Artificial Intelligence 1
Winter Semester 2024/25

– Lecture Notes –
Part III: Knowledge and Inference

Prof. Dr. Michael Kohlhase

Professur für Wissensrepräsentation und -verarbeitung
Informatik, FAU Erlangen-Nürnberg
Michael.Kohlhase@FAU.de

2025-02-06

This document contains Part III of the course notes for the course "Artificial Intelligence 1" held at FAU Erlangen-Nürnberg in the Winter Semesters 2016/17 ff. **A Video Nugget** covering this document can be found at `https://fau.tv/clip/id/22466`.

This part of the course introduces representation languages and inference methods for structured state representations for agents: In contrast to the atomic and factored state representations from **??**, we look at state representations where the relations between objects are not determined by the problem statement, but can be determined by inference-based methods, where the knowledge about the environment is represented in a formal langauge and new knowledge is derived by transforming expressions of this language.

We look at propositional logic – a rather weak representation langauge – and first-order logic – a much stronger one – and study the respective inference procedures. In the end we show that computation in Prolog is just an inference process as well. Other parts of the lecture notes can be found at `http://kwarc.info/teaching/AI/notes-*.pdf`.

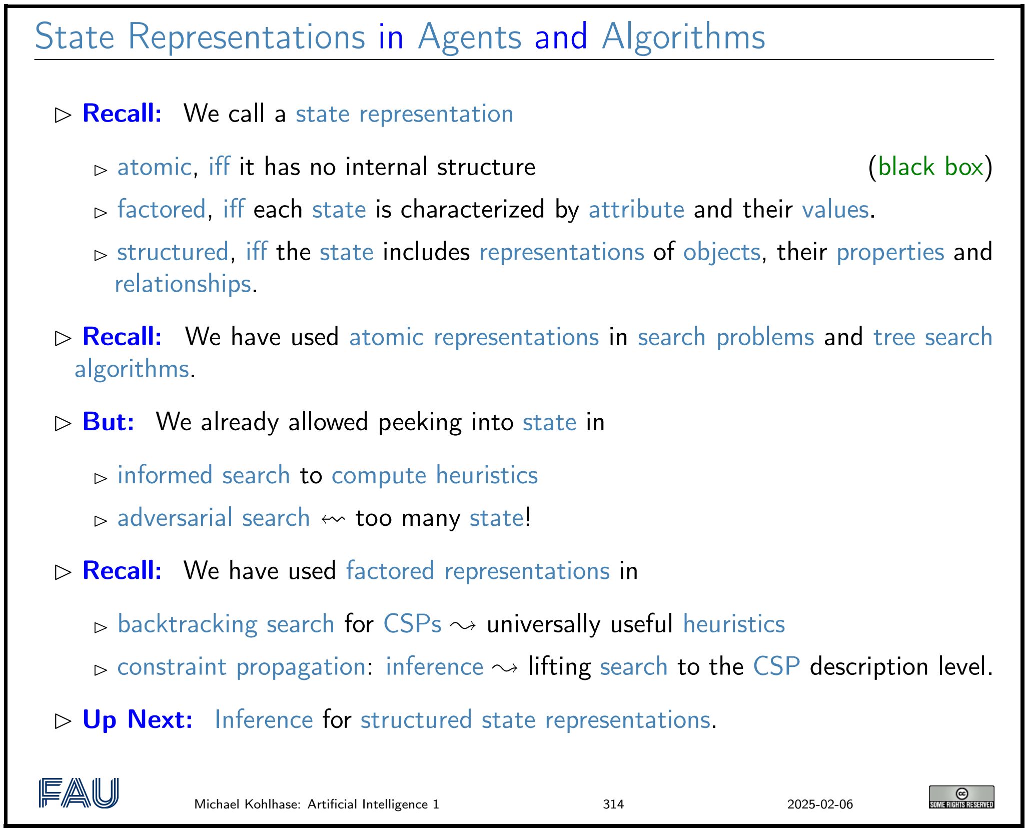
Contents

Chapter 10

Propositional Logic & Reasoning, Part I: Principles

10.1 Introduction: Inference with Structured State Representations

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/22455`.



State Representations in Agents and Algorithms

- **Recall:** We call a state representation
  - atomic, iff it has no internal structure (black box)
  - factored, iff each state is characterized by attribute and their values.
  - structured, iff the state includes representations of objects, their properties and relationships.
- **Recall:** We have used atomic representations in search problems and tree search algorithms.
- **But:** We already allowed peeking into state in
  - informed search to compute heuristics
  - adversarial search ↭ too many state!
- **Recall:** We have used factored representations in
  - backtracking search for CSPs ⇝ universally useful heuristics
  - constraint propagation: inference ⇝ lifting search to the CSP description level.
- **Up Next:** Inference for structured state representations.

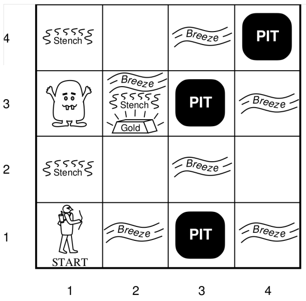
FAU Michael Kohlhase: Artificial Intelligence 1 314 2025-02-06

10.1.1 A Running Example: The Wumpus World

To clarify the concepts and methods for inference with structured state representations, we now introduce an extended example (the Wumpus world) and the agent model (logic-based agents) that use them. We will refer back to both from time to time below.

The Wumpus world is a very simple game modeled after the early text adventure games of the 1960 and 70ies, where the player entered a world and was provided with textual information about percepts and could explore the world via actions. The main difference is that we use it as an agent environment in this course.

