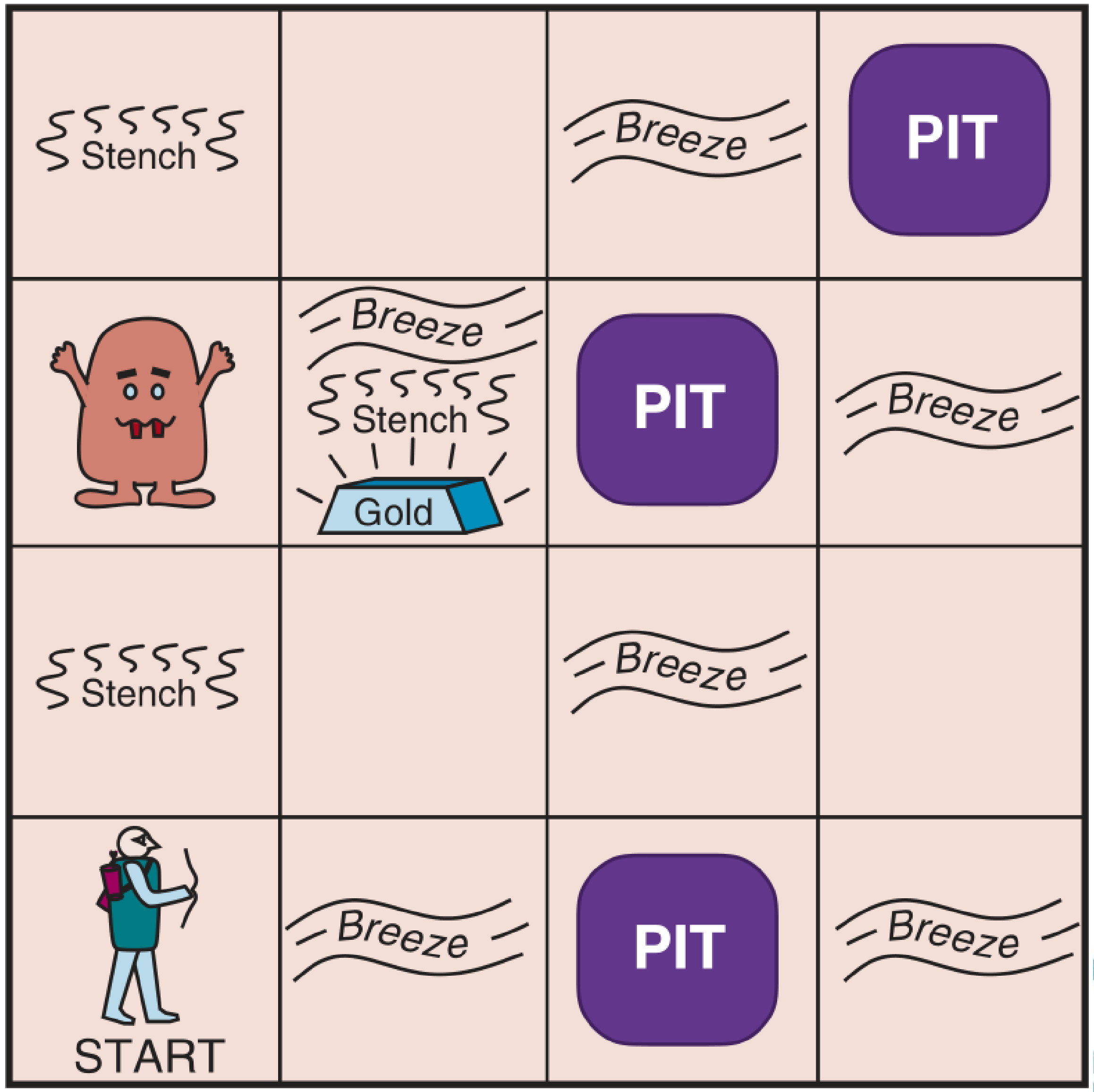
The Wumpus world $\mathcal{KB}$

Symbols for each $[x, y]$:

- $P_{i,j}$ is true if there is a pit in $[i, j]$
- $B_{i,j}$ is true if there is a breeze in $[i, j]$
- and the same for W (wumpus), S (stench), and L (agent's location)

$\mathcal{KB}$ for the *reduced* Wumpus world:

- $R_1 : \neg P_{1,1}$
- $R_2 : B_{1,1} \iff P_{1,2} \vee P_{2,1}$
- $R_3 : B_{2,1} \iff P_{1,1} \vee P_{2,2} \vee P_{3,1}$



