Logical Representation & Reasoning: Propositional Logic

Part I: Motivation

Course on AI@IID

Slides adapted from Mannes Poel by M. Birna van Riemsdijk

Ways of using logic

- Mathematical analysis and proving theorems
- Describing hardware and software systems
  - Logical circuits
- Analyzing computational systems
  - Proving correctness of a program
- Artificial Intelligence
  - Programming the reasoning and decision making

Computational Technique

Machine Reasoning:
logic-based approaches

Week 1-3

Optimisation:
algorithmic approaches

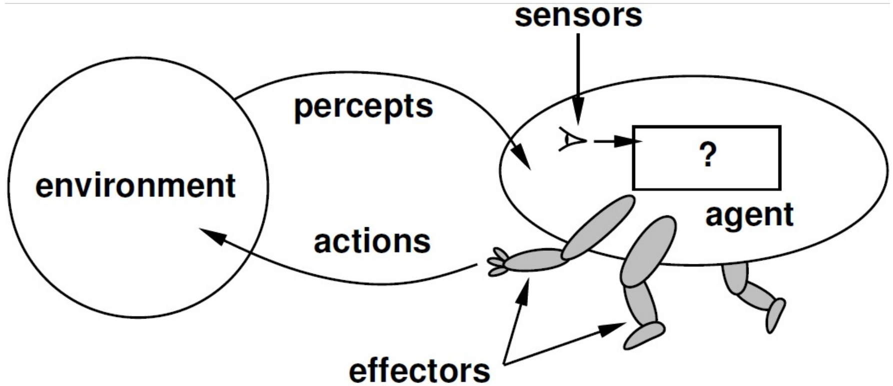
Week 2

Machine Learning:
data-oriented approaches

Week 4-6

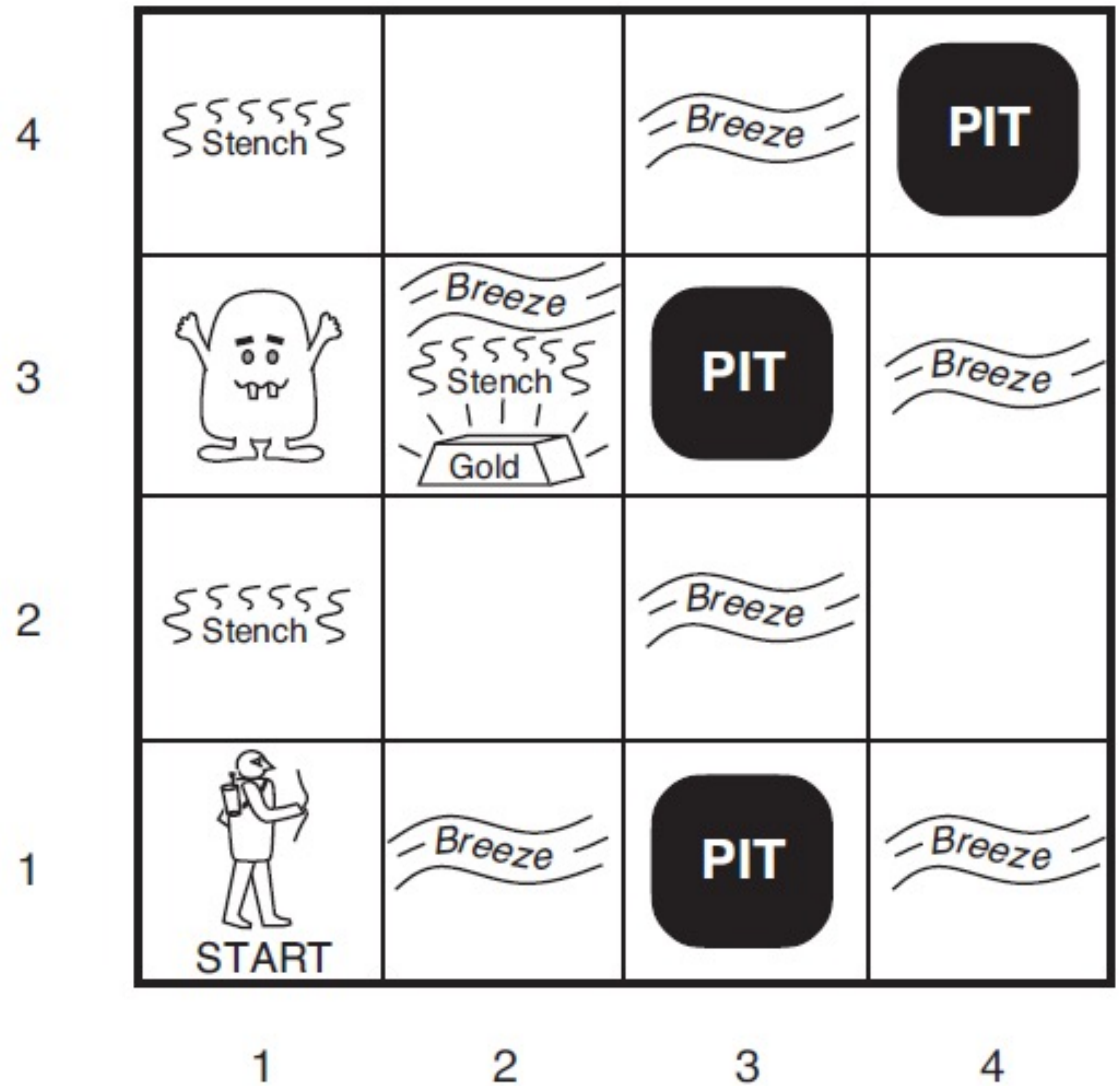
Designing rational agents

- Knowledge/logic based approach to designing and implementing the action selection [?] component of rational agents.