The Wumpus World



Definition 10.1.1. The Wumpus world is a simple game where an agent explores a cave with 16 cells that can contain pits, gold, and the Wumpus with the goal of getting back out alive with the gold.
The agent cannot observe the locations of pits, gold, and the Wumpus, but some of their effects in the cell it currently visits.

▷ **Definition 10.1.2 (Actions).** The agent can perform the following actions: goForward, turnRight (by 90°), turnLeft (by 90°), shoot arrow in direction you're facing (you got exactly one arrow), grab an object in current cell, leave cave if you're in cell $[1, 1]$.

▷ **Definition 10.1.3 (Initial and Terminal States).** Initially, the agent is in cell $[1, 1]$ facing east. If the agent falls down a pit or meets live Wumpus it dies.

▷ **Definition 10.1.4 (Percepts).** The agent can experience the following percepts: stench, breeze, glitter, bump, scream, none.

- ▹ Cell adjacent (i.e. north, south, west, east) to Wumpus: stench (else: none).
- ▹ Cell adjacent to pit: breeze (else: none).
- ▹ Cell that contains gold: glitter (else: none).
- ▹ You walk into a wall: bump (else: none).
- ▹ Wumpus shot by arrow: scream (else: none).

The game is complex enough to warrant structured state representations and can easily be extended to include uncertainty and non-determinism later.
As our focus is on inference processes here, let us see how a human player would reason when entering the Wumpus world. This can serve as a model for designing our artificial agents.

Reasoning in the Wumpus World

▷ **Example 10.1.5 (Reasoning in the Wumpus World).**
As humans we mark cells with the knowledge inferred so far: **A**: agent, **V**: visited, **OK**: safe, **P**: pit, **W**: Wumpus, **B**: breeze, **S**: stench, **G**: gold.

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

(1) Initial state

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

(2) One step to right

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

(3) Back, and up to [1,2]

- ▷ *The Wumpus is in [1,3]*! How do we know?
- ▷ No stench in [2,1], so the stench in [1,2] can only come from [1,3].
- ▷ *There's a pit in [3,1]*! How do we know?
- ▷ No breeze in [1,2], so the breeze in [2,1] can only come from [3,1].

▷ **Note:** The agent has more knowledge than just the percepts ⇝ inference!

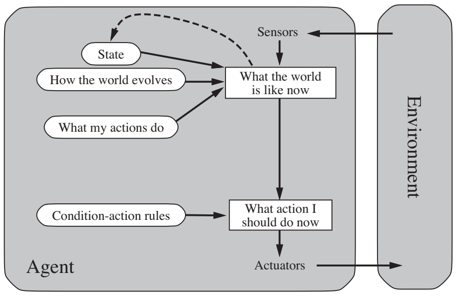
Let us now look into what kind of agent we would need to be successful in the Wumpus world: it seems reasonable that we should build on a model-based agent and specialize it to structured state representations and inference.

Agents that Think Rationally

▷ **Problem:** But how can we build an agent that can do the necessary inferences?

▷ **Idea:** Think Before You Act!
"Thinking" = Inference about knowledge represented using logic.

▷ **Definition 10.1.6.** A logic-based agent is a model-based agent that represents the world state as a logical formula and uses inference to think about the state of the environment and its own actions. Agent schema:



The formal language of the logical system acts as a world description language.
Agent function:

```
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($action$,$t$))
 $t$ := $t$+1
return $action$
```

Its agent function maintains a knowledge base about the environment, which is updated with percept descriptions (formalizations of the percepts) and action descriptions. The next action is the result of a suitable inference-based query to the knowledge base.

10.1.2 Propositional Logic: Preview

We will now give a preview of the concepts and methods in propositional logic based on the Wumpus world before we formally define them below. The focus here is on the use of PL^0 as a world description language and understanding how inference might work.
We will start off with our preview by looking into the use of PL^0 as a world description language for the Wumpus world. For that we need to fix the language itself (its syntax) and the meaning of expressions in PL^0 (its semantics).

Logic: Basic Concepts (Representing Knowledge)

▷ We now preview some of the concepts involved in logic so that you have an intuition for the formal definitions below.

▷ **Definition 10.1.7.** Syntax: What are legal formulae $\mathbf{A}$ in the logic?

▷ **Example 10.1.8.** "W" and "$W \Rightarrow S$".
($W \mathrel{\widehat{=}}$ *Wumpus is here*, $S \mathrel{\widehat{=}}$ *it stinks*, $W \Rightarrow S \mathrel{\widehat{=}}$ If W, then S)

▷ **Definition 10.1.9.** Semantics: Which formulae $\mathbf{A}$ are true?

▷ **Observation:** Whether $W \Rightarrow S$ is true depends on whether W and S are!

▷ **Idea:** Capture the state of W and S... in a variable assignment.

▷ **Definition 10.1.10.** For a variable assignment φ, write $\varphi \models \mathbf{A}$ if φ is true in the Wumpus world described by φ.

▷ **Example 10.1.11.** If $\varphi := \{W \mapsto \mathsf{T}, S \mapsto \mathsf{F}\}$, then $\varphi \models W$ but $\varphi \not\models (W \Rightarrow S)$.

▷ **Intuition:** Knowledge about the state of the world is described by formulae, interpretations evaluate them in the current world (they should turn out true!)

▷ **Definition 10.1.12.** The process of representing a natural language text in the formal language of a logical system is called formalization.

▷ **Observation:** Formalizing a NL text or utterance makes it machine-actionable. (the ultimate purpose of AI)

▷ **Observation:** Formalization is an art/skill, not a science!