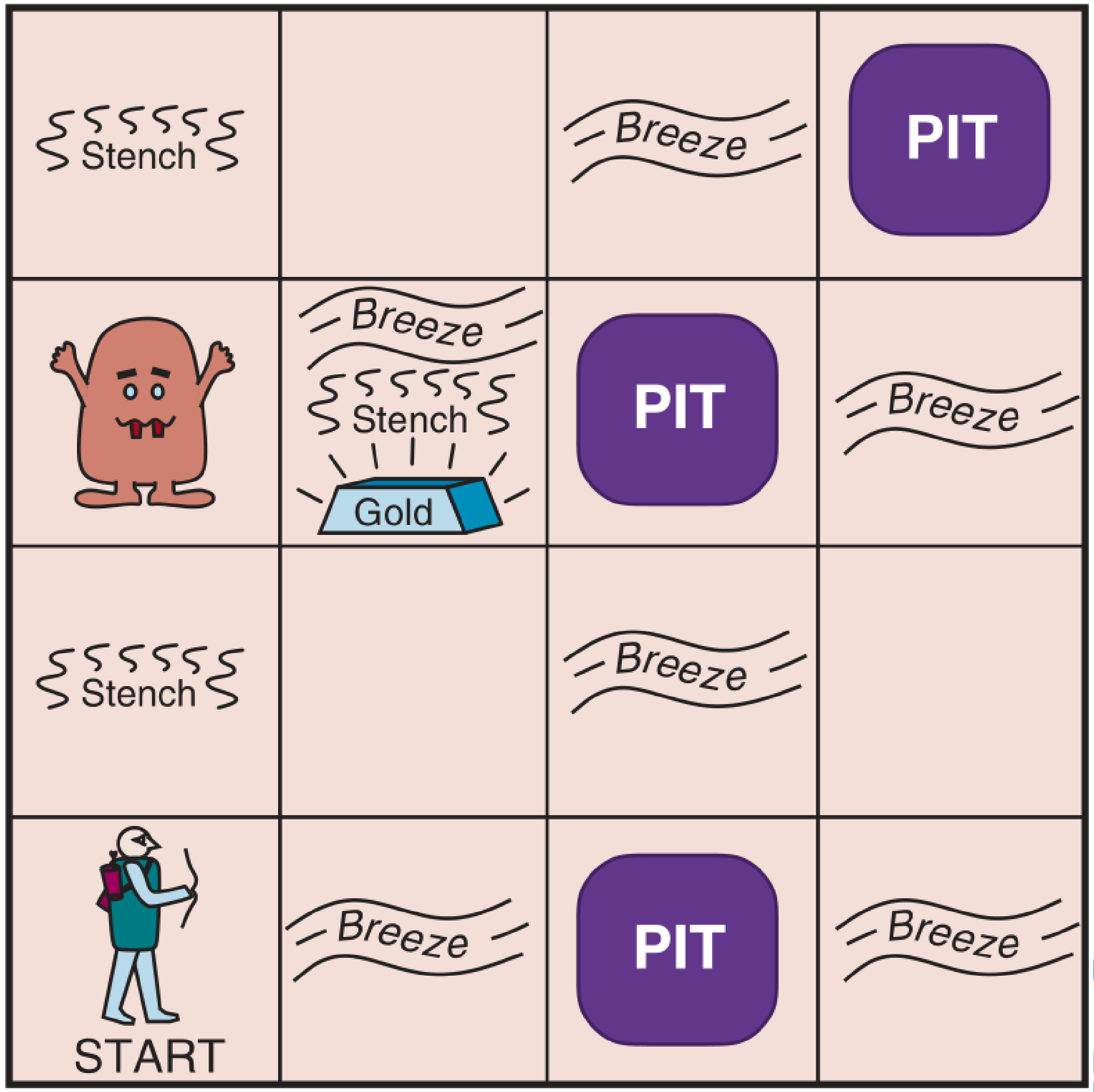
The Wumpus world $\mathcal{KB}$

Symbols for each $[x, y]$:

- $P_{i,j}$ is true if there is a pit in $[i, j]$
- $B_{i,j}$ is true if there is a breeze in $[i, j]$
- and the same for W (wumpus), S (stench), and L (agent's location)

$\mathcal{KB}$ for the *reduced* Wumpus world:

- $R_1 : \neg P_{1,1}$
- $R_2 : B_{1,1} \iff P_{1,2} \lor P_{2,1}$
- $R_3 : B_{2,1} \iff P_{1,1} \lor P_{2,2} \lor P_{3,1}$
- $R_4 : \neg B_{1,1}$