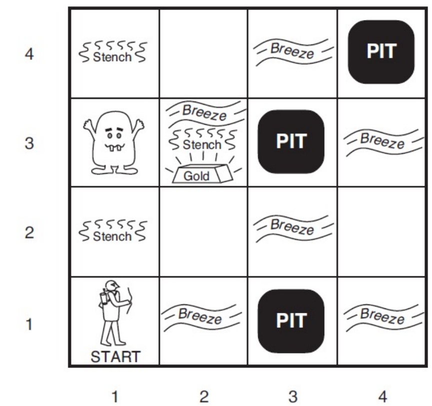


- Approach: logical modelling of the environment, the knowledge base *KB* of the agent and inference (reasoning) with this knowledge

Example: Wumpus World

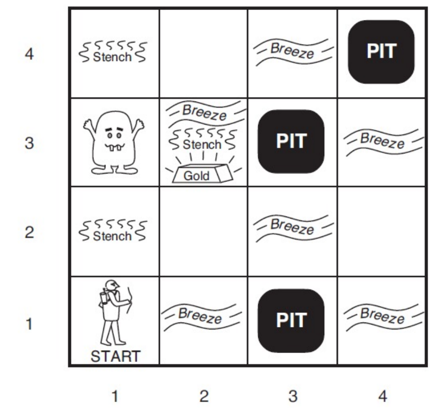


Goal



– Agent should find the gold, and climb out of the Wumpus cave with the gold
– Avoid pits (it cannot get out anymore), avoid Wumpus (it will get eaten)
– Use as few actions as possible

Environment Type

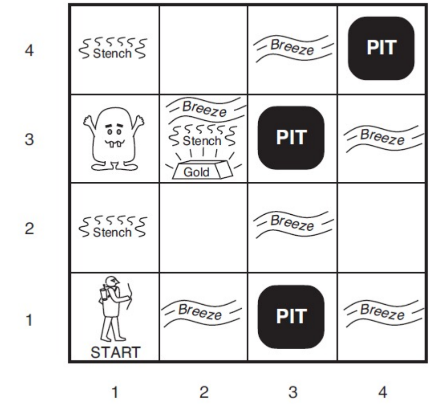


- Discrete: finite number of distinct states and actions
- Static: no changes in the evironment during the game (except by the agent),
- “Single agent”: Wumpus does not move
- Partially observable: agent only perceives what is in its own square, does not have full knowledge of the environment at the start

Actuators

- *Forward;* move one square forward in direction the agent faces if no wall in front, otherwise stay put
- *TurnLeft*, *TurnRight;* turn direction the agent is facing
- *Grab;* pick up the gold when on gold square.
- *Shoot;* fire arrow in direction the agent is facing, either hits (and kills) Wumpus or hits wall; only one arrow. Wumpus does not move!
- *Climb*, to climb out of the wumpus cave, only from square [1,1].

Sensors



– *Stench*, in squares adjacent (not diagonal) to the wumpus there is stench.

– *Breeze*, squares adjacent to a pit will have a *Breeze.*

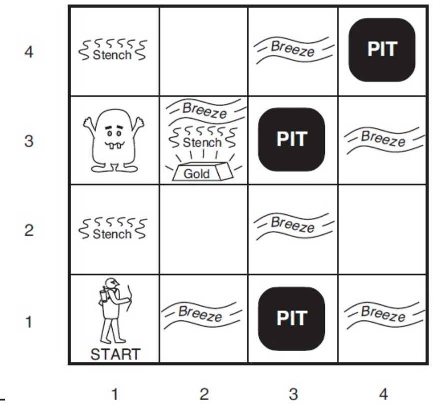
– *Glitter*, on the square with gold the agent will perceive a glitter.

