# VNU-HUS MAT1206E/3508: Introduction to AI

## Limitations of Logic

Hoàng Anh Đức

Bộ môn Tin học, Khoa Toán-Cơ-Tin học
Đại học KHTN, ĐHQG Hà Nội
hoanganhduc@hus.edu.vn

# Contents

# The Search Space Problem

- In the search for a proof, depending on the calculus, potentially *there are (infinitely) many ways to apply inference rules at each step*
- This is the main reason for the *explosive growth of the search space*


Because of the search space problem, *automated provers* today can *only prove relatively simple theorems in special domains with few axioms*

# The Search Space Problem

| | Automated Provers | Human Experts |
|---|---|---|
| **Number of inferences performed per second** | 20000 | 1 |
| **Solve difficult problems** | slow | fast |

**Reasons**

- Humans use *intuitive calculi* that work on a higher level
  - This intuitive calculi often *carry out many of the simple inferences of an automated prover in one step*
- Humans work with *lemmas*
  - We already know the lemmas are true and *do not need to re-prove* them each time
- Humans use *intuitive meta knowledge* (informal!)
  - Intuition is an important advantage of humans. Without it, we could not solve any difficult problems
- Humans use *heuristics*
  - In many cases, heuristics can greatly simplify or shorten the way to the goal

# The Search Space Problem



**Problems**

- Often humans are *unable to formulate intuitive meta-knowledge verbally*!
- Humans *learn heuristics by experience*

**Solution 1**

Try to “copy nature”

- Learn heuristics by the application of machine learning techniques

**Solution 2**

Design interactive systems that operate under the control of the user

- For example, computer algebra programs such as Mathematica, Maple, or Maxima can automatically carry out difficult symbolic mathematical manipulations
- The search for the proof is left fully to the human

# The Search Space Problem



## Example 1 (Resolution + Machine Learning Techniques)

A *resolution prover* has, during the search for a proof, *hundreds or more possibilities for resolution steps at each step*, but *only a few lead to the goal*. It would be *ideal* if the prover could *ask an oracle which two clauses it should use in the next step to quickly find the proof*

(1) A *proof-directing module evaluates* the different alternatives for the next step heuristically and *chooses* the alternative with the *best rating*
   - Rating the available clauses by *a function that calculates a value based on the number of literals, the number of positive literals, the complexity of the terms, etc.* for every pair of resolvable clauses

(2) How to *compute* such a *function*?
   - Use machine learning algorithms to learn from successful proofs
   - Successful resolution steps are stored as positive
   - Unsuccessful resolution steps are stored as negative
   - Machine learning system generates a program for the evaluation of clauses

# Decidability and Incompleteness





- There are *correct and complete calculi and theorem provers*
- Any *theorem (i.e., a true statement)* can be *proved in a finite amount of time*
- What if *the statement is not true*?
  - There is *no process* that can *prove or refute any formula from PL1 in finite time*

## Theorem 1

*The set of valid formulas in first-order predicate logic is semidecidable*

- This theorem implies that there are programs (theorem provers) which, given a true (valid) formula as input, determine its truth in finite time
- If the formula is not valid, however, it may happen that the prover never halts

# Decidability and Incompleteness





**Note**

- *Propositional logic decidable* because *the truth table method provides all models of a formula in finite time*
- Evidently predicate logic with quantifiers and nested function symbols is a language somewhat too powerful to be decidable

## Exercise 1 ([Ertel 2025], Exercise 4.1, p. 75)

(a) With the following (false) argument, one could claim that PL1 is decidable: *"We take a complete proof calculus for PL1. With it we can find a proof for any true formula in finite time. For every other formula $\phi$ I proceed as follows: I apply the calculus to $\neg\phi$ and show that $\neg\phi$ is true. Thus $\phi$ is false. Thus I can prove or refute every formula in PL1."* Find the mistake in the argument and change it so it becomes correct (**Hint:** There are three types of formulas: satisfiable, valid, and unsatisfiable. Can the above process be applied for all of these types?)

(b) Construct a decision process for the set of true and unsatisfiable formulas in PL1

# Decidability and Incompleteness



**Higher-order Logic**

- A *first-order logic* can only *quantify over variables*
- A *second-order logic* can also *quantify over formulas of the first order*
- A *third-order logic* can *quantify over formulas of the second order*
- and so on

## Example 2 (Second-order Logic Formula)

- If a predicate $p(n)$ holds for $n$, then $p(n+1)$ also holds
- $\forall p\, p(n) \Rightarrow p(n+1)$

# Decidability and Incompleteness





## Theorem 2 (Gödel incompleteness theorem)

*Every axiom system for the natural numbers with addition and multiplication (arithmetic) is incomplete. That is, there are true statements in arithmetic that are not provable.*

## Proof Idea.

- Gödel's proof works with what is called *Gödelization*.
- Every arithmetic formula is encoded as a number (Gödel number).
- Gödelization is now used to formulate the proposition $F =$ *"I am not provable."* in the language of arithmetic.
- $F$ *is true and not provable*.
  - Assume $F$ is false. Then we can prove $F$ and therefore show that $F$ is not provable. This is a contradiction.

