Fundamentals of Artificial Intelligence – Logical Agents

Matthias Althoff

TU München

Winter semester 2022/23

Organization

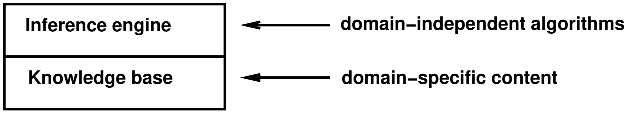
The content is covered in the AI book by the section “Logical Agents”.

Learning Outcomes

- You understand the difference between a *knowledge base* and an *inference engine*.
- You understand the difference between *syntax*, *semantics*, and *models*.
- You understand the difference between *satisfaction* and *entailment*.
- You can create and evaluate sentences in propositional logic.
- You can create truth tables of sentences in propositional logic.
- You can apply inference by enumeration.
- You understand the concepts *logical equivalence*, *validity*, and *satisfiability*.
- You can systematically apply theorem proving given a set of inference rules.
- You can apply *proof by resolution*.
- You can convert a sentence in propositional logic into *conjunctive normal form*.
- You can prove correctness of Horn clauses by *forward chaining* and *backward chaining*.

Knowledge Base

A **knowledge base** is a set of sentences in a formal language.



Possibilities to gain knowledge:

- **Inference**: Makes it possible to derive new knowledge from old knowledge.
- **Declarative approach**: New knowledge is added from "outside" by providing knowledge.
- **Perception**: New knowledge is added by the agent from its own perception.

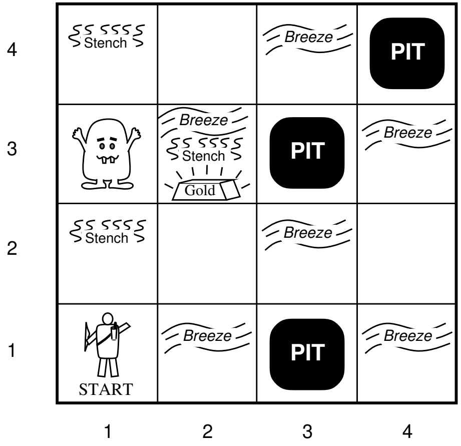
Agents can be viewed at the

- **knowledge level**: what they know, regardless of how implemented;
- **implementation level**: data structures in the knowledge base and algorithms that manipulate them.

The Wumpus World

We introduce the Wumpus world to demonstrate the benefits of knowledge (note that the previous search algorithms do not need knowledge).

- Cave consisting of rooms connected by passageways.
- Lurking somewhere is the terrible Wumpus, who eats anyone who enters his room.
- You have one arrow to shoot him before finding a heap of gold.

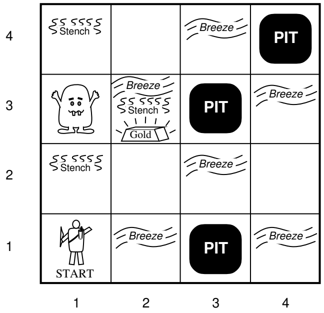


Tweedback Questions

Does a pure search technique exist so that we arrive at the gold without getting killed by Wumpus?

Wumpus World PEAS description

- **Performance measure**:
  gold +1000, death -1000
  -1 per step, -10 for using the arrow
- **Environment**:
  - Squares adjacent to Wumpus are smelly
  - Squares adjacent to pits are breezy
  - Glitter iff gold is in the same square
  - Shooting kills Wumpus if you are facing it
  - Shooting uses up the only arrow
  - Grabbing picks up gold if in same square
  - Releasing drops the gold in same square
- **Actuators**: left turn, right turn,
  forward, grab, release, shoot
- **Sensors**: breeze, glitter, smell

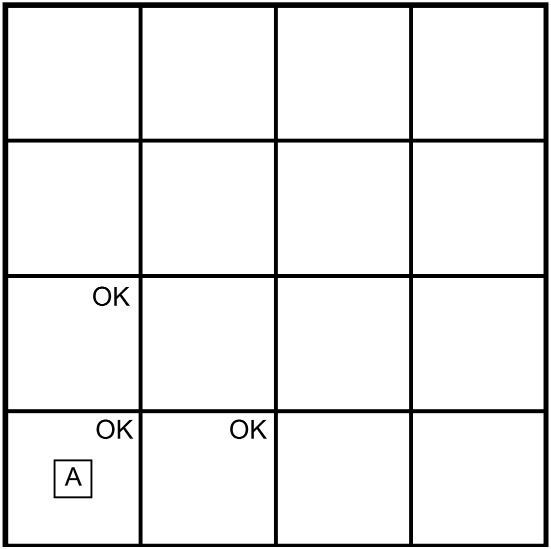


Wumpus World Characterization

- **Observable**: No – only local perception.
- **Deterministic**: Yes – outcomes are exactly specified.
- **Episodic**: No – sequential since actions change the environment.
- **Static**: Yes – Wumpus and pits do not move.
- **Discrete**: Yes.
- **Single-agent**: Yes – Wumpus is essentially a natural feature.

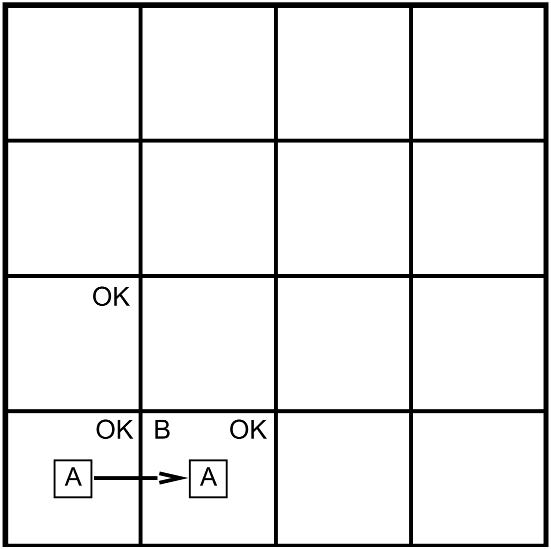
The main challenge is the initial ignorance of the environment; overcoming this ignorance seems to require logical reasoning.

Exploring a Wumpus World (1)



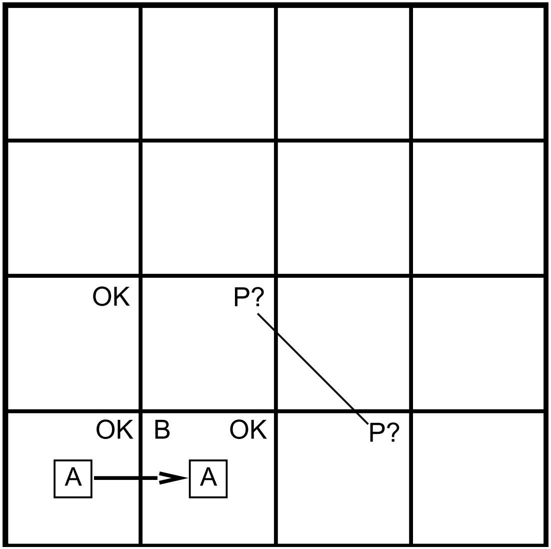
B: breeze,
S: stench,
G: glitter,
A: agent,
P: pit,
W: Wumpus,
OK: safe square.