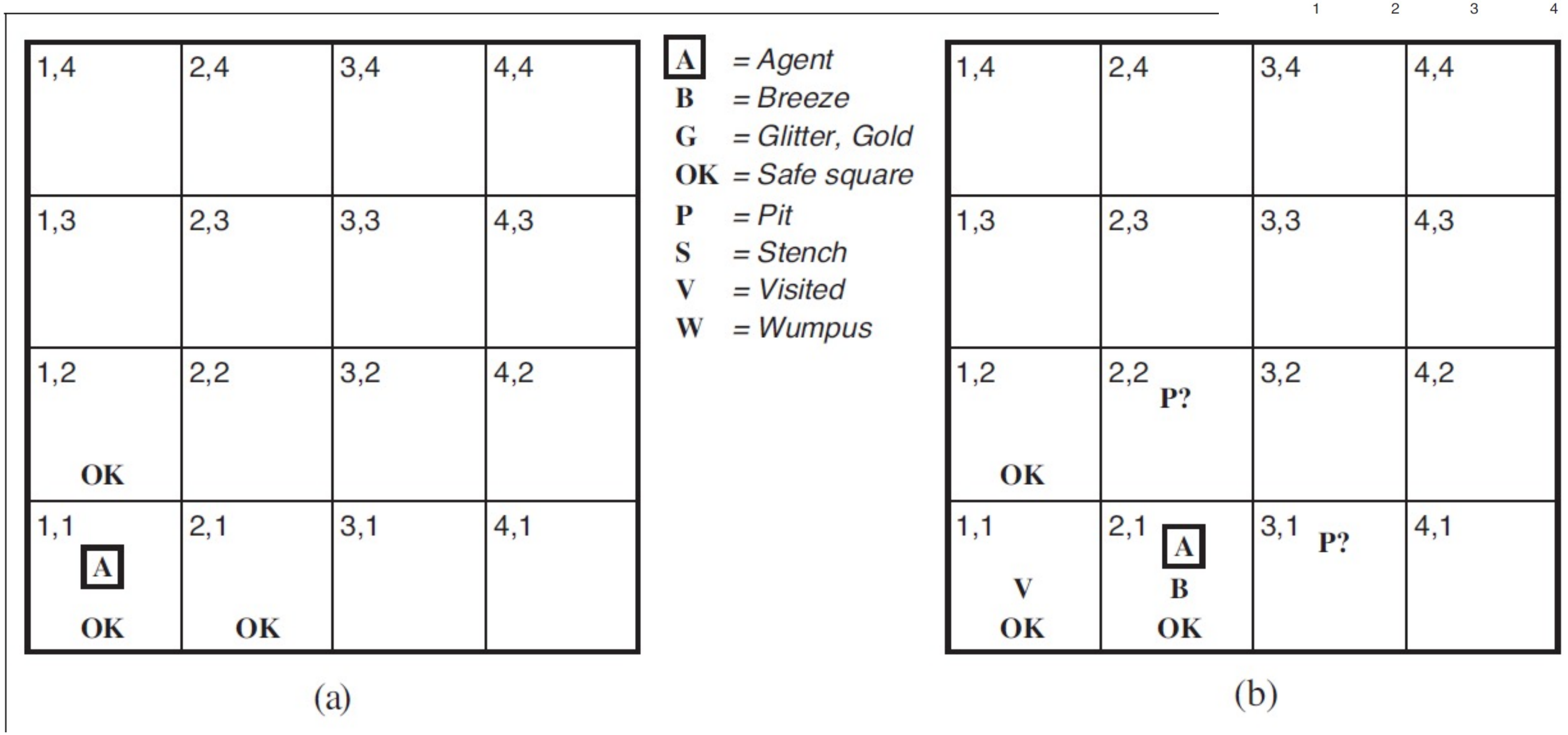
– *Bump*, when hitting the wall.

– *Scream*, when the wumpus is killed.

Transmitted to agent as list of five items, e.g., [Stench, none, Glitter, none, none]

Action Selection (1)

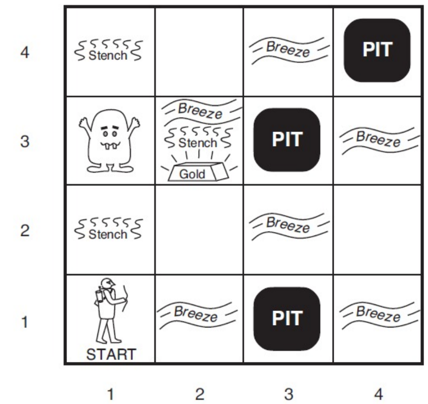


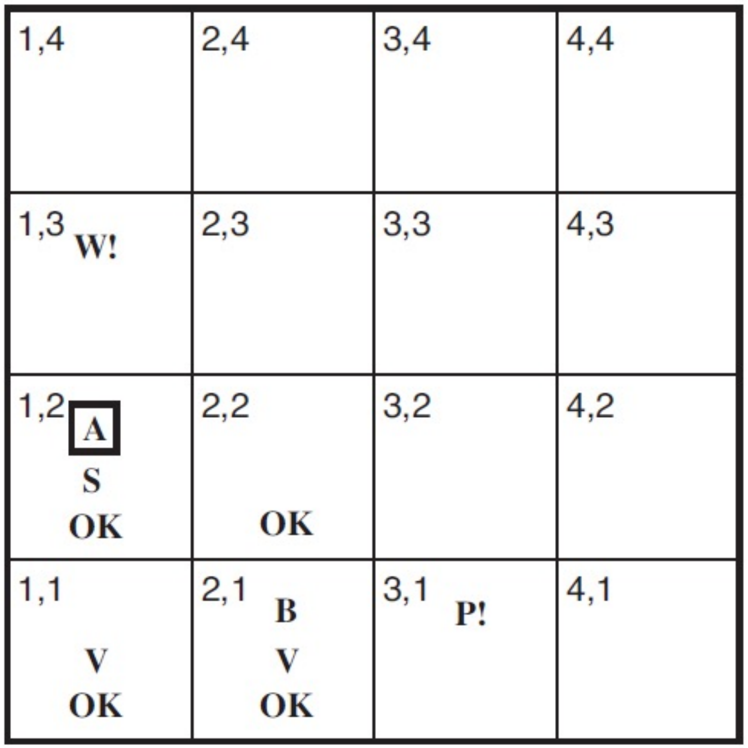


[None,None,None,None,None]