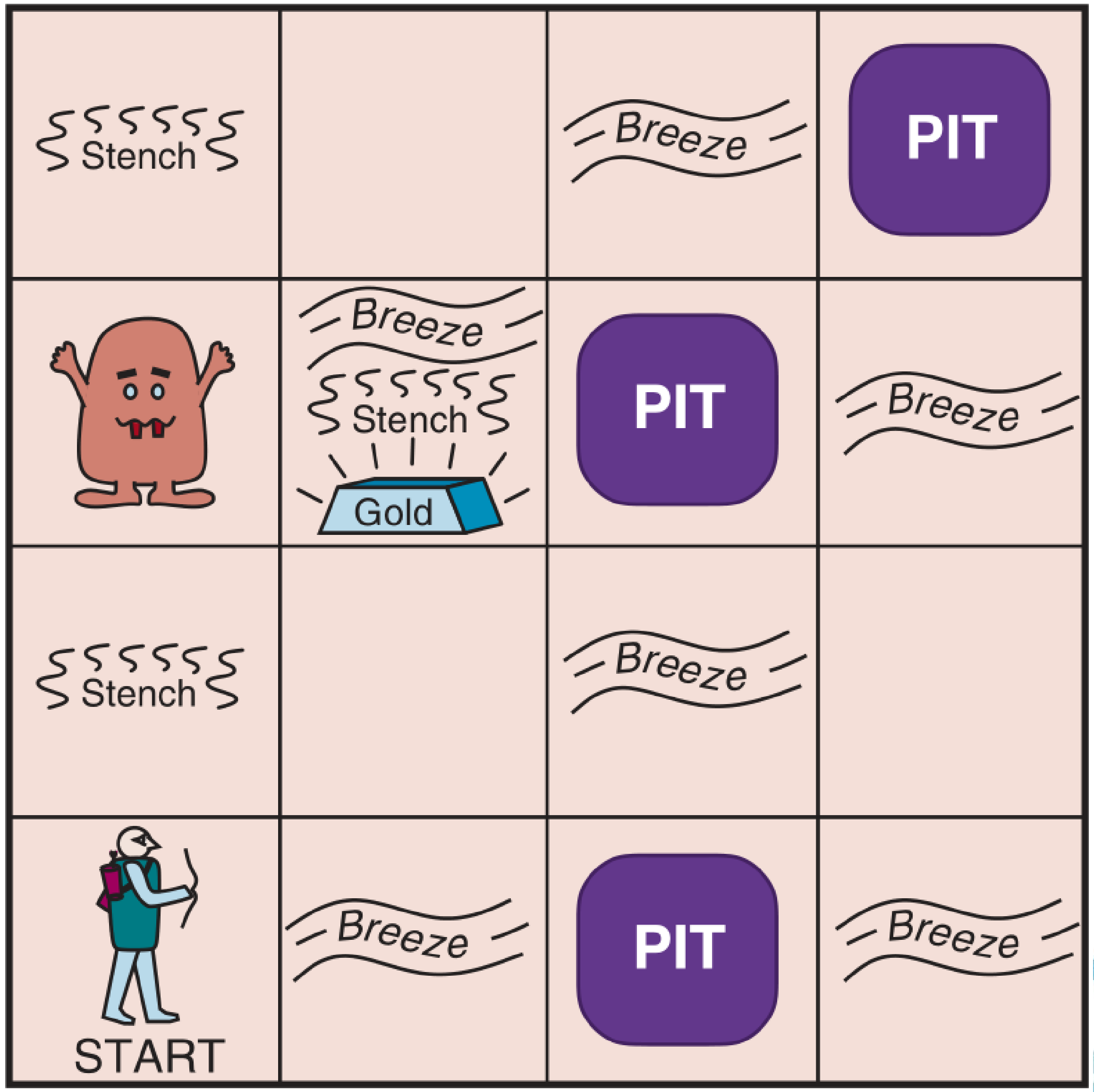
The Wumpus world $\mathcal{KB}$

Symbols for each $[x, y]$:

- $P_{i,j}$ is true if there is a pit in $[i, j]$
- $B_{i,j}$ is true if there is a breeze in $[i, j]$
- and the same for W (wumpus), S (stench), and L (agent's location)

$\mathcal{KB}$ for the *reduced* Wumpus world:

- $R_1 : \neg P_{1,1}$
- $R_2 : B_{1,1} \iff P_{1,2} \vee P_{2,1}$
- $R_3 : B_{2,1} \iff P_{1,1} \vee P_{2,2} \vee P_{3,1}$
- $R_4 : \neg B_{1,1}$
- $R_5 : B_{2,1}$