It is critical to understand that while PL^0 as a logical system is given once and for all, the agent designer still has to formalize the situation (here the Wumpus world) in the world description language (here PL^0; but we will look at more expressive logical systems below). This formalization is the seed of the knowledge base, the logic-based agent can then add to via its percepts and action descriptions, and that also forms the basis of its inferences. We will look at this aspect now.

Logic: Basic Concepts (Reasoning about Knowledge)

- ▷ **Definition 10.1.13.** Entailment: Which $\mathbf{B}$ follow from $\mathbf{A}$, written $\mathbf{A} \models \mathbf{B}$, meaning that, for all φ with $\varphi \models \mathbf{A}$, we have $\varphi \models \mathbf{B}$? E.g., $P \wedge (P \Rightarrow Q) \models Q$.
- ▷ **Intuition:** Entailment $\widehat{=}$ ideal outcome of reasoning, everything that we can possibly conclude. e.g. determine Wumpus position as soon as we have enough information
- ▷ **Definition 10.1.14.** Deduction: Which formulas $\mathbf{B}$ can be derived from $\mathbf{A}$ using a set $\mathcal{C}$ of inference rules (a calculus), written $\mathbf{A} \vdash_{\mathcal{C}} \mathbf{B}$?
- ▷ **Example 10.1.15.** If $\mathcal{C}$ contains $\dfrac{\mathbf{A} \quad \mathbf{A} \Rightarrow \mathbf{B}}{\mathbf{B}}$ then $P, P \Rightarrow Q \vdash_{\mathcal{C}} Q$
- ▷ **Intuition:** Deduction $\widehat{=}$ process in an actual computer trying to reason about entailment. E.g. a mechanical process attempting to determine Wumpus position.
- ▷ **Critical Insight:** Entailment is purely semantical and gives a mathematical foundation of reasoning in PL^0, while Deduction is purely syntactic and can be implemented well. (but this only helps if they are related)
- ▷ **Definition 10.1.16.** Soundness: whenever $\mathbf{A} \vdash_{\mathcal{C}} \mathbf{B}$, we also have $\mathbf{A} \models \mathbf{B}$.
- ▷ **Definition 10.1.17.** Completeness: whenever $\mathbf{A} \models \mathbf{B}$, we also have $\mathbf{A} \vdash_{\mathcal{C}} \mathbf{B}$.

General Problem Solving using Logic

- ▷ **Idea:** Any problem that can be formulated as reasoning about logic. ⇝ use off-the-shelf reasoning tool.
- ▷ *Very* successful using propositional logic and modern SAT solvers! (Propositional satisfiability testing; ??)

Propositional Logic and Its Applications

- ▷ Propositional logic = canonical form of knowledge + reasoning.
  - ▹ Syntax: Atomic propositions that can be either true or false, connected by “and, or, and not”.
  - ▹ Semantics: Assign value to every proposition, evaluate connectives.
- ▷ **Applications:** Despite its simplicity, widely applied!

- ▹ **Product configuration** (e.g., Mercedes). Check consistency of customized combinations of components.
- ▹ **Hardware verification** (e.g., Intel, AMD, IBM, Infineon). Check whether a circuit has a desired property p.
- ▹ **Software verification**: Similar.
- ▹ **CSP applications**: Propositional logic can be (successfully!) used to formulate and solve constraint satisfaction problems. (see ??)

▷ ?? gives an example for verification.

Michael Kohlhase: Artificial Intelligence 1 321 2025-02-06

10.1.3 Propositional Logic: Agenda

Our Agenda for This Topic

▷ **This subsection**: Basic definitions and concepts; tableaux, resolution.

- ▹ Sets up the framework. Resolution is the quintessential reasoning procedure underlying most successful SAT solvers.

▷ **Next Section (??)**: The Davis Putnam procedure and clause learning; practical problem structure.

- ▹ State-of-the-art algorithms for reasoning about propositional logic, and an important observation about how they behave.

Michael Kohlhase: Artificial Intelligence 1 322 2025-02-06

Our Agenda for This Chapter

▷ Propositional logic: What's the syntax and semantics? How can we capture deduction?

- ▹ We study this logic formally.

▷ **Tableaux, Resolution**: How can we make deduction mechanizable? What are its properties?

- ▹ Formally introduces the most basic machine-oriented reasoning algorithm.

▷ **Killing a Wumpus**: How can we use all this to figure out where the Wumpus is?