Exploring a Wumpus World (2)



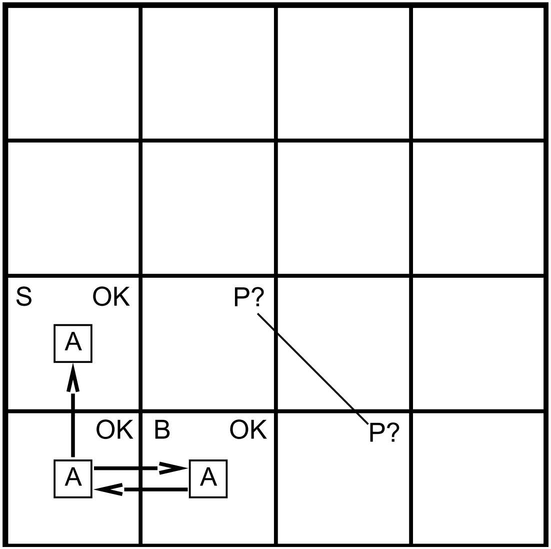
B: breeze,
S: stench,
G: glitter,
A: agent,
P: pit,
W: Wumpus,
OK: safe square.

Exploring a Wumpus World (3)



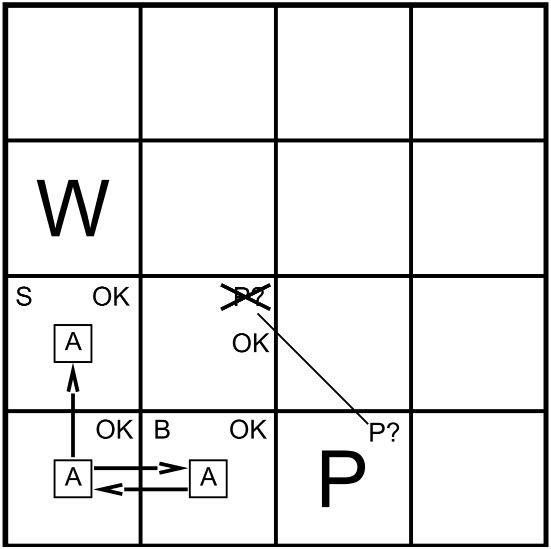
B: breeze,
S: stench,
G: glitter,
A: agent,
P: pit,
W: Wumpus,
OK: safe square.

Exploring a Wumpus World (4)



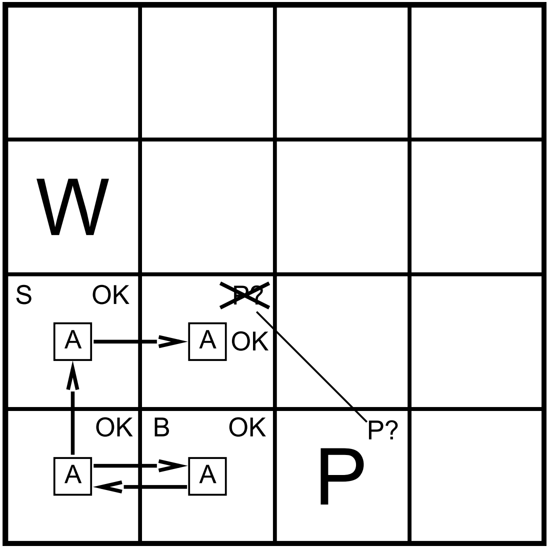
B: breeze,
S: stench,
G: glitter,
A: agent,
P: pit,
W: Wumpus,
OK: safe square.

Exploring a Wumpus World (5)



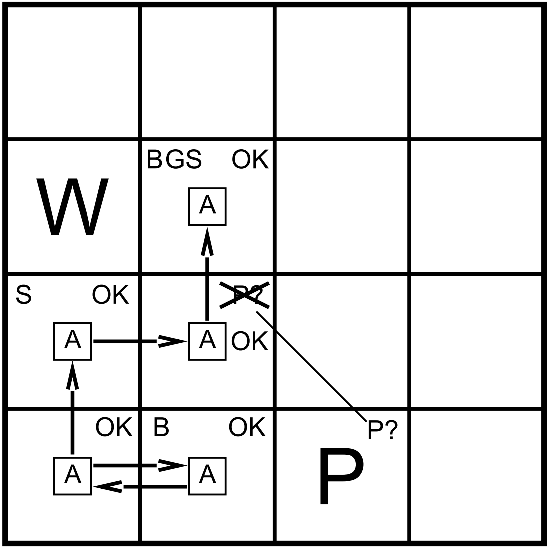
B: breeze,
S: stench,
G: glitter,
A: agent,
P: pit,
W: Wumpus,
OK: safe square.

Exploring a Wumpus World (6)



B: breeze,
S: stench,
G: glitter,
A: agent,
P: pit,
W: Wumpus,
OK: safe square.

Exploring a Wumpus World (7)



B: breeze,
S: stench,
G: glitter,
A: agent,
P: pit,
W: Wumpus,
OK: safe square.

Tweedback Questions

Search does not prevent us from getting killed.

Is it possible that even by using logic we get killed?

Basics of Logic (1)

The main concept of logic is explained based on ordinary arithmetic, which everybody is familiar with.

Syntax

Specifies how correct sentences are formed, e.g., $x + y = 4$ is well-formed, while $x4y+ =$ is not.

Semantics

The semantics defines the meaning of sentences, i.e., when a sentence is true. For instance, $x + y = 4$ is true for $x = y = 2$ and false for $x = y = 1$.

Model

Models are differently defined depending on the discipline. Here, models are instances which evaluate sentences to true or false. For instance, we have x men and y women playing a card game, then the sentence $x + y = 4$ is true for the models $x = 4, y = 0$; $x = 3, y = 1$; and so on.

Basics of Logic (2)

Satisfaction

If a sentence α is true in model *m*, we say that *m* **satisfies** α. We use the notation $M(\alpha)$ to mean the set of all models of α.

Entailment

Entailment is the relationship between two sentences where the truth of one sentence requires the truth of the other sentence, which is written as

$$\alpha \models \beta$$

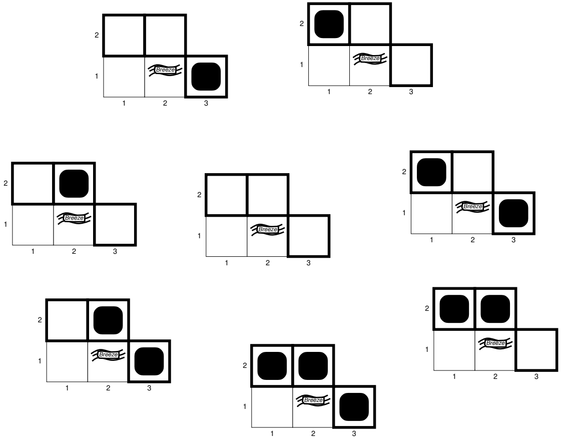
if α entails β. Formally, entailment is defined as

$$\alpha \models \beta \text{ if and only if } M(\alpha) \subseteq M(\beta).$$

For instance, the sentence $x = 0$ entails $xy = 0$.

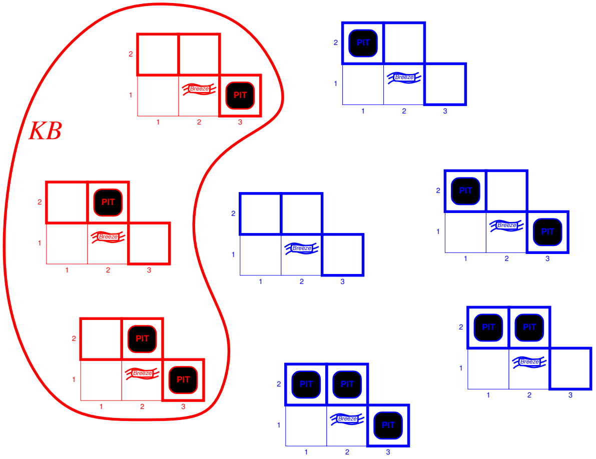
Logical Reasoning in the Wumpus World (1)

The agent in $[2, 1]$ is interested (among other things) whether the adjacent squares contain pits. Each of those squares might have or not have a pit, resulting in $2^3 = 8$ models:



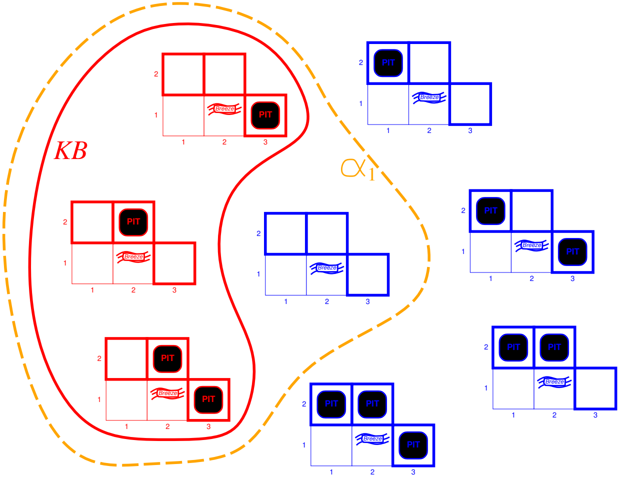
Logical Reasoning in the Wumpus World (2)

The knowledge base (KB) is a set of sentences. Models in which the knowledge base is true are shown below (the agent has only explored [1, 1] and [2, 1]):



Logical Reasoning in the Wumpus World (3)

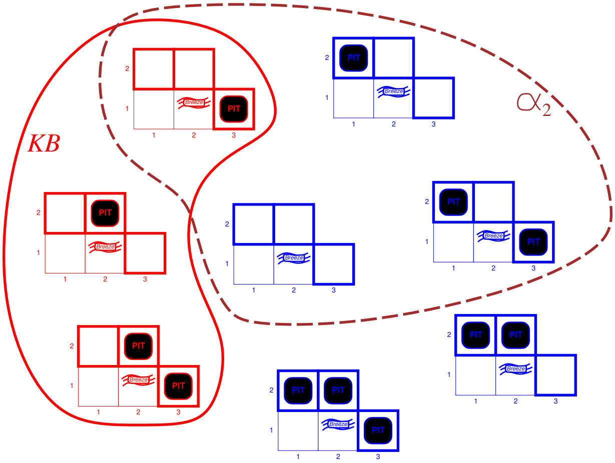
For what models is the sentence $\alpha_1 =$ "There is no pit in $[1, 2]$" true?



In every model, in which KB is true, α_1 is also true. Thus, $KB \models \alpha_1$.

Logical Reasoning in the Wumpus World (4)

For what models is the sentence $\alpha_2 =$ "There is no pit in $[2,2]$" true?



Not every model in which KB is true is α_2 also true. Thus, $KB \not\models \alpha_2$.

Syntax of Propositional Logic

We apply the aforementioned techniques to a particular logic: propositional logic, which is the simplest commonly-used logic.

The proposition symbols S_1, S_2, etc, are sentences.

- If S is a sentence, $\neg S$ 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**)

Backus-Naur Form (AP: atomic proposition, e.g., true, false, A, B, etc.)

$$S ::= AP | \neg S | S_1 \wedge S_2 | S_1 \vee S_2 | S_1 \Rightarrow S_2 | S_1 \Leftrightarrow S_2 | (S)$$

Operator precedence (descending order): $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$

Syntax of Propositional Logic: Examples

Reminder: Backus-Naur Form

$$S ::= AP | \neg S | S_1 \wedge S_2 | S_1 \vee S_2 | S_1 \Rightarrow S_2 | S_1 \Leftrightarrow S_2 | (S)$$

Operator precedence (descending order): $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$

- $true$ yes
- $true(\wedge S_1)$ no
- $S_1 \Rightarrow\Rightarrow S_2$ no
- $\neg\neg S$ yes
- $S_1 \Rightarrow (S_2 \Rightarrow S_3)$ yes
- $S_1 \neg \Rightarrow S_2$ no
- $S_1 \Rightarrow \neg S_2$ yes

Semantics of Propositional Logic

Each model specifies true/false for each proposition symbol,

e.g.,	$P_{1,2}$	$P_{2,2}$	$P_{3,1}$
	true	*false*	*true*

Rules for evaluating truth with respect to a model m:

$\neg S$	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
i.e.,	is false iff	S_1	is true *and*	S_2	is false
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$	is true *and*	$S_2 \Rightarrow S_1$	is true

- A simple recursive process evaluates an arbitrary sentence, e.g., $P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = \mathit{true} \wedge (\mathit{false} \vee \mathit{true}) = \mathit{true} \wedge \mathit{true} = \mathit{true}$
- Entailment vs. implication: $P \models Q$ if and only if the sentence $P \Rightarrow Q$ is always true for any model (i.e. it is a tautology, see later).

Truth Tables

The rules can also be expressed with a truth table that specifies the truth value for each possible assignment:

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

All assignments are intuitive, except the implication. The sentence

"5 is even implies Tokyo is the capital of Germany"

is true. Think of an implication $P \Rightarrow Q$ as saying

"If P is true, then I am claiming that Q is true."

Wumpus World Sentences

Symbols for each $[x, y]$ location

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

To derive $\neg P_{1,2}$ in the previous example, we need the following sentences R_i:

- There is no pit in $[1, 1]$: $\neg P_{1,1}$ (R_1)
- A square is breezy if and only if there is an adjacent pit. This has to be stated for all squares; we just include the relevant ones:

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

- The previous sentences are true in all Wumpus worlds. Now we introduce the percepts (particular to this world): $\neg B_{1,1}$ (R_4), $B_{2,1}$ (R_5).

Inference by Enumeration (1)

A simple technique to decide whether $KB \models \alpha$ is to enumerate all models and check whether α is true in every model in which KB is true.

Our example: the relevant proposition symbols are $B_{1,1}$, $B_{2,1}$, $P_{1,1}$, $P_{1,2}$, $P_{2,1}$, $P_{2,2}$, and $P_{3,1}$, resulting in $2^7 = 128$ models; in 3 cases, KB is true:

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

In those 3 models, $\neg P_{1,2}$ is true, such that $KB \models \neg P_{1,2}$.

Inference by Enumeration (2)

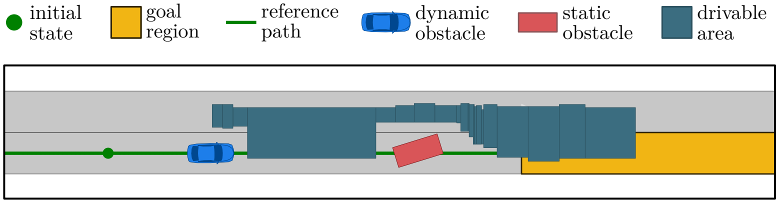
Not relevant for the exam

- If KB and α contain n symbols, there are 2^n models.
- Thus, the time complexity of enumeration is $\mathcal{O}(2^n)$.
- The space complexity is only $\mathcal{O}(n)$ because the enumeration is depth-first.
- Later we show algorithms that are more efficient on average. However, propositional entailment is co-NP-complete, so every known inference algorithm is exponential in the size of the input.
- The proposed technique is a special case of **Model Checking** (see lecture by Prof. Jan Kretinsky).

Example: Application in Automated Driving (1)

Not relevant for the exam

- Reachable sets of an automated vehicle are the set of states that can be reached by the vehicle over time.
- Reachable sets can be constrained by propositional logic to expedite the search for specification-compliant trajectories.
- We assume that a high-level maneuver planner provides specifications in propositional logic.