INFORMATIKA UNPAR

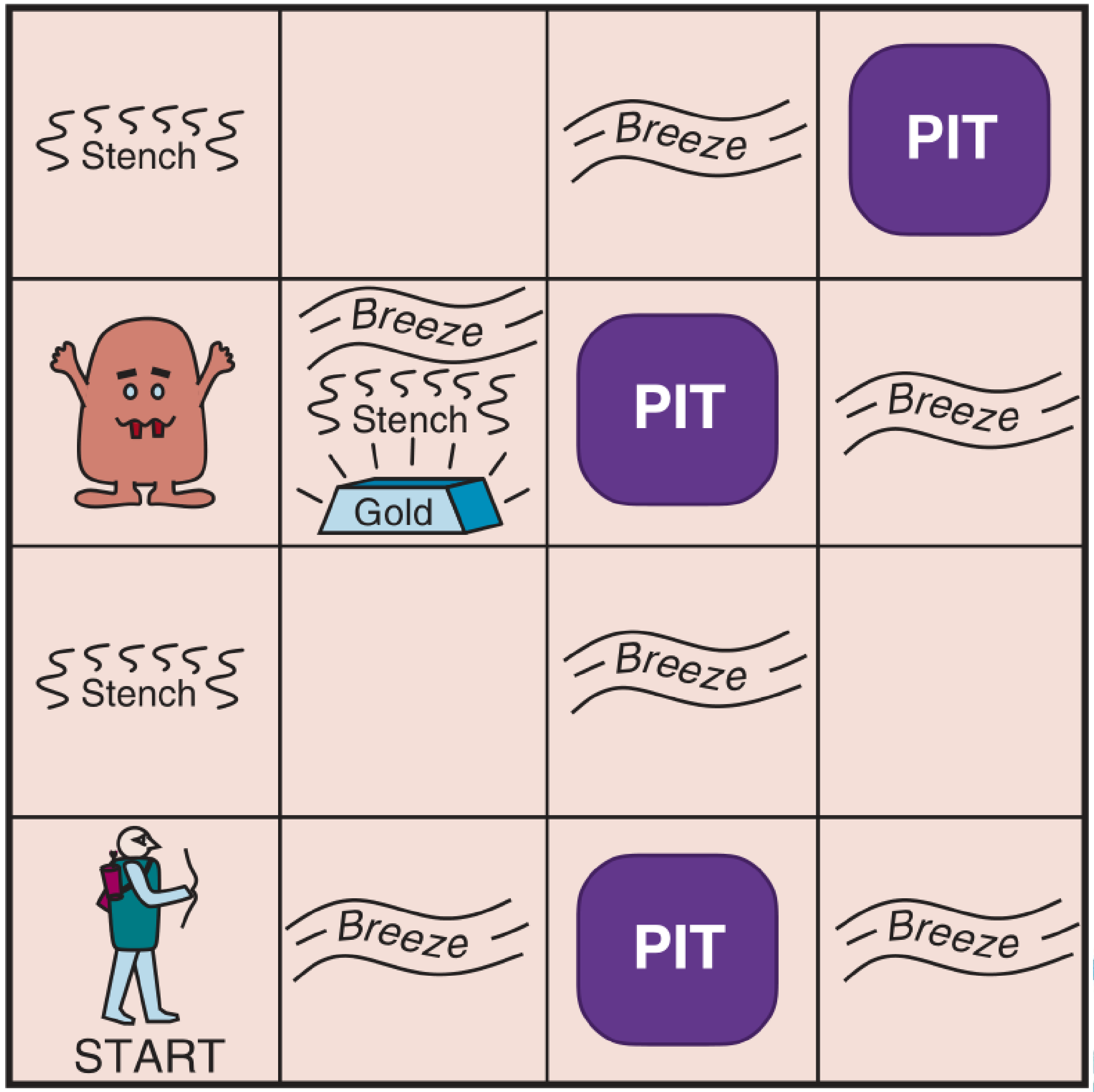
The Wumpus world $\mathcal{KB}$

Symbols for each $[x, y]$:

- $P_{i,j}$ is true if there is a pit in $[i, j]$
- $B_{i,j}$ is true if there is a breeze in $[i, j]$
- and the same for W (wumpus), S (stench), and L (agent's location)

$\mathcal{KB}$ for the *reduced* Wumpus world:

- $R_1 : \neg P_{1,1}$
- $R_2 : B_{1,1} \iff P_{1,2} \vee P_{2,1}$
- $R_3 : B_{2,1} \iff P_{1,1} \vee P_{2,2} \vee P_{3,1}$
- $R_4 : \neg B_{1,1}$
- $R_5 : B_{2,1}$



Questions (based on above $\mathcal{KB}$):

- $\mathcal{KB} \models: P_{1,2}$
- $\mathcal{KB} \models: P_{2,2}$

INFORMATIKA UNPAR

Model Checking

Model Checking

Based on truth table enumeration

Model Checking

Based on truth table enumeration

Models: assignments of TRUE or FALSE to every proposition symbol.

Model Checking

Based on truth table enumeration

Models: assignments of TRUE or FALSE to every proposition symbol.

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

Based on truth table enumeration

Models: assignments of TRUE or FALSE to every proposition symbol.

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	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*	<u>*true*</u>
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	<u>*true*</u>
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	<u>*true*</u>
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*

If $\mathcal{KB}$ and α contain n symbols, then there are 2^n models and the complexity of the enumeration (algorithm) is $O(2^n)$

Theorem Proving

Theorem Proving

Method checking is inefficient: truth tables might have an exponential number of models

Theorem Proving

Method checking is inefficient: truth tables might have an exponential number of models

Search for proofs is a more efficient because we can ignore irrelevant information

Theorem Proving

Method checking is inefficient: truth tables might have an exponential number of models

Search for proofs is a more efficient because we can ignore irrelevant information

Apply inference rules directly to the $\mathcal{KB}$ without consulting the model. It is **sound**.

Theorem Proving

Method checking is inefficient: truth tables might have an exponential number of models

Search for proofs is a more efficient because we can ignore irrelevant information

Apply inference rules directly to the $\mathcal{KB}$ without consulting the model. It is **sound**.

- Modus Ponens
- And-Elimination
- standard logical equivalences: biconditional elimination, De Morgan rule, etc

INFORMATIKA UNPAR

Theorem Proving

Method checking is inefficient: truth tables might have an exponential number of models

Search for proofs is a more efficient because we can ignore irrelevant information

Apply inference rules directly to the $\mathcal{KB}$ without consulting the model. It is **sound**.

- Modus Ponens
- And-Elimination
- standard logical equivalences: biconditional elimination, De Morgan rule, etc

Inference as a search problem!

Theorem Proving

Method checking is inefficient: truth tables might have an exponential number of models

Search for proofs is a more efficient because we can ignore irrelevant information

Apply inference rules directly to the $\mathcal{KB}$ without consulting the model. It is **sound**.

- Modus Ponens
- And-Elimination
- standard logical equivalences: biconditional elimination, De Morgan rule, etc

Inference as a search problem!

Ways to ensure **completeness**:

INFORMATIKA UNPAR

Theorem Proving

Method checking is inefficient: truth tables might have an exponential number of models

Search for proofs is a more efficient because we can ignore irrelevant information

Apply inference rules directly to the $\mathcal{KB}$ without consulting the model. It is **sound**.

- Modus Ponens
- And-Elimination
- standard logical equivalences: biconditional elimination, De Morgan rule, etc

Inference as a search problem!