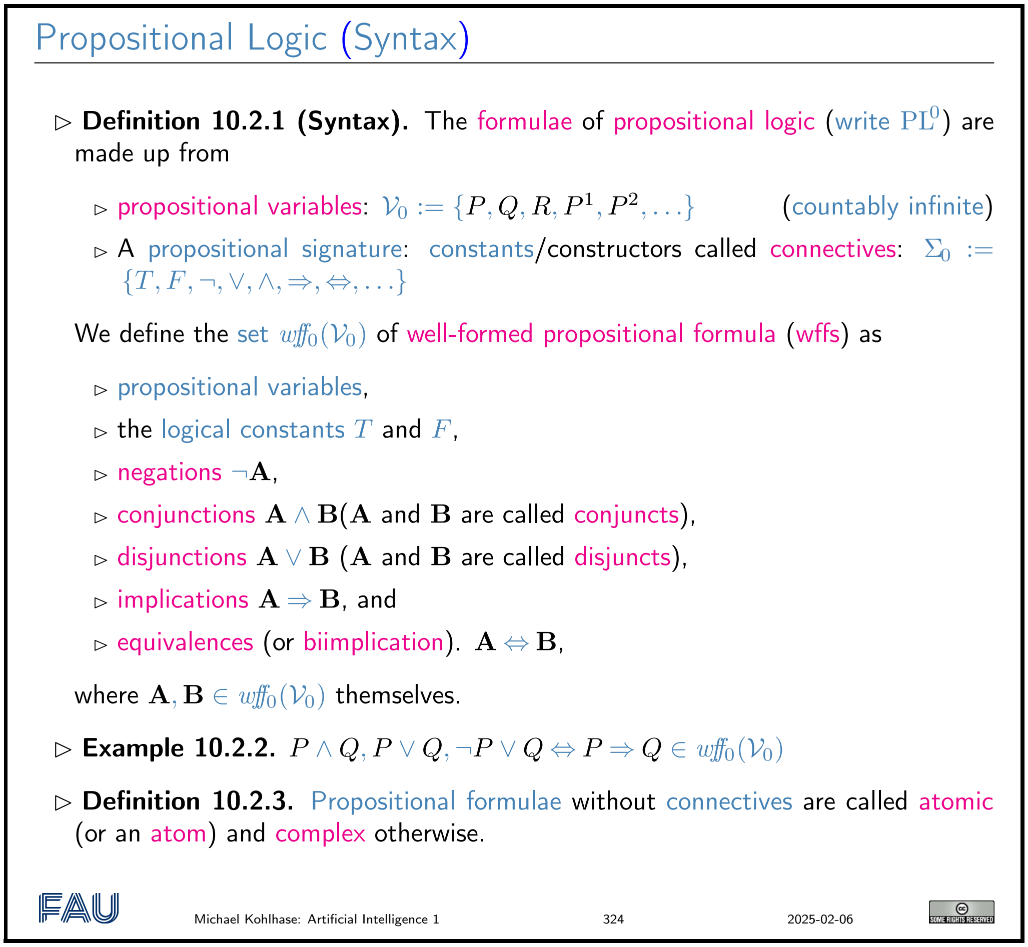
- ▹ Coming back to our introductory example.

Michael Kohlhase: Artificial Intelligence 1 323 2025-02-06

10.2 Propositional Logic (Syntax/Semantics)

Video Nuggets covering this section can be found at `https://fau.tv/clip/id/22457` and `https://fau.tv/clip/id/22458`.

We will now develop the formal theory behind the ideas previewed in the last section and use that as a prototype for the theory of the more expressive logical systems still to come in AI-1. As PL^0 is a very simple logical system, we could cut some corners in the exposition but we will stick closer to a generalizable theory.



Propositional Logic (Syntax)

▷ **Definition 10.2.1 (Syntax).** The formulae of propositional logic (write PL^0) are made up from

- ▷ propositional variables: $\mathcal{V}_0 := \{P, Q, R, P^1, P^2, \ldots\}$ (countably infinite)
- ▷ A propositional signature: constants/constructors called connectives: $\Sigma_0 := \{T, F, \neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, \ldots\}$

We define the set $\mathit{wff}_0(\mathcal{V}_0)$ of well-formed propositional formula (wffs) as

- ▷ propositional variables,
- ▷ the logical constants T and F,
- ▷ negations $\neg\mathbf{A}$,
- ▷ conjunctions $\mathbf{A} \wedge \mathbf{B}$($\mathbf{A}$ and $\mathbf{B}$ are called conjuncts),
- ▷ disjunctions $\mathbf{A} \vee \mathbf{B}$ ($\mathbf{A}$ and $\mathbf{B}$ are called disjuncts),
- ▷ implications $\mathbf{A} \Rightarrow \mathbf{B}$, and
- ▷ equivalences (or biimplication). $\mathbf{A} \Leftrightarrow \mathbf{B}$,

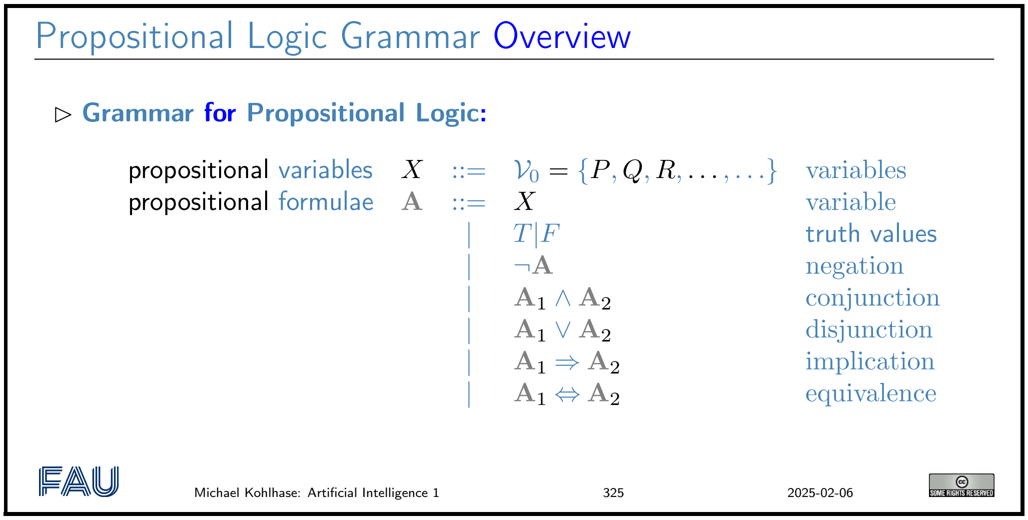
where $\mathbf{A}, \mathbf{B} \in \mathit{wff}_0(\mathcal{V}_0)$ themselves.

▷ **Example 10.2.2.** $P \wedge Q, P \vee Q, \neg P \vee Q \Leftrightarrow P \Rightarrow Q \in \mathit{wff}_0(\mathcal{V}_0)$

▷ **Definition 10.2.3.** Propositional formulae without connectives are called atomic (or an atom) and complex otherwise.