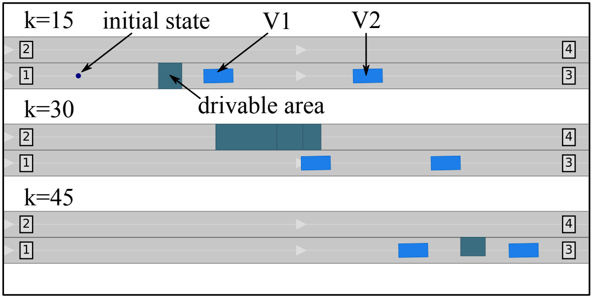
Example: Application in Automated Driving (2)

Not relevant for the exam

Overtaking maneuver

$$\mathbf{G}_{[0,15]}\,(\mathbf{Behind}(V_1) \land \mathbf{AlignedWith}(V_1)) \land$$
$$\mathbf{G}_{[16,39]}\,(\mathbf{InLanelet}(L_2) \lor \mathbf{InLanelet}(L_4)) \land$$
$$\mathbf{G}_{[40,45]}\,(\mathbf{InFrontOf}(V_1) \land \mathbf{Behind}(V_2) \land \mathbf{InLanelet}(L_3))$$

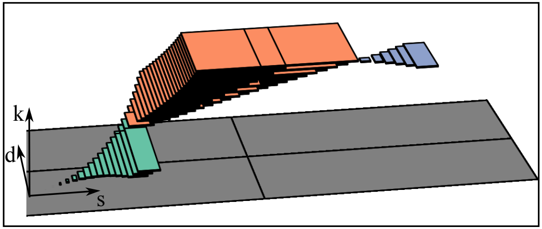
$\mathbf{G}_{[a,b]}$ is syntactic sugar specifying time steps for which the propositions should hold.

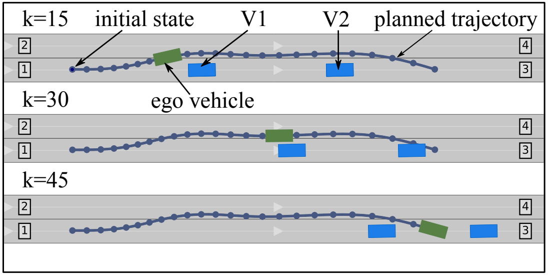


Example: Application in Automated Driving (3)

Not relevant for the exam

Reachable set over time and exemplary trajectory planned therein:





Introduction to Theorem Proving (1)

- Instead of using enumeration, we apply rules of inference directly to sentences in theorem proving.
- Theorem proving does not require any models!
- If the number of models is large, but the length of the proof is short, theorem proving can be more efficient than enumeration.

We require some concepts for theorem proving:

Logical equivalence

Two sentences α and β are logically equivalent if they are true in the same set of models, which is written as $\alpha \equiv \beta$. Alternative definition:

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

Introduction to Theorem Proving (2)

Validity

A sentence is valid if it is true in **all** models (e.g., $P \vee \neg P$). Valid sentences are also known as **tautologies**.

Satisfiability

A sentence is satisfiable if it is true in **some** model, e.g., the expression $P_1 \wedge P_2$ is satisfiable for $P_1 = P_2 = \mathit{true}$, whereas $P_1 \wedge \neg P_1$ is not satisfiable.

- The problem of determining the satisfiability of sentences is also called a **SAT** problem, which is NP-complete.
- Validity and satisfiability are connected: α is valid if $\neg\alpha$ is unsatisfiable.

Not for the Exam

Inference and Proofs

We discuss useful **inference rules** that can be applied to derive a **proof** – a chain of conclusions that lead to the desired goal.

Modus Ponens

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

The notation means that when $\alpha \Rightarrow \beta$ and α are given, β can be inferred.

And-Elimination

$$\frac{\alpha \land \beta}{\alpha}$$

Further inference rules can be obtained by using well-known logical equivalences (see next slide).

Logical Equivalences

Standard logical equivalences

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

Tweedback Question

How can we prove the above equivalences?

A We can prove these equivalences by other yet-to-be proven equivalences of the list.

B We can show the correctness by enumeration.

Inference from Equivalences

From the previous table, we can generate from *bidirectional elimination* the inference rules

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)}, \quad \frac{(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}.$$

The inference rule works in both directions due to the equivalence. This is not possible in general, e.g., Modus Ponens does not work in the opposite direction to obtain $\alpha \Rightarrow \beta$ and α from β.

Proof for the Wumpus World Example

1. We start with the sentence on slide 27:
   $R_2: \quad B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
2. Bidirectional elimination (see slide 36):
   $R_6: \quad (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
3. And-Elimination (see slide 35):
   $R_7: \quad ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
4. Contraposition (see slide 36):
   $R_8: \quad (\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1}))$
5. Modus Ponens (see slide 35) with R_8 and $R_4 = \neg B_{1,1}$:
   $R_9: \quad \neg(P_{1,2} \vee P_{2,1})$
6. De Morgan (see slide 36):
   $R_{10}: \quad \neg P_{1,2} \wedge \neg P_{2,1}$
7. Thus, neither $[1,2]$ nor $[2,1]$ contains a pit.

Automated Theorem Proving

The previous method was done “by hand”. How can one automate this?

We can use the previously introduced search methods on the following problem:

- **Initial state**: the initial knowledge base.
- **Actions**: all the inference rules applied to all the sentences that match the top half of the inference rule.
- **Result**: the result of an action is to add the sentence in the bottom half of the inference rule.
- **Goal**: a state that contains the sentence to prove.

In practical cases, finding a proof can be more efficient than enumeration because not all possible models have to be generated.

Proof by Resolution (InferencePropLogic.ipynb)

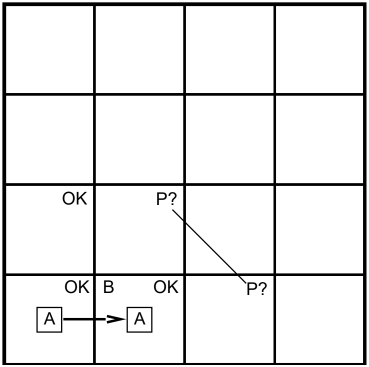
- So far, we have not discussed completeness, i.e., does the algorithm find a proof if one exists?
- For instance, the previous proof does not work without the *bidirectional elimination*.
- We introduce the inference rule **resolution**, which yields a complete inference algorithm when coupled with a complete search algorithm.

We begin with a

1. Wumpus world example,
2. generalize it,
3. and prove why resolution leads to a complete algorithm when using propositional logic.

Resolution in the Wumpus World (1)

We start with the following situation:



We add the following facts to the knowledge base:

$R_{11}: \quad \neg B_{1,2}$

$R_{12}: \quad B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3})$

Resolution in the Wumpus World (2)

1. By the same process that led to R_{10} on slide 39, we can derive the absence of pits in $[2, 2]$ and $[1, 3]$:
   $R_{13}: \quad \neg P_{2,2}$
   $R_{14}: \quad \neg P_{1,3}$
2. Bidirectional elimination to R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$ (slide 27), followed by Modus Ponens with R_5: $B_{2,1}$ (slide 27) yields
   $R_{15}: \quad P_{1,1} \vee P_{2,2} \vee P_{3,1}$
3. Now comes the first resolution rule: $\neg P_{2,2}$ in R_{13} resolves with $P_{2,2}$ in R_{15} to give the **resolvent**
   $R_{16}: \quad P_{1,1} \vee P_{3,1}$
4. Similarly, R_1: $\neg P_{1,1}$ (slide 27) resolves with $P_{1,1}$ in R_{16} to
   $R_{17}: \quad P_{3,1}$
5. Now we know that the pit can only be in $[3, 1]$!

Resolution Inference Rules

Unit resolution rule

Given literals l_i (atomic proposition or its negation) we have that

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

where l_i and m are **complementary literals** (i.e., $l_i \equiv \neg m$).

The unit resolution rule can be generalized:

Full resolution rule

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

where l_i and m_j are complementary literals.

Example: $\frac{P_{1,1} \vee P_{3,1}, \quad \neg P_{1,1} \vee \neg P_{2,2}}{P_{3,1} \vee \neg P_{2,2}}$.