Ways to ensure **completeness**:

- Proof by *resolution*: use powerful inference rules (resolution rules)

Theorem Proving

Method checking is inefficient: truth tables might have an exponential number of models

Search for proofs is a more efficient because we can ignore irrelevant information

Apply inference rules directly to the $\mathcal{KB}$ without consulting the model. It is **sound**.

- Modus Ponens
- And-Elimination
- standard logical equivalences: biconditional elimination, De Morgan rule, etc

Inference as a search problem!

Ways to ensure **completeness**:

- Proof by *resolution*: use powerful inference rules (resolution rules)
- *Forward or Backward chaining*: use of modus ponens on a restricted form of propositions (Horn clauses)

Resolution

Resolution: a single inference rule

Resolution

Resolution: a single inference rule

Resolution yields a complete (and sound) inference algorithm when coupled with any complete search algorithm

Resolution

Resolution: a single inference rule

Resolution yields a complete (and sound) inference algorithm when coupled with any complete search algorithm

Unit resolution rule:

$$\frac{\ell_1 \vee \ldots \vee \ell_i \vee \ldots \vee \ell_k, \quad m}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k}$$

where ℓ_i and m are complementary literals

INFORMATIKA UNPAR

Resolution

Resolution: a single inference rule

Resolution yields a complete (and sound) inference algorithm when coupled with any complete search algorithm

Unit resolution rule:

$$\frac{\ell_1 \vee \ldots \vee \ell_i \vee \ldots \vee \ell_k, \quad m}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k}$$

where ℓ_i and m are complementary literals

Resolution inference rule

$$\frac{\ell_1 \vee \ldots \vee \ell_i \vee \ldots \vee \ell_k, \quad m_1 \vee \ldots \vee m_j \vee \ldots \vee m_n}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee}$$

where ℓ_i and m_j are complementary literals

INFORMATIKA UNPAR

Resolution

Resolution: a single inference rule

Resolution yields a complete (and sound) inference algorithm when coupled with any complete search algorithm

Unit resolution rule:

$$\frac{\ell_1 \vee \ldots \vee \ell_i \vee \ldots \vee \ell_k, \quad m}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k}$$

where ℓ_i and m are complementary literals

Resolution inference rule

$$\frac{\ell_1 \vee \ldots \vee \ell_i \vee \ldots \vee \ell_k, \quad m_1 \vee \ldots \vee m_j \vee \ldots \vee m_n}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee}$$

where ℓ_i and m_j are complementary literals

The $\mathcal{KB}$ must be transformed to CNF (Conjunctive Normal Form): Conjunction of disjunction of literals. Example: $(A \vee B \vee \neg C) \wedge (C \vee \neg D)$

INFORMATIKA UNPAR

Example of Resolution

Wumpus: there is no breeze at [1,1], so there can be no pits in neighboring squares

INFORMATIKA UNPAR

Example of Resolution

Wumpus: there is no breeze at [1,1], so there can be no pits in neighboring squares

$$\mathcal{KB} = R_2 \wedge R_4 = (B_{1,1} \iff (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$$

Example of Resolution

Wumpus: there is no breeze at [1,1], so there can be no pits in neighboring squares

$$\mathcal{KB} = R_2 \wedge R_4 = (B_{1,1} \iff (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$$

We want to prove that there is no pits at [1,2]

$$\alpha = \neg P_{1,2}$$

INFORMATIKA UNPAR

Example of Resolution

Wumpus: there is no breeze at [1,1], so there can be no pits in neighboring squares

$$\mathcal{KB} = R_2 \wedge R_4 = (B_{1,1} \iff (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$$

We want to prove that there is no pits at [1,2]

$$\alpha = \neg P_{1,2}$$

Answer:

- To show $KB \models \alpha$ we show that $(KB \wedge \neg\alpha)$ is unsatisfiable

INFORMATIKA UNPAR

Example of Resolution

Wumpus: there is no breeze at [1,1], so there can be no pits in neighboring squares

$$\mathcal{KB} = R_2 \wedge R_4 = (B_{1,1} \iff (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$$

We want to prove that there is no pits at [1,2]

$$\alpha = \neg P_{1,2}$$

Answer:

- To show $KB \models \alpha$ we show that $(KB \wedge \neg\alpha)$ is unsatisfiable
- Convert $(KB \wedge \neg\alpha)$ into CNF

INFORMATIKA UNPAR

Example of Resolution

Wumpus: there is no breeze at [1,1], so there can be no pits in neighboring squares

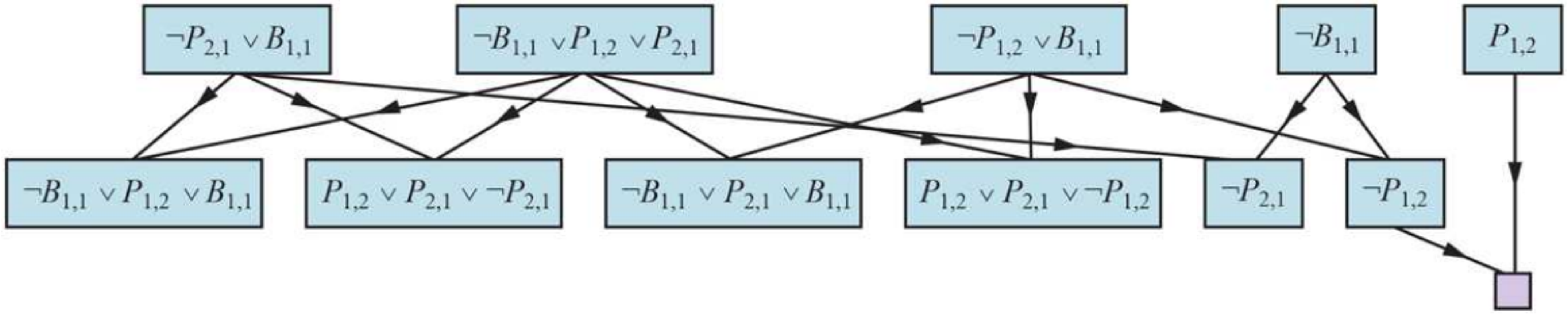
$$\mathcal{KB} = R_2 \wedge R_4 = (B_{1,1} \iff (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$$