□

# Decidability and Incompleteness





**Note**

The deeper background of Theorem 2 is that mathematical theories (axiom systems) and, more generally, *languages become incomplete if the language becomes too powerful* (e.g., PL1).

## Example 3 ("Too powerful language")

- Set theory is so powerful that one can formulate paradoxes (= statements that contradict themselves) with it.
- For example, a paradox in set theory: "The set of all the barbers who all shave those who do not shave themselves"

## Exercise 2 ([Ertel 2025], Exercise 4.2, p. 75)

(a) Given the statement "There is a barber who shaves every person who does not shave himself." Consider whether this barber shaves himself.

(b) Let $M = \{x \mid x \notin x\}$. Describe this set and consider whether $M$ contains itself.

**Dilemma:** Languages which are powerful enough to describe mathematics and interesting applications also contain contradictions and incompletenesses

# Example: The Flying Penguin

**Goal**

We present an example to demonstrate a fundamental problem of logic and possible solution approaches.

1. Tweety is a penguin
2. Penguins are birds
3. Birds can fly

Formalized in PL1, the *knowledge base* $KB$ is:

$$\textit{penguin}(\textit{tweety})$$
$$\textit{penguin}(x) \Rightarrow \textit{bird}(x)$$
$$\textit{bird}(x) \Rightarrow \textit{fly}(x)$$

## Exercise 3

(a) Show that $KB \vdash \textit{fly}(\textit{twetty})$ (for example, with resolution)

(b) Show that if we add $\textit{penguin}(x) \Rightarrow \neg\textit{fly}(x)$ to the knowledge base $KB$, then $KB \vdash \neg\textit{fly}(\textit{twetty})$

# Example: The Flying Penguin

- From Exercise 3, both $fly(twetty)$ and $\neg fly(twetty)$ can be derived $\Rightarrow$ *The knowledge base is inconsistent*
- Although we explicitly state that penguins cannot fly, the opposite can still be derived. This is because of *the monotony of PL1*.

**Monotonic Logic**

A logic is called *monotonic* if, for an arbitrary knowledge base $KB$ and an arbitrary formula $\phi$, the set of formulas derivable from $KB$ is a subset of the formulas derivable from $KB \cup \phi$.

**Recap**

- Evidently the formalization of the flight attributes of penguins is insufficient
- To prevent the formula $fly(twetty)$ from being derived, our first attempt is to add new formulas to the $KB$. This does not work

# Example: The Flying Penguin



- We continue by replacing the obviously false statement "(all) birds can fly" in $KB$ with the more exact statement "(all) birds except penguins can fly" and obtain as $KB_2$ the following clauses:

$$\textit{penguin}(\textit{tweety})$$
$$\textit{penguin}(x) \Rightarrow \textit{bird}(x)$$
$$\textit{bird}(x) \wedge \neg\textit{penguin}(x) \Rightarrow \textit{fly}(x)$$
$$\textit{penguin}(x) \Rightarrow \neg\textit{fly}(x)$$

- **Problem solved!** We *can now derive $\neg$fly(tweety) but not fly(tweety)*, because to derive *fly*(*tweety*) we would need $\neg$*penguin*$(x)$, which is not derivable

# Example: The Flying Penguin

- A problem arises when we want to add a new bird, say the raven Abraxas (from the German book “The Little Witch”), and obtain $KB_3$

$$
\begin{aligned}
\textit{raven}(\textit{abraxas})\\
\textit{raven}(x) \Rightarrow \textit{bird}(x)\\
\textit{penguin}(\textit{tweety})\\
\textit{penguin}(x) \Rightarrow \textit{bird}(x)\\
\textit{bird}(x) \wedge \neg\textit{penguin}(x) \Rightarrow \textit{fly}(x)\\
\textit{penguin}(x) \Rightarrow \neg\textit{fly}(x)
\end{aligned}
$$

- At the moment, we cannot say anything about the flight attributes of Abraxas because we forgot to formulate that ravens are not penguins. Thus we extend $KB_3$ to $KB_4$

$$
\begin{aligned}
\textit{raven}(\textit{abraxas})\\
\textit{raven}(x) \Rightarrow \textit{bird}(x)\\
\textit{raven}(x) \Rightarrow \neg\textit{penguin}(x)\\
\textit{penguin}(\textit{tweety})\\
\textit{penguin}(x) \Rightarrow \textit{bird}(x)\\
\textit{bird}(x) \wedge \neg\textit{penguin}(x) \Rightarrow \textit{fly}(x)\\
\textit{penguin}(x) \Rightarrow \neg\textit{fly}(x)
\end{aligned}
$$

# Example: The Flying Penguin





- The fact that ravens are not penguins, which is self-evident to humans, must be explicitly added here
- For the construction of a knowledge base with all 9800 or so types of birds worldwide, it must therefore be specified for every type of bird (except for penguins) that it is not a member of penguins
- *In general, for every object in the knowledge base, in addition to its attributes, all of the attributes it does not have must be listed.*

## Exercise 4 ([Ertel 2025], Exercise 4.3, p. 75)

Use an automated theorem prover (for example E [Schulz 2002]) and apply it to all five different axiomatizations of the Tweety example mentioned above. Validate the example's statements.