Soundness of the Resolution Rule

We discuss the soundness of

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

informally:

- $\mathrm{l_i}$ **is true and** $\mathrm{m_j}$ **is false**:
  Hence, $m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n$ must be true, because $m_1 \vee \ldots \vee m_n$ is given.
- $\mathrm{m_j}$ **is true and** $\mathrm{l_i}$ **is false**:
  Hence, $l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k$ must be true, because $l_1 \vee \ldots \vee l_k$ is given.
- $\mathrm{l_i}$ **is either true or false**, so one of these conclusions holds, as stated in the resolution rule.

Conjunctive Normal Form

- The resolution rule only applies to disjunction of literals, which are also called **clauses**.
- Fortunately, every sentence of propositional logic can be reformulated as a conjunction of clauses, which is also referred to as **conjunctive normal form (CNF)**

Conjunctive Normal Form

A sentence with literals x_{ij} of the form $\bigwedge_i \bigvee_j (\neg) x_{ij}$ is in conjunctive normal form.

Examples:

- $(A \vee B \vee C) \wedge (\neg A \vee B \vee C)$ yes
- $A \wedge B \wedge C \vee (\neg A \wedge B \vee C)$ no
- $A \wedge B \wedge C \wedge (\neg A \vee B \vee C)$ yes

Conversion to CNF

We demonstrate the conversion to CNF by converting $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$:

1. Eliminate $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$:
$$\Big(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})\Big) \wedge \Big((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}\Big)$$
2. Eliminate $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$:
$$\Big(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}\Big) \wedge \Big(\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1}\Big)$$
3. "Moving $\neg$ inwards" by application of the following equivalences (see slide 36)
$$\neg(\neg\alpha) \equiv \alpha \qquad \text{(double-negation elimination)}$$
$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \qquad \text{(De Morgan)}$$
$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \qquad \text{(De Morgan)}$$
We only require the last rule in the example:
$$\Big(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}\Big) \wedge \Big((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1}\Big)$$
4. Now we only have nested $\wedge$ and $\vee$ operators applied to literals. It remains to swap $\wedge$ and $\vee$ using the distributivity law:
$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

A Resolution Algorithm

Inference procedures based on resolution use the principle of **proof by contradiction**:

To show that $KB \models \alpha$, we show that $KB \land \neg\alpha$ is unsatisfiable.

Basic procedure

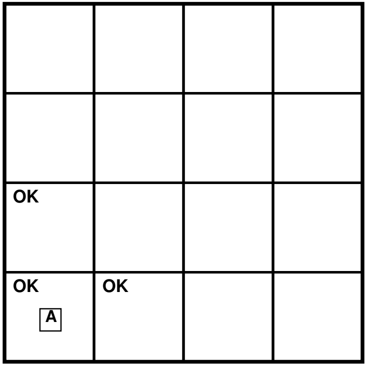
1. $KB \land \neg\alpha$ is converted into CNF
2. The resolution rule is applied to the resulting clauses: each pair that contains complementary literals is resolved to produce a new clause, which is added to the others (if not already present)
3. The process continues until
   - there are no new clauses to be added $\Rightarrow KB \not\models \alpha$;
   - two clauses resolve to yield the *empty* clause $\Rightarrow KB \models \alpha$.

Example of the Resolution Algorithm (1)

Wumpus World: the agent is in [1,1] and there is no breeze, so there can be no pits in the neighboring squares. The knowledge base is

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

and we would like to prove $\alpha = \neg P_{1,2}$



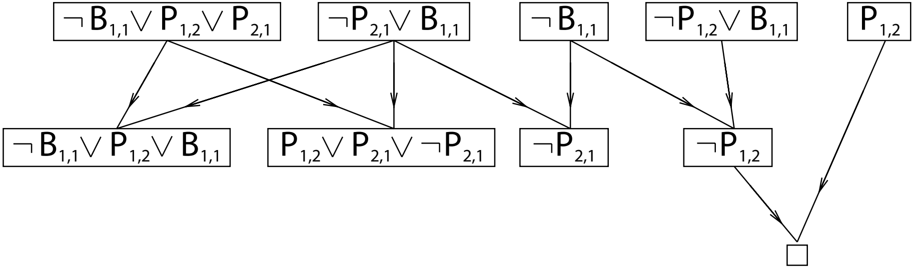
Example of the Resolution Algorithm (2)

We start with the conversion of $KB \land \neg\alpha = (B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1} \land P_{1,2}$ into CNF:

1. Eliminate $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$:
$$\Big(B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})\Big) \land \Big((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1}\Big) \land \neg B_{1,1} \land P_{1,2}$$
2. Eliminate $\alpha \Rightarrow \beta$ with $\neg\alpha \lor \beta$:
$$\Big(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}\Big) \land \Big(\neg(P_{1,2} \lor P_{2,1}) \lor B_{1,1}\Big) \land \neg B_{1,1} \land P_{1,2}$$
3. "Moving $\neg$ inwards" (see slide 36):
$$\Big(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}\Big) \land \Big((\neg P_{1,2} \land \neg P_{2,1}) \lor B_{1,1}\Big) \land \neg B_{1,1} \land P_{1,2}$$
4. Now we only have nested $\land$ and $\lor$ operators applied to literals. It remains to swap $\land$ and $\lor$ using the distributivity law:
$(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1}) \land \neg B_{1,1} \land P_{1,2}$

Example of the Resolution Algorithm (3)

When we convert $KB \wedge \neg\alpha$ into CNF, we obtain the clauses on the top:


$\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}$
$\neg P_{2,1} \vee B_{1,1}$
$\neg B_{1,1}$
$\neg P_{1,2} \vee B_{1,1}$
$P_{1,2}$
$\neg B_{1,1} \vee P_{1,2} \vee B_{1,1}$
$P_{1,2} \vee P_{2,1} \vee \neg P_{2,1}$
$\neg P_{2,1}$
$\neg P_{1,2}$
$\square$

The second row of the figure shows clauses obtained by resolving pairs.

We obtain the empty clause by resolving $P_{1,2}$ with $\neg P_{1,2}$, so that $KB \models \alpha$

Resolution Algorithm

```
function PL-Resolution ($KB$, $\alpha$) returns true, or false
clauses $\leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
new $\leftarrow$ { }
loop do
    for each pair of clauses $C_i$,$C_j$ in clauses do
        resolvents $\leftarrow$ PL-Resolve($C_i$,$C_j$)
        if resolvents contains the empty clause then return true
        new $\leftarrow$ new $\cup$ resolvents
    if new $\subseteq$ clauses then return false
    clauses $\leftarrow$ clauses $\cup$ new
```

Completeness of Resolution

It remains to show why resolution is complete for propositional logic.

Resolution closure

The **resolution closure** $RC(S)$ of a set of clauses S is the set of all clauses derivable by repeated application of the resolution rule to S and its derivatives.

- $RC(S)$ is finite, because there are only finitely many distinct clauses that can be constructed out of the symbols P_1, …, P_k.
- Hence, PL-resolution always terminates.

Ground resolution theorem

If a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clause.

Proof of the Ground Resolution Theorem (1)

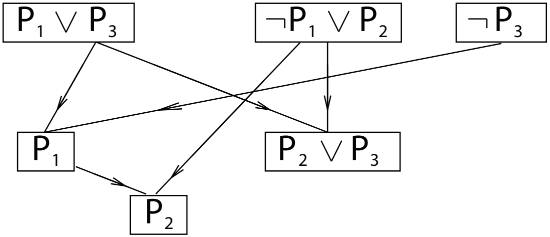
Not relevant for the exam

The theorem is proven by its contrapositive ($\alpha \Rightarrow \beta \equiv \neg\beta \Rightarrow \neg\alpha$): if the closure $RC(S)$ does **not** contain the empty clause, then S is satisfiable.