We want to prove that there is no pits at [1,2]

$$\alpha = \neg P_{1,2}$$

Answer:

- To show $KB \models \alpha$ we show that $(KB \wedge \neg\alpha)$ is unsatisfiable
- Convert $(KB \wedge \neg\alpha)$ into CNF



Example of Resolution

Wumpus: there is no breeze at [1,1], so there can be no pits in neighboring squares

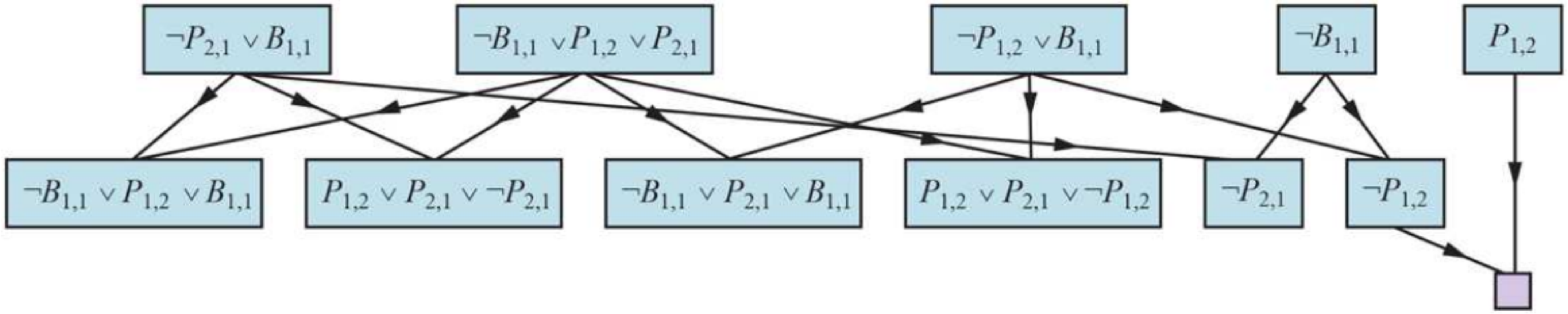
$$\mathcal{KB} = R_2 \wedge R_4 = (B_{1,1} \iff (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$$

We want to prove that there is no pits at [1,2]

$$\alpha = \neg P_{1,2}$$

Answer:

- To show $KB \models \alpha$ we show that $(KB \wedge \neg\alpha)$ is unsatisfiable
- Convert $(KB \wedge \neg\alpha)$ into CNF



- An empty clause is yielded, meaning the query is proven

Chaining

Chaining

$\mathcal{KB}$ = conjunction of *Horn clauses*

$\mathcal{KB}$ = conjunction of *Horn clauses*

Horn clause is a disjunction of literals of which at most one is positive

$$\neg\ell_1 \vee \neg\ell_2 \vee \ldots \vee \neg\ell_n \vee \ell_{n+1} \equiv \ell_1 \wedge \ell_2 \wedge \ldots \wedge \ell_n \implies \ell_{n+1}$$

Chaining

$\mathcal{KB}$ = conjunction of *Horn clauses*

Horn clause is a disjunction of literals of which at most one is positive

$$\neg \ell_1 \vee \neg \ell_2 \vee \ldots \vee \neg \ell_n \vee \ell_{n+1} \equiv \ell_1 \wedge \ell_2 \wedge \ldots \wedge \ell_n \implies \ell_{n+1}$$

Inference with Horn clauses can be done through the **forward-chaining** and **backward-chaining** algorithms that run in linear time