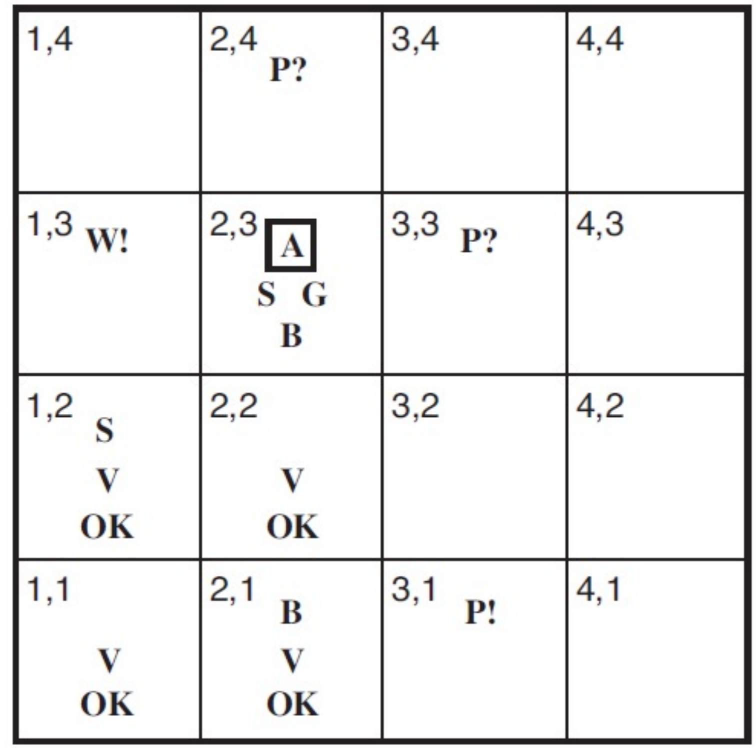
[None,Breeze,None,None,None]

Action Selection (2)





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



[Stench,None,None,None,None]

[Stench,Breeze,Glitter,None,None]

Knowledge-Based Agent

- Represent knowledge about logical reasoning steps needed for action selection in the agent
- Express using a Knowledge Representation Language
- Derive new knowledge by means of inferences, i.e., applying logical reasoning steps to derive new information from existing information
- For agents: derive next action from information and inferences about the state of the environment

Logical Representation & Reasoning: Propositional Logic
Part II: Syntax

Course on AI@IID

Slides adapted from Mannes Poel by
M. Birna van Riemsdijk

What is a logic?

1. Structured way of representing statements about “the world”: *syntax*
2. Abstract representation of states of the world: *models*
3. Precise definition of when a sentence is true with respect to a model: *semantics*
4. Rules for deriving new formulas (logical statements) from existing ones: *reasoning*

Different logics are suitable for representing different aspects about the world or computer programs, e.g., knowledge, time, actions, …

Propositional Logic

- Knowledge is represented by propositions
- **Atomic sentences (atoms)**: $p, q, gold, breeze, \ldots$
- **Complex sentences** are constructed from atomic sentences using **parentheses** and **logical connectives**: $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\Rightarrow$ $(if \ldots then \ldots)$, $\Leftrightarrow$ $(if\ and\ only\ if)$

  e.g., $p, q, p \vee q, p \wedge q, \neg p, p \Rightarrow q$
- **Literals**: $p, q, \neg p$. Positive literals and negative literals.

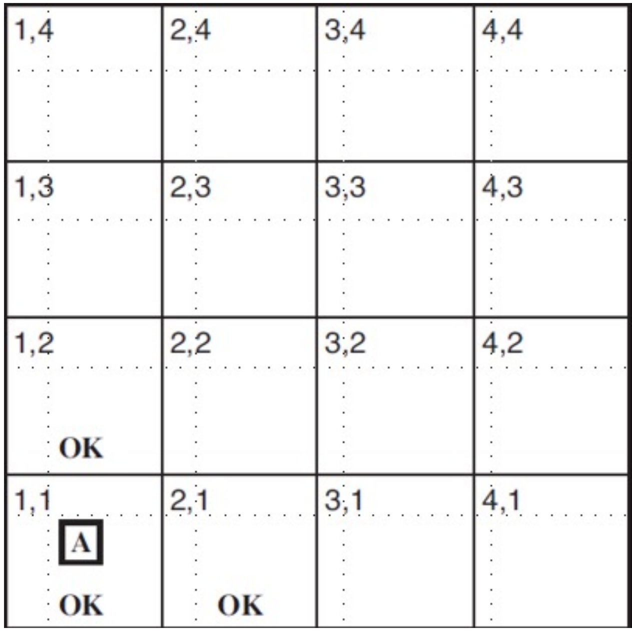
Syntax

- If S is a sentence, then (S) is a sentence
- If S is a sentence, then [S] is a sentence
- If S is a sentence, then $\neg S$ is a sentence (negation)
- If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)
- If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)
- If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)
- If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional or equivalence)
- Atomic sentences: p, q, r, s, t, ...

Knowledge-Based Wumpus Agent

- How to formalize this kind of reasoning for an agent who searches for gold in the Wumpus world.
- No pit in [1,1]: $\neg P_{1,1}$ (could also be denoted by q, but this is more intuitive)
- Breeze in [1,1] implies pit in adjacent squares:
$$B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})$$
- But we also need: $\left(P_{1,2} \vee P_{2,1}\right) \Rightarrow B_{1,1}$
- Leads to: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

Logic model (KB) for Wumpus world



- $\neg P_{1,1}$
- $B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$
- $B_{2,1} \Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{3,1})$
- The agent also knows:
- $\neg B_{1,1}$
- How can the agent deduce that there is no pit in [1,2]: $\neg P_{1,2}$?

Propositional Logic

- Almost no structure. Knowledge such as "There is a cat in the tree behind the house" can be represented by just one proposition p.
- Similar "there is food at location (2,3)" can be represented by q.
- Like variables in a programming language, you can give them a "meaningful" name. But that is not obligatory.
- Cannot model relations such as *Mother(x,y)* etc.