We can construct a model for S with suitable truth values for $P_1, \ldots, P_k$:

For i from 1 to k:

- If a clause in $RC(S)$ contains the literal $\neg P_i$ and all its other literals are false under the assignment chosen for $P_1, \ldots, P_{i-1}$, then assign *false* to P_i.
- Otherwise assign *true* to P_i.



Proof of the Ground Resolution Theorem (2)

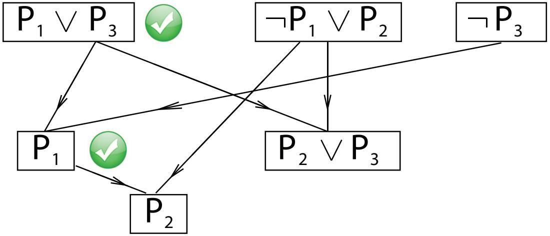
Not relevant for the exam

The theorem is proven by its contrapositive ($\alpha \Rightarrow \beta \equiv \neg\beta \Rightarrow \neg\alpha$): if the closure $RC(S)$ does **not** contain the empty clause, then S is satisfiable.

We can construct a model for S with suitable truth values for $P_1, \ldots, P_k$:

For i from 1 to k:

- If a clause in $RC(S)$ contains the literal $\neg P_i$ and all its other literals are false under the assignment chosen for $P_1, \ldots, P_{i-1}$, then assign *false* to P_i.
- Otherwise assign *true* to P_i.



P_1 : *true*

Proof of the Ground Resolution Theorem (3)

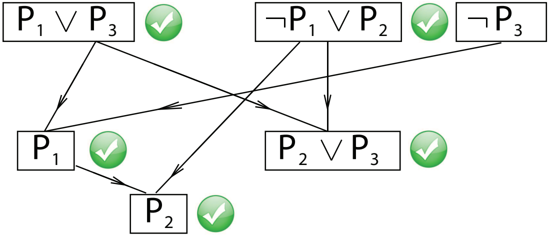
Not relevant for the exam

The theorem is proven by its contrapositive ($\alpha \Rightarrow \beta \equiv \neg\beta \Rightarrow \neg\alpha$): if the closure $RC(S)$ does **not** contain the empty clause, then S is satisfiable.

We can construct a model for S with suitable truth values for $P_1, \ldots, P_k$:

For i from 1 to k:

- If a clause in $RC(S)$ contains the literal $\neg P_i$ and all its other literals are false under the assignment chosen for $P_1, \ldots, P_{i-1}$, then assign *false* to P_i.
- Otherwise assign *true* to P_i.



P_1 : *true*
P_2 : *true*

Proof of the Ground Resolution Theorem (4)

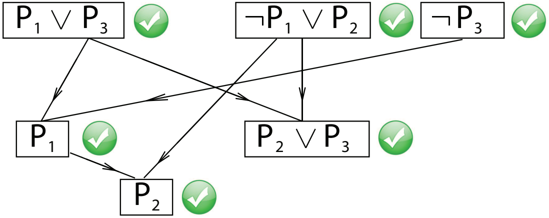
Not relevant for the exam

The theorem is proven by its contrapositive ($\alpha \Rightarrow \beta \equiv \neg\beta \Rightarrow \neg\alpha$): if the closure $RC(S)$ does **not** contain the empty clause, then S is satisfiable.

We can construct a model for S with suitable truth values for $P_1, \ldots, P_k$:

For i from 1 to k:

- If a clause in $RC(S)$ contains the literal $\neg P_i$ and all its other literals are false under the assignment chosen for $P_1, \ldots, P_{i-1}$, then assign *false* to P_i.
- Otherwise assign *true* to P_i.



P_1 : *true*
P_2 : *true*
P_3 : *false*

Proof of the Ground Resolution Theorem (5)

Not relevant for the exam

The theorem is proven by its contrapositive ($\alpha \Rightarrow \beta \equiv \neg\beta \Rightarrow \neg\alpha$): if the closure $RC(S)$ does **not** contain the empty clause, then S is satisfiable.

We can construct a model for S with suitable truth values for $P_1, \dots, P_k$:

For i from 1 to k:

- If a clause in $RC(S)$ contains the literal $\neg P_i$ and all its other literals are false under the assignment chosen for $P_1, \dots, P_{i-1}$, then assign *false* to P_i.
- Otherwise assign *true* to P_i.

- This assignment is a model of S. To see this, assume the opposite – a clause becomes false at stage i when all its literals are false:

$$(\mathit{false} \vee \mathit{false} \vee \cdots \mathit{false} \vee P_i) \text{ or } (\mathit{false} \vee \mathit{false} \vee \cdots \mathit{false} \vee \neg P_i).$$

  The model construction will choose the truth value for P_i such that the clause is true. The clause can only be falsified if **both** clauses are in $RC(S)$. Since $RC(S)$ is closed under resolution, it will contain the resolvent, whose literals $P_1, \dots, P_{i-1}$ are all false by assignment.
- This contradicts the assumption that the first falsified clause appears at stage i.
- Hence, we have proven that the construction never falsifies a clause in $RC(S)$.

Horn Clauses

Some simple forms of sentences do not require proof by resolution. We introduce Horn clauses for which very efficient inference algorithms exist.

Horn clause

- proposition symbol; or
- (conjunction of symbols) $\Rightarrow$ symbol

Which are Horn clauses?

- $(L_{1,1} \wedge Breeze) \Rightarrow B_{1,1}$ yes
- $L_{1,1}$ yes $(\equiv true \Rightarrow L_{1,1})$
- $(L_{1,1} \vee Breeze) \Rightarrow B_{1,1}$ no

A knowledge base consisting of Horn clauses only requires Modus Ponens as an inference method:

$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

AND-OR Graph

Forward chaining is best illustrated by an **AND-OR** graph.

AND-OR graph

- Links joined by an arc indicate a conjunction: every link must be proven
- Links joined without an arc indicate a disjunction: only one link has to be proven

The knowledge base and the corresponding AND-OR graph:

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

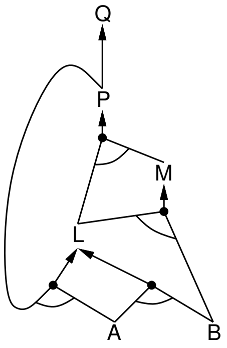
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

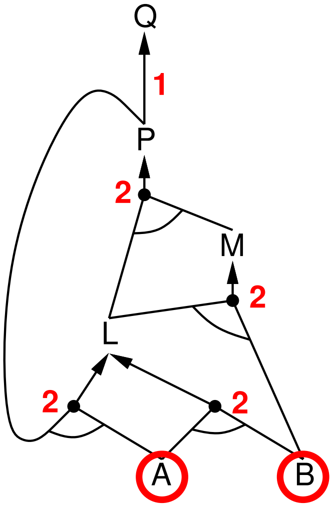


Forward Chaining (1)

1. Fire any rule whose premises are satisfied in the *KB*,
2. add its conclusion to the *KB*,
3. until query is found.

Forward chaining time complexity is only linear!

(red circle: frontier; red filling: explored)

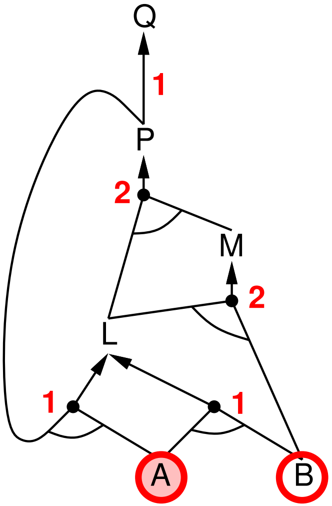


Forward Chaining (2)

1. Fire any rule whose premises are satisfied in the *KB*,
2. add its conclusion to the *KB*,
3. until query is found.

Forward chaining time complexity is only linear!