Chaining

$\mathcal{KB}$ = conjunction of *Horn clauses*

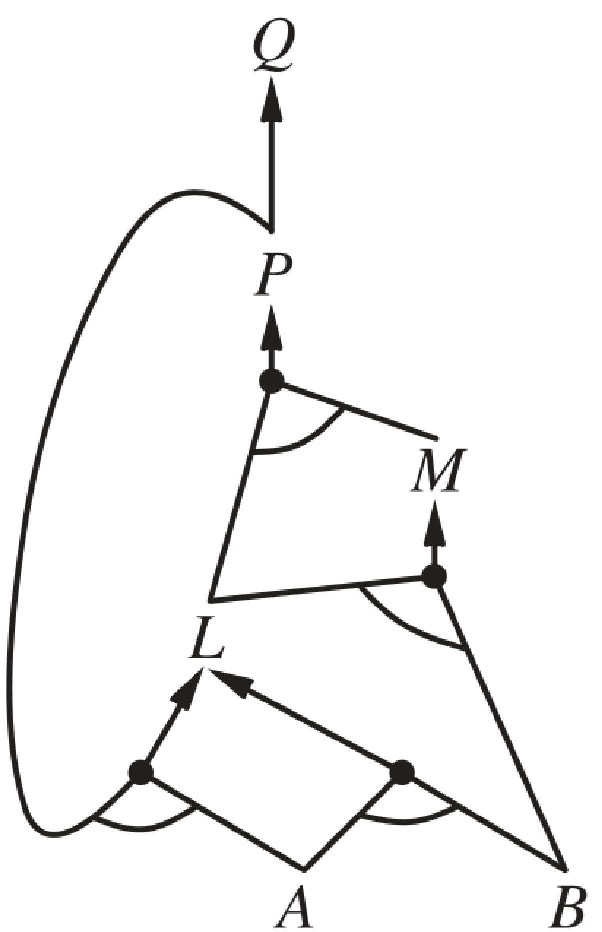
Horn clause is a disjunction of literals of which at most one is positive

$$\neg\ell_1 \vee \neg\ell_2 \vee \ldots \vee \neg\ell_n \vee \ell_{n+1} \equiv \ell_1 \wedge \ell_2 \wedge \ldots \wedge \ell_n \implies \ell_{n+1}$$

Inference with Horn clauses can be done through the **forward-chaining** and **backward-chaining** algorithms that run in linear time

Example:

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



Representation Languages

Propositional Logic for $\mathcal{KB}$ has limitations

Representation Languages

Propositional Logic for $\mathcal{KB}$ has limitations

- unable to express information about different objects and their relations

INFORMATIKA UNPAR

Representation Languages

Propositional Logic for $\mathcal{KB}$ has limitations

- unable to express information about different objects and their relations
- unable to express a fact for a set of objects without enumerating all of them

Representation Languages

Propositional Logic for $\mathcal{KB}$ has limitations

- unable to express information about different objects and their relations
- unable to express a fact for a set of objects without enumerating all of them

Why don't we use other languages?

Representation Languages

Propositional Logic for $\mathcal{KB}$ has limitations

- unable to express information about different objects and their relations
- unable to express a fact for a set of objects without enumerating all of them

Why don't we use other languages?

- natural language: use for communication, ambiguity

INFORMATIKA UNPAR

Representation Languages

Propositional Logic for $\mathcal{KB}$ has limitations

- unable to express information about different objects and their relations
- unable to express a fact for a set of objects without enumerating all of them

Why don't we use other languages?

- natural language: use for communication, ambiguity
- programming language: no mechanism for deriving facts from other facts, lack the expressiveness required to directly handle partial information

Representation Languages

Propositional Logic for $\mathcal{KB}$ has limitations

- unable to express information about different objects and their relations
- unable to express a fact for a set of objects without enumerating all of them

Why don't we use other languages?

- natural language: use for communication, ambiguity
- programming language: no mechanism for deriving facts from other facts, lack the expressiveness required to directly handle partial information

Combining the best of formal and natural languages

Representation Languages

Propositional Logic for $\mathcal{KB}$ has limitations

- unable to express information about different objects and their relations
- unable to express a fact for a set of objects without enumerating all of them

Why don't we use other languages?

- natural language: use for communication, ambiguity
- programming language: no mechanism for deriving facts from other facts, lack the expressiveness required to directly handle partial information

Combining the best of formal and natural languages

- Propositional logic: declarative, context-independet, unambigous,

Representation Languages

Propositional Logic for $\mathcal{KB}$ has limitations

- unable to express information about different objects and their relations
- unable to express a fact for a set of objects without enumerating all of them

Why don't we use other languages?

- natural language: use for communication, ambiguity
- programming language: no mechanism for deriving facts from other facts, lack the expressiveness required to directly handle partial information

Combining the best of formal and natural languages

- Propositional logic: declarative, context-independet, unambigous,
- natural language: representational ideas

INFORMATIKA UNPAR

Declarative language that can also recognize:

Declarative language that can also recognize:

- Objects: all nouns and noun phrases

First Order Logic

Declarative language that can also recognize:

- Objects: all nouns and noun phrases
- Relations: either unary or n-ary relations between objects

First Order Logic

Declarative language that can also recognize:

- Objects: all nouns and noun phrases
- Relations: either unary or n-ary relations between objects
- Functions: relations that only have one value for a given input

INFORMATIKA UNPAR

First Order Logic

Declarative language that can also recognize:

- Objects: all nouns and noun phrases
- Relations: either unary or n-ary relations between objects
- Functions: relations that only have one value for a given input

Syntax of FOL:

First Order Logic

Declarative language that can also recognize:

- Objects: all nouns and noun phrases
- Relations: either unary or n-ary relations between objects
- Functions: relations that only have one value for a given input

Syntax of FOL:

- Terms are either: constant symbols (e.g. 13), variables, or functions (e.g. sqrt(x))

INFORMATIKA UNPAR

First Order Logic

Declarative language that can also recognize:

- Objects: all nouns and noun phrases
- Relations: either unary or n-ary relations between objects
- Functions: relations that only have one value for a given input

Syntax of FOL:

- Terms are either: constant symbols (e.g. 13), variables, or functions (e.g. sqrt(x))
- Atomic formulas are predicates applied to terms (e.g., sister(a,b))