Logical Representation & Reasoning: Propositional Logic
Part III: Semantics

Course on AI@IID

Slides adapted from Mannes Poel by M. Birna van Riemsdijk

Interpretation of sentences

- We have a syntax to describe which are valid sentences of propositional logic
- How to define what these sentences mean (semantics)?
  - e.g., the meaning of a software program is a description of what happens when executed
  - how to describe the meaning of a logical sentence?

Semantics: truth values

- Describe meaning of a sentence in terms of its truth value (true/false or equivalently 1/0)
- How to determine truth value of a sentence?
  - based on truth values of atoms,
  - and how to combine these truth values for complex sentences
- E.g., let p, q be atoms with p is true, q is true. Then $p \wedge q$ is true.

Models

- An assignment of truth values (true/false) for the atoms of a sentence in proposition logic is called a model.
- Models can be thought of as an abstract representation of the state of a real or abstract "world", e.g., *rain* is true, $P_{1,1}$ is false.
- We say *m* is a model of a sentence α or m satisfies α, if α is true in *m*, notation: $m \models \alpha$
  - *E.g., {p is true, q is true} is a model of* $p \wedge q$
- *M(α)* is the set of all models of α

Truth table

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

$$earlyMeeting \Rightarrow yoghurt$$

Semantics of sentences

Given a model *m* (truth assigment to the atomic sentences or symbols). Then

- $\neg S$ is true iff S is false
- $S_1 \wedge S_2$ is true iff S_1 is true and S_2 is true
- $S_1 \vee S_2$ is true iff S_1 is true or S_2 is true
- $S_1 \Rightarrow S_2$ is true iff S_1 is false or S_2 is true
- $S_1 \Leftrightarrow S_2$ is true iff $S_1 \Rightarrow S_2$ is true and $S_2 \Rightarrow S_1$ is true

Truth table: example

	tea	coffee	juice	$(tea \lor coffee)$	$\land$	$juice$
1	0	0	0	0	0	see 3rd
2	0	0	1	0	0	column
3	0	1	0	1	0	
4	0	1	1	1	1	
5	1	0	0	1	0	
6	1	0	1	1	1	
7	1	1	0	1	0	
8	1	1	1	1	1	

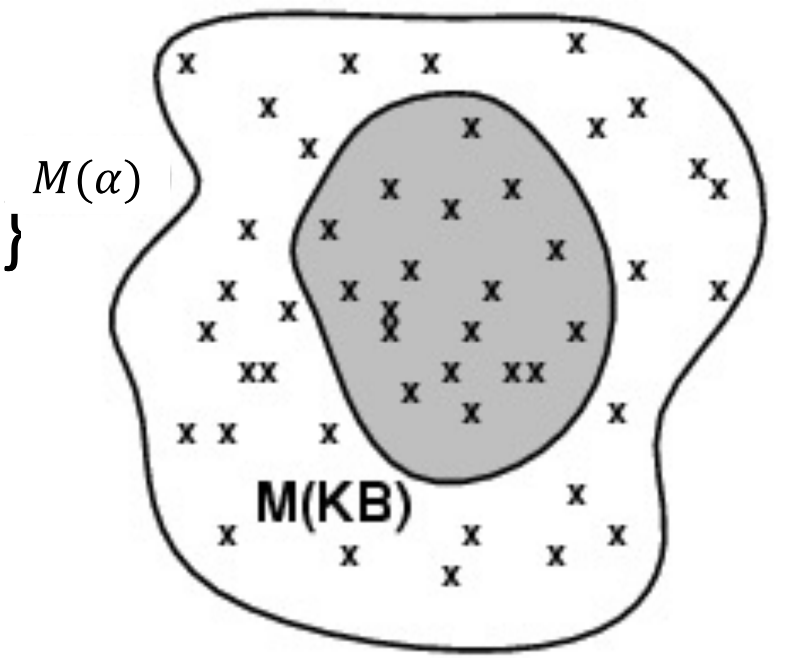
Convention:
- *false* = 0
- *true* = 1

Models:
- 8 models (2^3) total
- 3 models satisfy the proposition

Steps:
- Create a column for each atom and list the combinations of truth values
- From inner connectives to outer connectives: calculate the value of the complex formula based on the values of the atoms/subformulas

Entailment

- KB ⊨ α means that knowledge base KB, consisting of a set of propositional formulas, *entails* α. Meaning that in all models in which KB is true also α is true, i.e., $M(KB) \subseteq M(\alpha)$ (Model checking.)
  e.g., $p \wedge q \vDash p$ because
  M(KB) = {(p is true, q is true)}
  M(α) = {(p is true, q is true),
  (p is true, q is false)}



Concepts/Terminology

- **Logical equivalence:** $\alpha \equiv \beta$ if $M(\alpha) = M(\beta)$.
  - Example: $p \wedge q \equiv q \wedge p$.
- **Validity:** A sentence α is valid if it is true in all models (a.k.a. **tautologies**), e.g., $p \Rightarrow p$.
- **Satisfiability:** a sentence α is satisfiable if it is true in some model, i.e. $M(\alpha) \neq \emptyset$
- $KB \models \alpha$ if $KB \wedge \neg\alpha$ is unsatisfiable. (Proof by contradiction.)