# Example: The Flying Penguin





Another problem caused by the monotony is the so-called *frame problem*. This happens in complex planning problems in which the world can change

## Example 4 (An example of the frame problem)

- A blue house is painted red, then afterwards it is red
- However, with the knowledge base

$$\begin{aligned} \mathit{color}(\mathit{house}, \mathit{blue}) \\ \mathit{paint}(\mathit{house}, \mathit{red}) \\ \mathit{paint}(x, y) \Rightarrow \mathit{color}(x, y) \end{aligned}$$

  one can derive $\mathit{color}(\mathit{house}, \mathit{red})$

- Additionally, *$\mathit{color}(\mathit{house}, \mathit{blue})$ is already in the knowledge base, which leads to the conclusion that, after painting, the house is both blue and red*

# Example: The Flying Penguin





- To solve this problem, *non-monotonic logics* have been developed.
  - Knowledge (formulas) can be removed from the knowledge base.
- Despite great effort, these logics have at present, due to semantic and practical problems, not succeeded.
- Another interesting approach for modeling problems such as the Tweety example is *probability theory*.
  - The statement “all birds can fly” is false.
  - A statement something like “almost all birds can fly” is correct.
  - This statement becomes more exact if we give a probability for “birds can fly”.

# Modelling Uncertainty

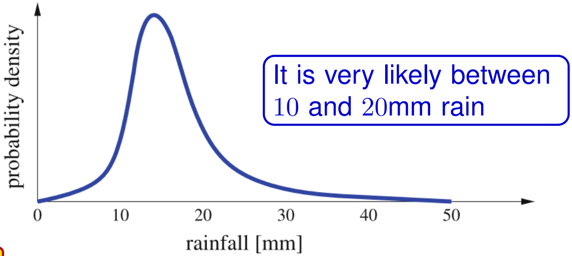
- *Two-valued logic* can and should only *model circumstances in which there is true, false, and no other truth values*.
- For many tasks in everyday reasoning, *two-valued logic is therefore not expressive enough*.
  - For example, the rule *bird*$(x) \Rightarrow$ *fly*$(x)$ is *true for almost all birds, but for some it is false*.
- As we already mentioned, to formulate uncertainty, we can use *probability theory*.
  - For example, we give a probability for *"birds can fly"*: $P(\textit{bird}(x) \Rightarrow \textit{fly}(x)) = 0.99$ *(i.e., "*$99\%$ *of all birds can fly")*
  - Later, we will see that here it is better to work with *conditional probabilities* such as $P(\textit{fly}|\textit{bird}) = 0.99$. With the help of *Bayesian networks*, complex applications with many variables can also be modelled.
  - *Fuzzy logic* is required for *"The weather is nice"*. Here it makes no sense to speak in terms of true and false.
  - The variable *weather_is_nice* is *continuous with values in* $[0, 1]$. *weather_is_nice* $= 0.7$ then means *"The weather is fairly nice"*.

# Modelling Uncertainty

- Probability theory also offers the possibility of *making statements about the probability of continuous variables*.

"There is a high probability that there will be some rain"

$$P(\textit{rainfall} = X) = Y$$




**Note**

This very *general and even visualizable representation* of both types of uncertainty we have discussed, together with *inductive statistics* and the theory of *Bayesian networks*, makes it possible, in principle, *to answer arbitrary probabilistic queries*.

# Modelling Uncertainty

Comparison of different formalisms for the modelling of uncertain knowledge

| **Formalism** | **Number of truth values** | **Probabilities expressible** |
|---|---|---|
| Propositional logic | $2$ | — |
| Fuzzy logic | $\infty$ | — |
| Discrete probabilistic logic | $n$ | yes |
| *Continuous probabilistic logic* | $\infty$ | *yes* |

# References


Ertel, Wolfgang (2025). *Introduction to Artificial Intelligence*. 3rd. Springer. DOI: 10.1007/978-3-658-43102-0.

Schulz, Stephan (2002). "E–A Brainiac Theorem Prover." In: *AI Communications* 15.2–3, pp. 111–126. URL: https://wwwlehre.dhbw-stuttgart.de/~sschulz/E/E.html.