Michael Kohlhase: Artificial Intelligence 1 324 2025-02-06

We can also express the formal language introduced by ?? as a context-free grammar.



Propositional Logic Grammar Overview

▷ **Grammar for Propositional Logic:**

$$\begin{array}{llcll}
\text{propositional variables} & X & ::= & \mathcal{V}_0 = \{P, Q, R, \ldots, \ldots\} & \text{variables} \\
\text{propositional formulae} & \mathbf{A} & ::= & X & \text{variable} \\
 & & | & T|F & \text{truth values} \\
 & & | & \neg\mathbf{A} & \text{negation} \\
 & & | & \mathbf{A}_1 \wedge \mathbf{A}_2 & \text{conjunction} \\
 & & | & \mathbf{A}_1 \vee \mathbf{A}_2 & \text{disjunction} \\
 & & | & \mathbf{A}_1 \Rightarrow \mathbf{A}_2 & \text{implication} \\
 & & | & \mathbf{A}_1 \Leftrightarrow \mathbf{A}_2 & \text{equivalence}
\end{array}$$

Michael Kohlhase: Artificial Intelligence 1 325 2025-02-06

Propositional logic is a very old and widely used logical system. So it should not be surprising that there are other notations for the connectives than the ones we are using in AI-1. We list the most important ones here for completeness.

Alternative Notations for Connectives

Here	Elsewhere		
$\neg\mathbf{A}$	$\sim\mathbf{A}$	$\overline{\mathbf{A}}$	
$\mathbf{A}\wedge\mathbf{B}$	$\mathbf{A}\,\&\,\mathbf{B}$	$\mathbf{A}\bullet\mathbf{B}$	$\mathbf{A},\mathbf{B}$
$\mathbf{A}\vee\mathbf{B}$	$\mathbf{A}+\mathbf{B}$	$\mathbf{A}\mid\mathbf{B}$	$\mathbf{A};\mathbf{B}$
$\mathbf{A}\Rightarrow\mathbf{B}$	$\mathbf{A}\rightarrow\mathbf{B}$	$\mathbf{A}\supset\mathbf{B}$	
$\mathbf{A}\Leftrightarrow\mathbf{B}$	$\mathbf{A}\leftrightarrow\mathbf{B}$	$\mathbf{A}\equiv\mathbf{B}$	
F	$\bot$	0	
T	$\top$	1	

These notations will not be used in AI-1, but sometimes appear in the literature.
The semantics of PL^0 is defined relative to a model, which consists of a universe of discourse and an interpretation function that we specify now.

Semantics of PL^0 (Models)

- **Warning:** For the official semantics of PL^0 we will separate the tasks of giving meaning to connectives and propositional variables to different mappings.
- This will generalize better to other logical systems. (and thus applications)
- **Definition 10.2.4.** A model $\mathcal{M}:=\langle\mathcal{D}_o,\mathcal{I}\rangle$ for propositional logic consists of
  - the universe $\mathcal{D}_o=\{\mathsf{T},\mathsf{F}\}$
  - the interpretation $\mathcal{I}$ that assigns values to essential connectives.
  - $\mathcal{I}(\neg)\colon\mathcal{D}_o\rightarrow\mathcal{D}_o;\mathsf{T}\mapsto\mathsf{F},\mathsf{F}\mapsto\mathsf{T}$
  - $\mathcal{I}(\wedge)\colon\mathcal{D}_o\times\mathcal{D}_o\rightarrow\mathcal{D}_o;\langle\alpha,\beta\rangle\mapsto\mathsf{T}$, iff $\alpha=\beta=\mathsf{T}$

  We call a constant a logical constant, iff its value is fixed by the interpretation.
- Treat the other connectives as abbreviations, e.g. $\mathbf{A}\vee\mathbf{B}\widehat{=}\neg(\neg\mathbf{A}\wedge\neg\mathbf{B})$ and $\mathbf{A}\Rightarrow\mathbf{B}\widehat{=}\neg\mathbf{A}\vee\mathbf{B}$, and $T\widehat{=}P\vee\neg P$ (only need to treat $\neg,\wedge$ directly)
- **Note:** PL^0 is a single-model logical system with canonical model $\langle\mathcal{D}_o,\mathcal{I}\rangle$.

We have a problem in the exposition of the theory here: As PL^0 semantics only has a single, canonical model, we could simplify the exposition by just not mentioning the universe and interpretation function. But we choose to expose both of them in the construction, since other versions of propositional logic – in particular the system $\mathrm{PL}^{\mathrm{nq}}$ below – that have a choice of models as they use a different distribution of the representation among constants and variables.

Semantics of PL^0 (Evaluation)

- **Problem:** The interpretation function $\mathcal{I}$ only assigns meaning to connectives.
- **Definition 10.2.5.** A variable assignment $\varphi\colon \mathcal{V}_0 \to \mathcal{D}_o$ assigns values to propositional variables.
- **Definition 10.2.6.** The value function $\mathcal{I}_\varphi\colon \mathit{wff}_0(\mathcal{V}_0) \to \mathcal{D}_o$ assigns values to PL^0 formulae. It is recursively defined,
  - $\mathcal{I}_\varphi(P) = \varphi(P)$ (base case)
  - $\mathcal{I}_\varphi(\neg\mathbf{A}) = \mathcal{I}(\neg)(\mathcal{I}_\varphi(\mathbf{A}))$.
  - $\mathcal{I}_\varphi(\mathbf{A} \wedge \mathbf{B}) = \mathcal{I}(\wedge)(\mathcal{I}_\varphi(\mathbf{A}), \mathcal{I}_\varphi(\mathbf{B}))$.
- **Note:** $\mathcal{I}_\varphi(\mathbf{A} \vee \mathbf{B}) = \mathcal{I}_\varphi(\neg(\neg\mathbf{A} \wedge \neg\mathbf{B}))$ is only determined by $\mathcal{I}_\varphi(\mathbf{A})$ and $\mathcal{I}_\varphi(\mathbf{B})$, so we think of the defined connectives as logical constants as well.
- **Alternative Notation:** Write $[\![\mathbf{A}]\!]_\varphi$ for $\mathcal{I}_\varphi(\mathbf{A})$. (and $[\![\mathbf{A}]\!]$, if $\mathbf{A}$ is ground)
- **Definition 10.2.7.** Two formulae $\mathbf{A}$ and $\mathbf{B}$ are called equivalent, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathcal{I}_\varphi(\mathbf{B})$ for all variable assignments φ.