Logical Representation & Reasoning: Propositional Logic

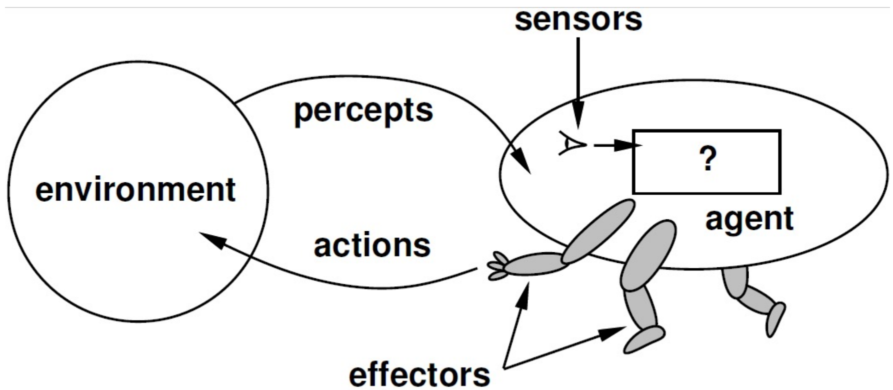
Part IV: Reasoning

Course on AI@IID

Slides adapted from Mannes Poel by M. Birna van Riemsdijk

Using logical reasoning

- Knowledge/logic based approach to designing and implementing the action selection [?] component of rational agents.



- Approach: logical modelling of the environment, the knowledge base *KB* of the agent and inference (reasoning) with this knowledge

Theorem proving

- Theorem proving: applying rules to derive a proof of $KB \vDash \alpha$ without model checking. Notation: $KB \vdash \alpha$
- Well known rule is **Modus Ponens**:

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

e.g., (earlyMeeting ⇒ yoghurt, earlyMeeting) ⊢ yoghurt

- Theorem proving is searching for proofs and can be formulated as a search problem.

Resolution Rule (1)

- Idea: eliminate opposite literals of two disjuncts.
- Unit resolution rule:

$$\frac{p \lor q \quad \neg q}{p} \qquad \frac{tea \lor coffee \quad \neg coffee}{tea}$$

- Resolution rule:

$$\frac{p \lor q \quad \neg q \lor r}{p \lor r} \qquad \frac{tea \lor coffee \quad \neg coffee \lor \neg yoghurt}{tea \lor \neg yoghurt}$$

Resolution rule (2)

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

Where l_i and m_j are complementary literals:

1. $l_i = \neg m_j$ or
2. $\neg l_i = m_j$

On a syntactic level.

One technical aspect: remove multiple copies of the same literal: $p \vee p \equiv p$.

Purpose of Resolution Rule

- The resolution rule is very powerful: with only the resolution rule we can prove $KB \vdash \alpha$ if $KB \vDash \alpha$ (completeness).
- Key components:
  - Any formula in propositional logic can be formulated as a conjunction of disjunctions of literals (Conjunctive Normal Form, CNF), e.g., $(p \vee \neg q) \wedge (\neg r \vee s)$.
  - Resolution can be applied to the conjuncts

Equivalence

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

Proof by Resolution for Propositional Logic

- Goal is to prove: $KB \vdash \alpha$

1. Add $\neg\alpha$ to KB and try to prove $KB \wedge \neg\alpha \vdash \perp$ proof by contradiction
2. Write $KB \wedge \neg\alpha$ in CNF
3. Apply *resolution* rule until no **new** clause can be added anymore or *false* is derived for instance due to $p \wedge \neg p$ (*empty clause*).

Conjunctive Normal Form (CNF)

- A sentence is in CNF if and only if (iff) it is of the form:

$$S_1 \wedge S_2 \wedge \cdots \wedge S_n$$

In which each S_i is of the form:

$$(l_1 \vee l_2 \vee \cdots \vee l_k)$$

With each l_i is a literal.

- Every syntactic correct sentence in Propositional Logic can be written in CNF

Procedure for CNF

1. Replace $S_1 \Leftrightarrow S_2$ by $(S_1 \Rightarrow S_2) \wedge (S_2 \Rightarrow S_1)$
2. Replace $S_1 \Rightarrow S_2$ by $\neg S_1 \vee S_2$
3. Push $\neg$ inwards until it hits a literal. Use

$$\neg(S_1 \wedge S_2) = \neg S_1 \vee \neg S_2$$
$$\neg(S_1 \vee S_2) = \neg S_1 \wedge \neg S_2$$
$$\neg\neg S_1 = S_1$$

4. Distribute $\vee$ over $\wedge$ whenever possible:

$$(S_1 \wedge S_2) \vee S_3 = (S_1 \vee S_3) \wedge (S_2 \vee S_3)$$

Resolution example (1)

$\mathrm{KB} =$

1. $bread$
2. $(tea \lor coffee) \land juice$
3. $earlyMeeting \rightarrow yoghurt$
4. $yoghurt \rightarrow \neg coffee$
5. $earlyMeeting$

Can we prove $KB \vdash tea$ by means of resolution?