(red circle: frontier; red filling: explored)

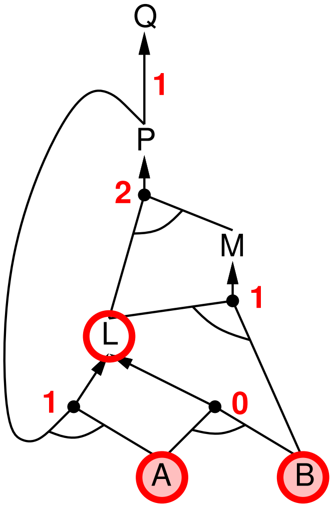


Forward Chaining (3)

1. Fire any rule whose premises are satisfied in the KB,
2. add its conclusion to the KB,
3. until query is found.

Forward chaining time complexity is only linear!

(red circle: frontier; red filling: explored)

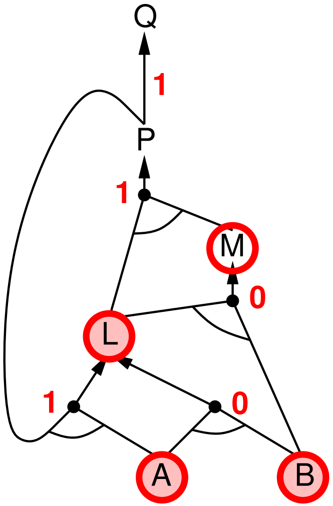


Forward Chaining (4)

1. Fire any rule whose premises are satisfied in the *KB*,
2. add its conclusion to the *KB*,
3. until query is found.

Forward chaining time complexity is only linear!

(red circle: frontier; red filling: explored)

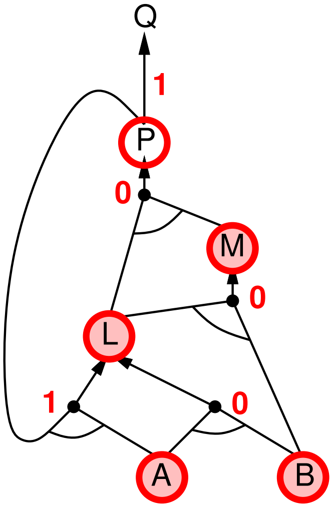


Forward Chaining (5)

1. Fire any rule whose premises are satisfied in the *KB*,
2. add its conclusion to the *KB*,
3. until query is found.

Forward chaining time complexity is only linear!

(red circle: frontier; red filling: explored)

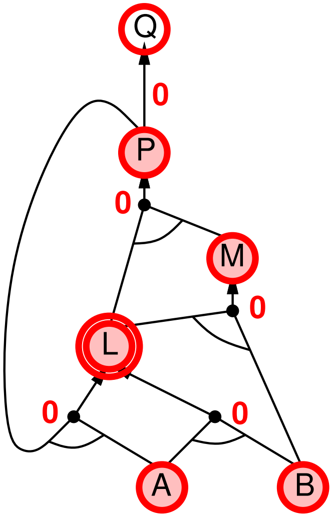


Forward Chaining (6)

1. Fire any rule whose premises are satisfied in the *KB*,
2. add its conclusion to the *KB*,
3. until query is found.

Forward chaining time complexity is only linear!

(red circle: frontier; red filling: explored)

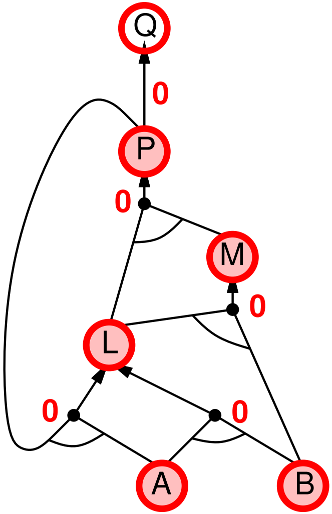


Forward Chaining (7)

1. Fire any rule whose premises are satisfied in the *KB*,
2. add its conclusion to the *KB*,
3. until query is found.

Forward chaining time complexity is only linear!

(red circle: frontier; red filling: explored)

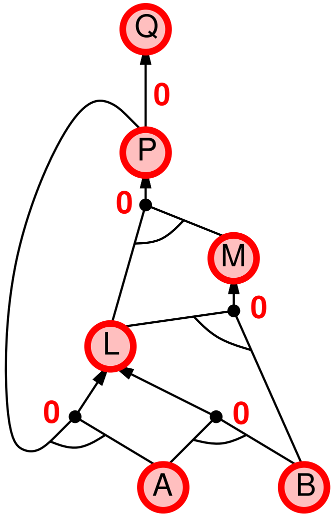


Forward Chaining (8)

1. Fire any rule whose premises are satisfied in the *KB*,
2. add its conclusion to the *KB*,
3. until query is found.

Forward chaining time complexity is only linear!

(red circle: frontier; red filling: explored)



Backward Chaining

Idea: work backwards from the query q:
to prove q by backward chaining,

- check if q is known already, or
- prove by backward chaining all premises of some rule concluding q.

Avoid loops: check if new subgoal is already on the goal stack.

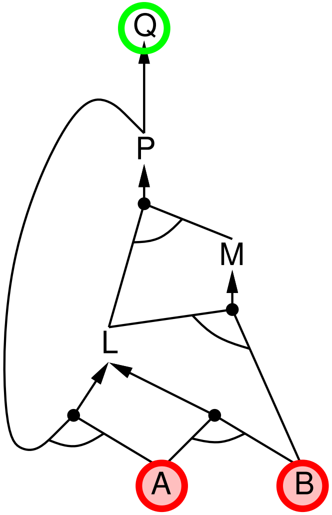
Avoid repeated work: check if new subgoal

- has already been proven true, or
- has already failed.

Backward Chaining: Example (1)

Backward chaining time complexity is also only linear!

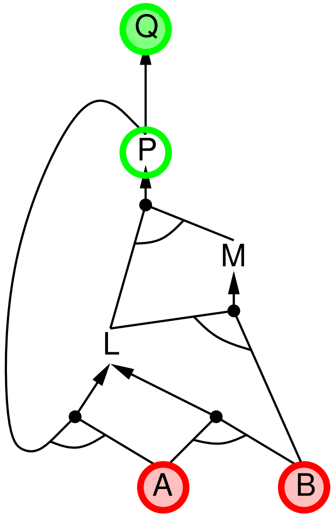
(green circle: frontier;
green filling: explored;
red filling: inferred, known as true)



Backward Chaining: Example (2)

Backward chaining time complexity is also only linear!

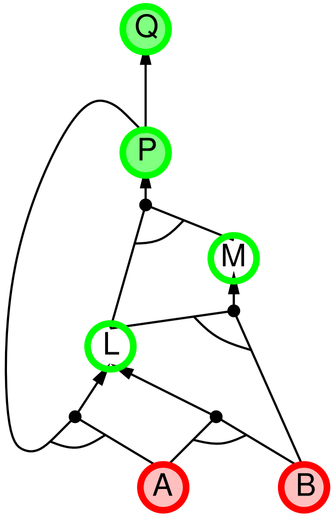
(green circle: frontier;
green filling: explored;
red filling: inferred, known as true)



Backward Chaining: Example (3)

Backward chaining time complexity is also only linear!

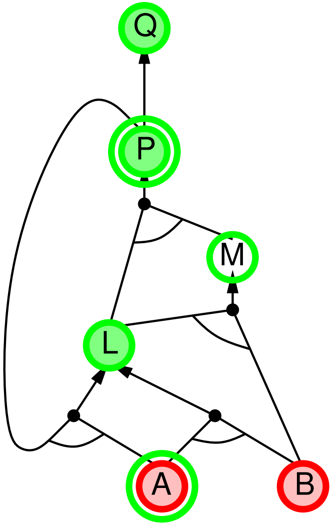
(green circle: frontier;
green filling: explored;
red filling: inferred, known as true)



Backward Chaining: Example (4)

Backward chaining time complexity is also only linear!