In particular in a interpretation-less exposition of propositional logic would have elided the homomorphic construction of the value function and could have simplified the recursive cases in ?? to $\mathcal{I}_\varphi(\mathbf{A} \wedge \mathbf{B}) = \mathsf{T}$, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T} = \mathcal{I}_\varphi(\mathbf{B})$.

But the homomorphic construction via $\mathcal{I}(\wedge)$ is standard to definitions in other logical systems and thus generalizes better.

Computing Semantics

- **Example 10.2.8.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

$$\begin{aligned}
& \mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1 \vee P_2), \mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathcal{I}_\varphi(P_1), \mathcal{I}_\varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}_\varphi(\neg(\neg P_1 \wedge P_2)), \mathcal{I}_\varphi(P_3 \wedge P_4))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\varphi(P_1), \varphi(P_2)), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(\neg P_1 \wedge P_2)), \mathcal{I}(\wedge)(\mathcal{I}_\varphi(P_3), \mathcal{I}_\varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathcal{I}(\vee)(\mathsf{T}, \mathsf{F}), \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}_\varphi(\neg P_1), \mathcal{I}_\varphi(P_2))), \mathcal{I}(\wedge)(\varphi(P_3), \varphi(P_4)))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathcal{I}_\varphi(P_1)), \varphi(P_2))), \mathcal{I}(\wedge)(\mathsf{T}, \mathsf{F}))) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\varphi(P_1)), \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathcal{I}(\neg)(\mathsf{T}), \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathcal{I}(\wedge)(\mathsf{F}, \mathsf{F})), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathcal{I}(\neg)(\mathsf{F}), \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathcal{I}(\vee)(\mathsf{T}, \mathsf{F})) \\
= \; & \mathcal{I}(\vee)(\mathsf{T}, \mathsf{T}) \\
= \; & \mathsf{T}
\end{aligned}$$

- What a mess!

Now we will also review some propositional identities that will be useful later on. Some of them we have already seen, and some are new. All of them can be proven by simple truth table arguments.

Propositional Identities

▷ **Definition 10.2.9.** We have the following identities in propositional logic:

Name	for $\wedge$	for $\vee$
Idempotence	$\varphi \wedge \varphi = \varphi$	$\varphi \vee \varphi = \varphi$
Identity	$\varphi \wedge T = \varphi$	$\varphi \vee F = \varphi$
Absorption 1	$\varphi \wedge F = F$	$\varphi \vee T = T$
Commutativity	$\varphi \wedge \psi = \psi \wedge \varphi$	$\varphi \vee \psi = \psi \vee \varphi$
Associativity	$\varphi \wedge (\psi \wedge \theta) = (\varphi \wedge \psi) \wedge \theta$	$\varphi \vee (\psi \vee \theta) = (\varphi \vee \psi) \vee \theta$
Distributivity	$\varphi \wedge (\psi \vee \theta) = \varphi \wedge \psi \vee \varphi \wedge \theta$	$\varphi \vee \psi \wedge \theta = (\varphi \vee \psi) \wedge (\varphi \vee \theta)$
Absorption 2	$\varphi \wedge (\varphi \vee \theta) = \varphi$	$\varphi \vee \varphi \wedge \theta = \varphi$
De Morgan rule	$\neg(\varphi \wedge \psi) = \neg\varphi \vee \neg\psi$	$\neg(\varphi \vee \psi) = \neg\varphi \wedge \neg\psi$
double negation	$\neg\neg\varphi = \varphi$	
Definitions	$\varphi \Rightarrow \psi = \neg\varphi \vee \psi$	$\varphi \Leftrightarrow \psi = (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$

▷ **Idea:** How about using these as inference component (simplification) to simplify calculations like the one in ??. (see below)

We will now use the distribution of values of a propositional formula under all variable assignments to characterize them semantically. The intuition here is that we want to understand theorems, examples, counterexamples, and inconsistencies in mathematics and everyday reasoning[1].

The idea is to use the formal language of propositional formulae as a model for mathematical language. Of course, we cannot express all of mathematics as propositional formulae, but we can at least study the interplay of mathematical statements (which can be true or false) with the copula "and", "or" and "not".

Semantic Properties of Propositional Formulae

▷ **Definition 10.2.10.** Let $\mathcal{M} := \langle \mathcal{U}, \mathcal{I} \rangle$ be our model, then we call $\mathbf{A}$

- ▷ true under φ (φ satisfies $\mathbf{A}$) in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$, (write $\mathcal{M} \models^\varphi \mathbf{A}$)
- ▷ false under φ (φ falsifies $\mathbf{A}$) in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{F}$, (write $\mathcal{M} \not\models^\varphi \mathbf{A}$)
- ▷ satisfiable in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$ for some assignment φ,
- ▷ valid in $\mathcal{M}$, iff $\mathcal{M} \models^\varphi \mathbf{A}$ for all variable assignments φ,
- ▷ falsifiable in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{F}$ for some assignments φ, and
- ▷ unsatisfiable in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{F}$ for all assignments φ.