Resolution example: step 1

Step 1: add the negation of the conclusion, i.e., $\neg tea$, to the KB

KB' =

1. $bread$
2. $(tea \lor coffee) \land juice$
3. $earlyMeeting \rightarrow yoghurt$
4. $yoghurt \rightarrow \neg coffee$
5. $earlyMeeting$
6. $\neg tea$

Resolution example: step 2

Step 2: write the new KB' in CNF

1. $bread$
2. $(tea \vee coffee) \wedge juice$
3. $earlyMeeting \rightarrow yoghurt$
4. $yoghurt \rightarrow \neg coffee$
5. $earlyMeeting$
6. $\neg tea$

1. $bread$
2. $tea \vee coffee,\ juice$
3. $\neg earlyMeeting \vee yoghurt$
4. $\neg yoghurt \vee \neg coffee$
5. $earlyMeeting$
6. $\neg tea$

Resolution example: step 3

Step 3: apply the resolution rule and try to derive the empty clause (falsum)

1. $bread$
2. $tea \vee coffee,\ juice$
3. $\neg earlyMeeting \vee yoghurt$
4. $\neg yoghurt \vee \neg coffee$
5. $earlyMeeting$
6. $\neg tea$
7. $coffee$ (2,6)
8. $\neg yoghurt$ (7,4)
9. $yoghurt$ (3,5)
10. $empty\ clause$ (8,9)

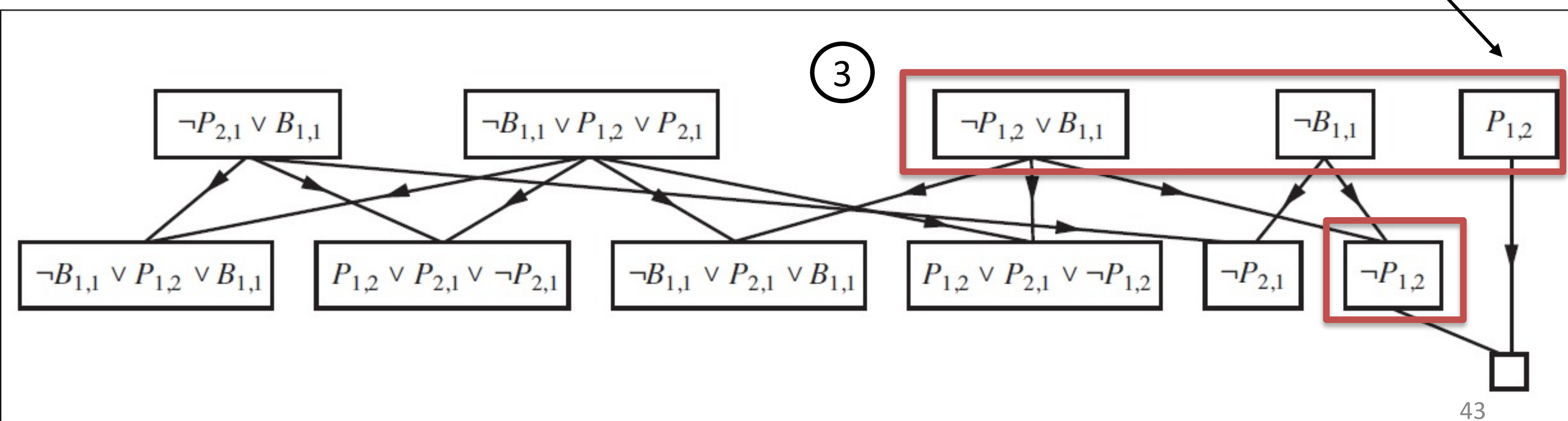
Proof by contradiction: assume the opposite of conclusion → contradiction → conclusion (tea) must follow.

Resolution example (Wumpus)

- $(B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1} \vdash \neg P_{1,2}$
- Write $KB \wedge \neg\alpha$ in CNF

1 Add negation of consequent

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



Soundness & Completeness

- Logical consequence:
  - $KB \vDash \alpha$: semantic entailment, defined through models
  - $KB \vdash \alpha$: a proof exist showing that by applying syntactic rules, α follows from KB.
- It is sound: if it is proven by resolution that α follows from *KB* then $KB \vDash \alpha$
- It is complete: If $KB \vDash \alpha$ then it can be proven by resolution that α follows from *KB.*

Logical Representation & Reasoning: Propositional Logic
Part V: Programming with Logic

Course on AI@IID

Slides by M. Birna van Riemsdijk

Programming in Prolog

- Practical assignment: Prolog
- Different programming paradigm!
- This course: basic introduction to Prolog
  - Compare with logic
  - Use for simple knowledge representation problems
- MOD8: more advanced programming skills

Programming with Logic

Knowledge Base + Inference Engine

Specified by programmer: logical statements

Inference engine derives new knowledge from the knowledge base by applying logical proof rules

Given by programming environment

Ideal of Logic Programming

- Programmer specifies knowledge ”declaratively”, i.e., only has to specify the logical statements that hold in the domain
- The inference engine does the work of executing the program (reasoning) to derive new statements, i.e., programmer does not need to know how the inference engine works
- In practice: (some) intricacies of the inference engine need to be understood to write and debug programs

Horn Clauses

- Logical statements in Prolog are restricted to "Horn clauses"
- Horn clause: disjunction of literals with (at most) 1 positive literal, e.g., $\neg p \vee \neg q \vee r$
- Horn clauses can more easily be interpreted when they are written as implications: $\neg p \vee \neg q \vee r \equiv \neg(p \wedge q) \vee r \equiv (p \wedge q) \Rightarrow r$
- I.e., a *rule* with a conjunction of atoms as the antecedent and a single atom as the conclusion

Rule-based reasoning: forward chaining

- Apply modus ponens to see what can be derived from a set of rules and atomic propositions

1. $b \land c \Rightarrow a$
2. $d \Rightarrow b$
3. d
4. c
5. b (2,3)
6. a (5,4,1)

- Used in rule-based expert systems
  - E.g., fever ∧ coughing ⇒ flue
  - Give *symptoms* fever and coughing as input and derive that we have the *disease* flue as output

Rule-based reasoning: backward chaining (1)

- Starting from the goal and reasoning backwards
  - E.g., does a follow from this KB?