INFORMATIKA
UNPAR

First Order Logic

Declarative language that can also recognize:

- Objects: all nouns and noun phrases
- Relations: either unary or n-ary relations between objects
- Functions: relations that only have one value for a given input

Syntax of FOL:

- Terms are either: constant symbols (e.g. 13), variables, or functions (e.g. sqrt(x))
- Atomic formulas are predicates applied to terms (e.g., sister(a,b))
- Connectives($\vee, \wedge, \neg, \implies, \iff$), equality($=$), and quatifiers ($\forall, \exists$)

INFORMATIKA UNPAR

First Order Logic

Declarative language that can also recognize:

- Objects: all nouns and noun phrases
- Relations: either unary or n-ary relations between objects
- Functions: relations that only have one value for a given input

Syntax of FOL:

- Terms are either: constant symbols (e.g. 13), variables, or functions (e.g. sqrt(x))
- Atomic formulas are predicates applied to terms (e.g., sister(a,b))
- Connectives($\vee, \wedge, \neg, \implies, \iff$), equality($=$), and quatifiers ($\forall, \exists$)

Sentences can be created by applying connectives, equality, and/or quantifiers to atomic formulas

First Order Logic

Declarative language that can also recognize:

- Objects: all nouns and noun phrases
- Relations: either unary or n-ary relations between objects
- Functions: relations that only have one value for a given input

Syntax of FOL:

- Terms are either: constant symbols (e.g. 13), variables, or functions (e.g. sqrt(x))
- Atomic formulas are predicates applied to terms (e.g., sister(a,b))
- Connectives($\vee, \wedge, \neg, \implies, \iff$), equality($=$), and quatifiers ($\forall, \exists$)

Sentences can be created by applying connectives, equality, and/or quantifiers to atomic formulas

Examples:

INFORMATIKA UNPAR

First Order Logic

Declarative language that can also recognize:

- Objects: all nouns and noun phrases
- Relations: either unary or n-ary relations between objects
- Functions: relations that only have one value for a given input

Syntax of FOL:

- Terms are either: constant symbols (e.g. 13), variables, or functions (e.g. sqrt(x))
- Atomic formulas are predicates applied to terms (e.g., sister(a,b))
- Connectives($\vee, \wedge, \neg, \implies, \iff$), equality($=$), and quatifiers ($\forall, \exists$)

Sentences can be created by applying connectives, equality, and/or quantifiers to atomic formulas

Examples:

- All birds except dodo fly: $\forall\, x\ \mathrm{bird}(x) \wedge \neg \mathrm{dodo}(x) \implies \mathrm{fly}(x)$

Declarative language that can also recognize:

- Objects: all nouns and noun phrases
- Relations: either unary or n-ary relations between objects
- Functions: relations that only have one value for a given input

Syntax of FOL:

- Terms are either: constant symbols (e.g. 13), variables, or functions (e.g. sqrt(x))
- Atomic formulas are predicates applied to terms (e.g., sister(a,b))
- Connectives($\vee, \wedge, \neg, \implies, \iff$), equality($=$), and quatifiers ($\forall, \exists$)

Sentences can be created by applying connectives, equality, and/or quantifiers to atomic formulas

Examples:

- All birds except dodo fly: $\forall\, x\ bird(x) \wedge \neg dodo(x) \implies fly(x)$
- Some wombat like ice-cream: $\exists\, x\ wombat(x) \wedge likes(x, ice - cream)$

INFORMATIKA UNPAR

Inference for FOL

There are procedures to do inference with a knowledge base of FOL formulas:

Inference for FOL

There are procedures to do inference with a knowledge base of FOL formulas:

- *Universal Instantiation* and *Existential Instantiation* with *Unification*
- *Forward chaining*: used in deductive databases. It can be combined with relational database operations
- Backward chaining: used in logic programming that provide very fast inference

Inference for FOL

There are procedures to do inference with a knowledge base of FOL formulas:

- *Universal Instantiation* and *Existential Instantiation* with *Unification*
- *Forward chaining*: used in deductive databases. It can be combined with relational database operations
- Backward chaining: used in logic programming that provide very fast inference

Note on natural language:

Conversion between natural language and logical expressions is possible due to the expressiveness of FOL. This is very valuable in many areas, such as development of virtual assitants like Alexa, Cortana, Siri, and many more.

Summary

Logic is used to represent the environment of the agent and reason about that env.

Summary

Logic is used to represent the environment of the agent and reason about that env.

While models are encoded explicitly, there are some limitations:

Summary

Logic is used to represent the environment of the agent and reason about that env.

While models are encoded explicitly, there are some limitations:

- It is difficult to model every aspect of the world

INFORMATIKA UNPAR

Summary

Logic is used to represent the environment of the agent and reason about that env.

While models are encoded explicitly, there are some limitations:

- It is difficult to model every aspect of the world
- rule-based and do not use data like machine learning

Summary

Logic is used to represent the environment of the agent and reason about that env.

While models are encoded explicitly, there are some limitations:

- It is difficult to model every aspect of the world
- rule-based and do not use data like machine learning
- do not handle uncertainty like probability, although fuzzy logic allows for degree of truth

INFORMATIKA
UNPAR

informatika.unpar.ac.id
informatika@unpar.ac.id
if.unpar
if.unpar