▷ **Example 10.2.11.** $x \vee x$ is satisfiable and falsifiable.

▷ **Example 10.2.12.** $x \vee \neg x$ is valid and $x \wedge \neg x$ is unsatisfiable.

▷ **Note:** As PL^0 is a single-model logical system, we can elide the reference to the model and regain the notation $\varphi \models \mathbf{A}$ from the preview for $\mathcal{M} \models^\varphi \mathbf{A}$.

▷ **Definition 10.2.13 (Entailment).** (aka. logical consequence)
We say that $\mathbf{A}$ entails $\mathbf{B}$ (write $\mathbf{A} \vDash \mathbf{B}$), iff $\mathcal{I}_\varphi(\mathbf{B}) = \mathsf{T}$ for all φ with $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$ (i.e. all assignments that make $\mathbf{A}$ true also make $\mathbf{B}$ true)

[1] Here (and elsewhere) we will use mathematics (and the language of mathematics) as a test tube for understanding reasoning, since mathematics has a long history of studying its own reasoning processes and assumptions.

Let us now see how these semantic properties model mathematical practice.

In mathematics we are interested in assertions that are true in all circumstances. In our model of mathematics, we use variable assignments to stand for "circumstances". So we are interested in propositional formulae which are true under all variable assignments; we call them valid. We often give examples (or show situations) which make a conjectured formula false; we call such examples counterexamples, and such assertions falsifiable. We also often give examples for certain formulae to show that they can indeed be made true (which is not the same as being valid yet); such assertions we call satisfiable. Finally, if a formula cannot be made true in any circumstances we call it unsatisfiable; such assertions naturally arise in mathematical practice in the form of refutation proofs, where we show that an assertion (usually the negation of the theorem we want to prove) leads to an obviously unsatisfiable conclusion, showing that the negation of the theorem is unsatisfiable, and thus the theorem valid.

A better mouse-trap: Truth Tables

▷ Truth tables visualize truth functions:

$\neg$	
$\top$	F
$\bot$	T

$\wedge$	$\top$	$\bot$
$\top$	T	F
$\bot$	F	F

$\vee$	$\top$	$\bot$
$\top$	T	T
$\bot$	T	F

▷ If we are interested in values for all assignments (e.g $z \wedge x \vee \neg(z \wedge y)$)

assignments			intermediate results			full
x	y	z	$e_1 := z \wedge y$	$e_2 := \neg e_1$	$e_3 := z \wedge x$	$e_3 \vee e_2$
F	F	F	F	T	F	T
F	F	T	F	T	F	T
F	T	F	F	T	F	T
F	T	T	T	F	F	F
T	F	F	F	T	F	T
T	F	T	F	T	T	T
T	T	F	F	T	F	T
T	T	T	T	F	T	T

Let us finally test our intuitions about propositional logic with a "real-world example": a logic puzzle, as you could find it in a Sunday edition of the local newspaper.

Hair Color in Propositional Logic

▷ There are three persons, Stefan, Nicole, and Jochen.

1. Their hair colors are black, red, or green.
2. Their study subjects are AI, Physics, or Chinese at least one studies AI.
   (a) Persons with red or green hair do not study AI.
   (b) Neither the Physics nor the Chinese students have black hair.
   (c) Of the two male persons, one studies Physics, and the other studies Chinese.

▷ **Question:** Who studies AI?
(A) Stefan (B) Nicole (C) Jochen (D) Nobody

▷ **Answer:** You can solve this using PL^0, if we accept $bla(S)$, etc. as propositional variables.
We first express what we know: For every $x \in \{S, N, J\}$ (Stefan, Nicole, Jochen) we have

1. $bla(x) \vee red(x) \vee gre(x)$; (note: three formulae)
2. $ai(x) \vee phy(x) \vee chi(x)$ and $ai(S) \vee ai(N) \vee ai(J)$
   (a) $ai(x) \Rightarrow \neg red(x) \wedge \neg gre(x)$.
   (b) $phy(x) \Rightarrow \neg bla(x)$ and $chi(x) \Rightarrow \neg bla(x)$.
   (c) $phy(S) \wedge chi(J) \vee phy(J) \wedge chi(S)$.

Now, we obtain new knowledge via entailment steps:

3. 1. together with 2.2a entails that $ai(x) \Rightarrow bla(x)$ for every $x \in \{S, N, J\}$,
4. thus $\neg bla(S) \wedge \neg bla(J)$ by 2.2c and 2.2b and
5. so $\neg ai(S) \wedge \neg ai(J)$ by 3. and 4.
6. With 2. the latter entails $ai(N)$.

The example shows that puzzles like that are a bit difficult to solve without writing things down. But if we formalize the situation in PL^0, then we can solve the puzzle quite handily with inference. Note that we have been a bit generous with the names of propositional variables; e.g. $bla(x)$, where $x \in \{S, N, J\}$, to keep the representation small enough to fit on the slide. This does not hinder the method in any way.