1. $a \Leftarrow b \land c$
2. $b \Leftarrow d$
3. d
4. c
5. $[a]$ (goal)
6. $[b, c]$ (1)
7. $[d, c]$ (2)
8. $[c]$ (3)
9. [] (4)

- Used in Prolog
  - Powerful mechanism in combination with unification (next lecture)

Rule-based reasoning: backward chaining (2)

- Inference rule: resolution

1. $a \vee \neg b \vee \neg c$
2. $b \vee \neg d$
3. d
4. c
5. $\neg a$ (neg. conclusion)
6. $\neg b \vee \neg c$ (1,5)
7. $\neg d \vee \neg c$ (6,2)
8. $\neg c$ (7,3)
9. Empty clause (8,4)

- Properties
  - Goal is a horn clause without positive literal
  - Resolvent of a goal and a rule is again a goal
  - If empty clause can be derived: conclusion is proven

Prolog Program

- Two types of statements
  - Facts: horn clauses consisting of a single positive literal, e.g., p
  - Rules: horn clauses with one positive literal and one or more negative literals, e.g., $\neg p \vee \neg q \vee r$ written in rule form as $(p \wedge q) \Rightarrow r \equiv r \Leftarrow (p \wedge q)$
  - *No negative literals* in rules or facts!
    - MOD8: Negation As Failure (NAF)

Prolog Syntax

- Prolog syntax
  - All statements end with a full stop, e.g., `p.`
  - Implication ($\Leftarrow$) denoted as `:-`
  - Conjunction denoted as comma `,`
  - A rule $(p \wedge q) \Rightarrow r \equiv r \Leftarrow (p \wedge q)$ denoted as `r :- p,q.`
- Prolog terminology
  - A rule with head ‘r’ and body ‘p, q’.
  - Interpreted as ‘in order to derive r, we need to be able to derive p and q’.

Running Prolog

- Program = knowledge base:

```
a :- b,c.
b :- d.
d.
c.
```

Prolog prompt:

```
?- a.
```

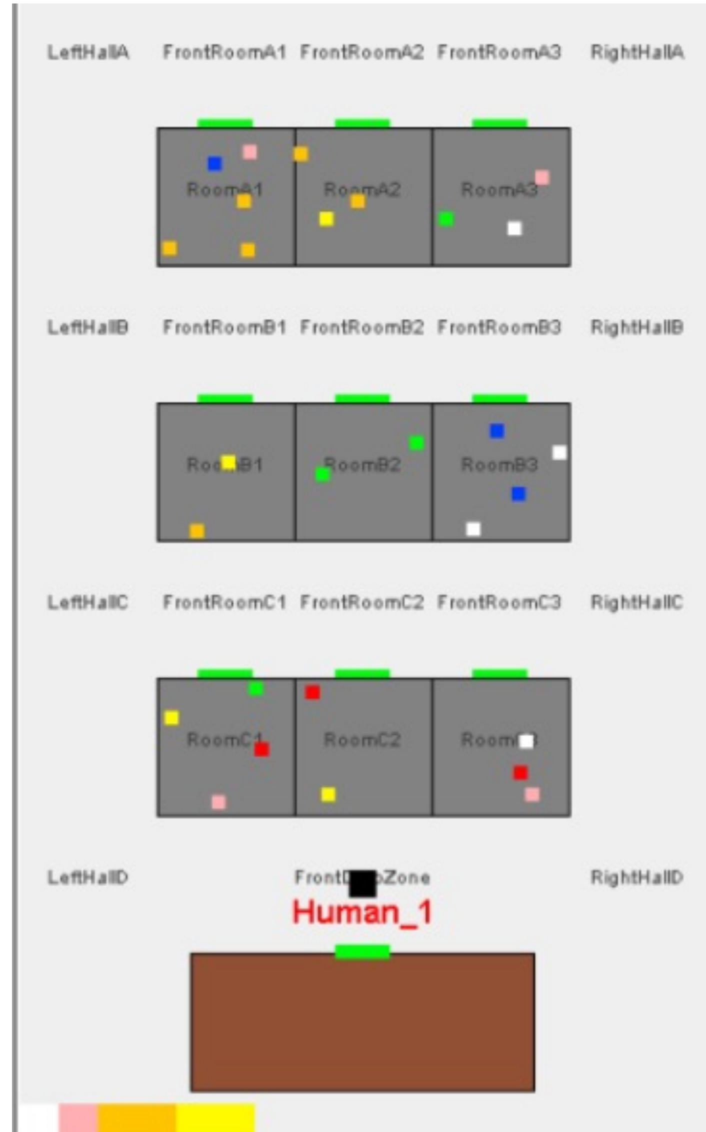
Prolog output:

```
true.
```

- By typing a goal that you want to prove in the prompt, you ask Prolog to try to derive it using the KB, i.e., execute the program.

Prolog as Knowledge Base (1)

- GOAL agent programming language
  - Koen Hindriks, VU



Prolog as Knowledge Base (2)

- Rules to model in the KB
  - Information about the environment on which actions depend, e.g.,
    - whether a block is clear, what the next block is to pick up, whether all rooms have been checked, derive higher-level concepts from more basic percepts