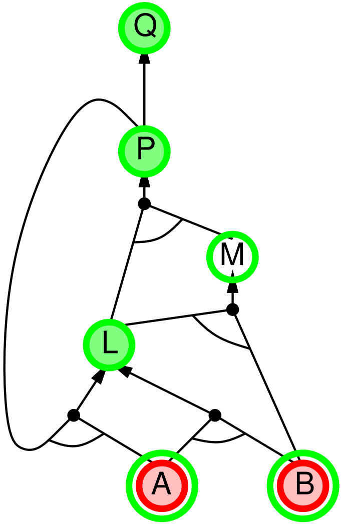
(green circle: frontier;
green filling: explored;
red filling: inferred, known as true)



Backward Chaining: Example (5)

Backward chaining time complexity is also only linear!

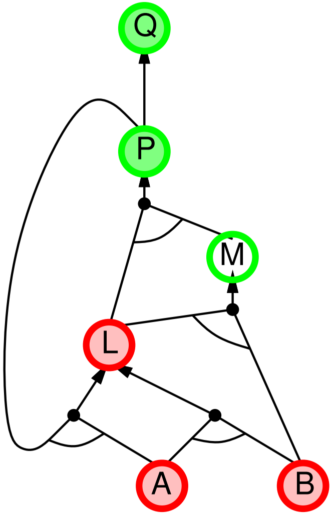
(green circle: frontier;
green filling: explored;
red filling: inferred, known as true)



Backward Chaining: Example (6)

Backward chaining time complexity is also only linear!

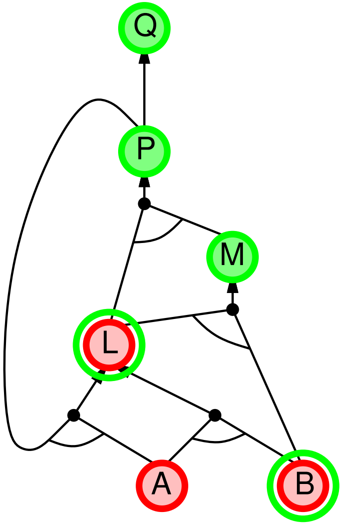
(green circle: frontier;
green filling: explored;
red filling: inferred, known as true)



Backward Chaining: Example (7)

Backward chaining time complexity is also only linear!

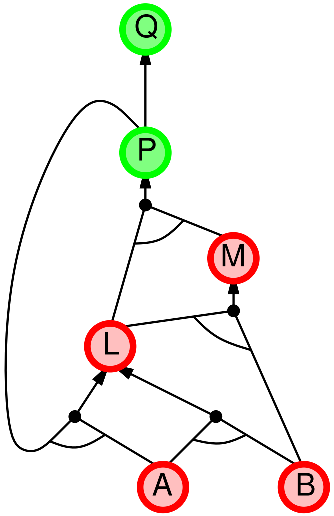
(green circle: frontier;
green filling: explored;
red filling: inferred, known as true)



Backward Chaining: Example (8)

Backward chaining time complexity is also only linear!

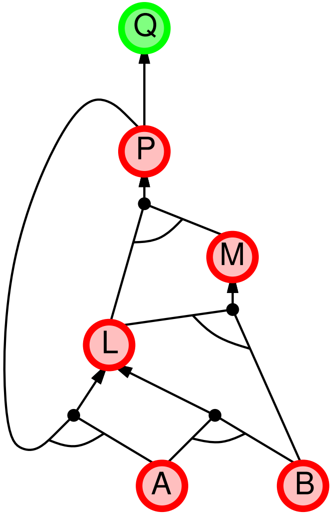
(green circle: frontier;
green filling: explored;
red filling: inferred, known as true)



Backward Chaining: Example (9)

Backward chaining time complexity is also only linear!

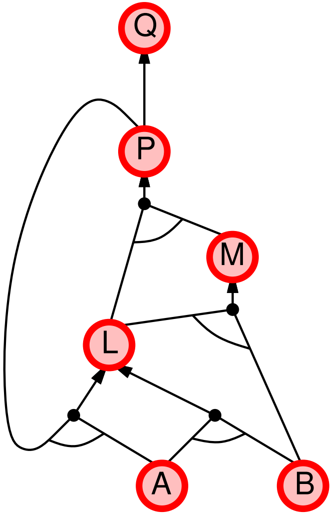
(green circle: frontier;
green filling: explored;
red filling: inferred, known as true)



Backward Chaining: Example (10)

Backward chaining time complexity is also only linear!

(green circle: frontier;
green filling: explored;
red filling: inferred, known as true)



Forward vs. Backward Chaining

Forward chaining

- Forward chaining is **data-driven**, automatic, and unconsciously processing.

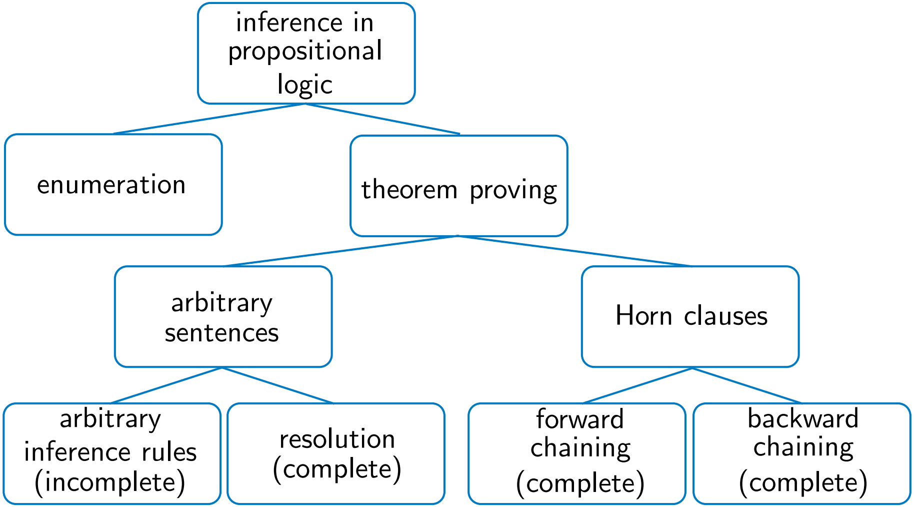
  It is popular in e.g., object recognition and routine decisions.
- Forward chaining may do lots of work that is irrelevant to the goal.

Backward chaining

- Backward chaining is **goal-driven** and appropriate for problem-solving.

  It is a good choice for problems, such as e.g., Where are my keys? How do I cook a meal?
- Computational effort of backward chaining can be *much less* than linear in time and space.

Overview of Inference Methods


inference in propositional logic
enumeration
theorem proving
arbitrary sentences
Horn clauses
arbitrary inference rules (incomplete)
resolution (complete)
forward chaining (complete)
backward chaining (complete)

Summary

- Intelligent agents need knowledge about the world in order to reach good solutions.
- Knowledge is contained in agents in the form of **sentences** in a **knowledge representation language** that are stored in a **knowledge base**.
- A knowledge-based agent is composed of a knowledge base and an inference mechanism, which infers new sentences for decision making.
- The set of possible models for propositional logic is finite, so entailment can be checked by enumerating models.
- **Inference rules** are patterns of sound inference to find proofs. The **resolution** rule yields a complete inference algorithm for knowledge bases in **conjunctive normal form**. **Forward and backward chaining** are natural reasoning algorithms for **Horn clauses**.