10.3 Inference in Propositional Logics

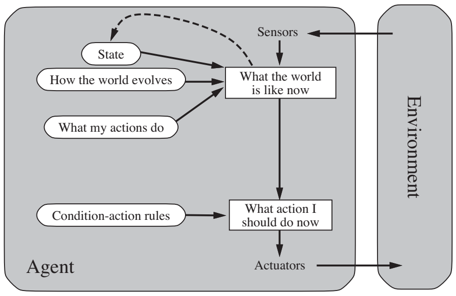
We have now defined syntax (the language agents can use to represent knowledge) and its semantics (how expressions of this language relate to agent's environment). Theoretically, an agent could use the entailment relation to derive new knowledge from percepts and the existing state representation – in the MAKE−PERCEPT−SENTENCE and MAKE−ACTION−SENTENCE subroutines below. But as we have seen in above, this is very tedious. A much better way would be to have a set of rules that directly act on the state representations.
Let us now look into what kind of agent we would need to be successful in the Wumpus world: it seems reasonable that we should build on a model-based agent and specialize it to structured state representations and inference.

Agents that Think Rationally

▷ **Problem:** But how can we build an agent that can do the necessary inferences?

▷ **Idea:** Think Before You Act!
"Thinking" = Inference about knowledge represented using logic.

▷ **Definition 10.3.1.** A logic-based agent is a model-based agent that represents the world state as a logical formula and uses inference to think about the state of the environment and its own actions. Agent schema:



The formal language of the logical system acts as a world description language. Agent function:

```
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($action$,$t$))
   $t$ := $t$+1
  return $action$
```

Its agent function maintains a knowledge base about the environment, which is updated with percept descriptions (formalizations of the percepts) and action descriptions. The next action is the result of a suitable inference-based query to the knowledge base.

A Simple Formal System: Prop. Logic with Hilbert-Calculus

▷ **Formulae:** Built from propositional variables: $P, Q, R...$ and implication: $\Rightarrow$

▷ **Semantics:** $\mathcal{I}_\varphi(P) = \varphi(P)$ and $\mathcal{I}_\varphi(\mathbf{A} \Rightarrow \mathbf{B}) = \mathsf{T}$, iff $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{F}$ or $\mathcal{I}_\varphi(\mathbf{B}) = \mathsf{T}$.

▷ **Definition 10.3.2.** The Hilbert calculus $\mathcal{H}^0$ consists of the inference rules:

$$\frac{}{P \Rightarrow Q \Rightarrow P}\,\mathrm{K} \qquad \frac{}{(P \Rightarrow Q \Rightarrow R) \Rightarrow (P \Rightarrow Q) \Rightarrow P \Rightarrow R}\,\mathrm{S}$$

$$\frac{\mathbf{A} \Rightarrow \mathbf{B} \quad \mathbf{A}}{\mathbf{B}}\,\mathrm{MP} \qquad \frac{\mathbf{A}}{[\mathbf{B}/X](\mathbf{A})}\,\mathrm{Subst}$$

▷ **Example 10.3.3.** A $\mathcal{H}^0$ theorem $\mathbf{C} \Rightarrow \mathbf{C}$ and its proof

Proof: We show that $\emptyset \vdash_{\mathcal{H}^0} \mathbf{C} \Rightarrow \mathbf{C}$

1. $(\mathbf{C} \Rightarrow (\mathbf{C} \Rightarrow \mathbf{C}) \Rightarrow \mathbf{C}) \Rightarrow (\mathbf{C} \Rightarrow \mathbf{C} \Rightarrow \mathbf{C}) \Rightarrow \mathbf{C} \Rightarrow \mathbf{C}$ (S with $[\mathbf{C}/P], [\mathbf{C} \Rightarrow \mathbf{C}/Q], [\mathbf{C}/R]$)
2. $\mathbf{C} \Rightarrow (\mathbf{C} \Rightarrow \mathbf{C}) \Rightarrow \mathbf{C}$ (K with $[\mathbf{C}/P], [\mathbf{C} \Rightarrow \mathbf{C}/Q]$)
3. $(\mathbf{C} \Rightarrow \mathbf{C} \Rightarrow \mathbf{C}) \Rightarrow \mathbf{C} \Rightarrow \mathbf{C}$ (MP on P.1 and P.2)
4. $\mathbf{C} \Rightarrow \mathbf{C} \Rightarrow \mathbf{C}$ (K with $[\mathbf{C}/P], [\mathbf{C}/Q]$)
5. $\mathbf{C} \Rightarrow \mathbf{C}$ (MP on P.3 and P.4)

This is indeed a very simple formal system, but it has all the required parts:

- A formal language: expressions built up from variables and implications.
- A semantics: given by the obvious interpretation function
- A calculus: given by the two axioms and the two inference rules.

The calculus gives us a set of rules with which we can derive new formulae from old ones. The axioms are very simple rules, they allow us to derive these two formulae in any situation. The proper inference rules are slightly more complicated: we read the formulae above the horizontal line as assumptions and the (single) formula below as the conclusion. An inference rule allows us to derive the conclusion, if we have already derived the assumptions.

Now, we can use these inference rules to perform a proof – a sequence of formulae that can be derived from each other. The representation of the proof in the slide is slightly compactified to fit onto the slide: We will make it more explicit here. We first start out by deriving the formula

$$(P \Rightarrow Q \Rightarrow R) \Rightarrow (P \Rightarrow Q) \Rightarrow P \Rightarrow R \tag{10.1}$$

which we can always do, since we have an axiom for this formula, then we apply the rule Subst, where $\mathbf{A}$ is this result, $\mathbf{B}$ is $\mathbf{C}$, and X is the variable P to obtain

$$(\mathbf{C} \Rightarrow Q \Rightarrow R) \Rightarrow (\mathbf{C} \Rightarrow Q) \Rightarrow \mathbf{C} \Rightarrow R \tag{10.2}$$

Next we apply the rule Subst to this where $\mathbf{B}$ is $\mathbf{C} \Rightarrow \mathbf{C}$ and X is the variable Q this time to obtain

$$(\mathbf{C} \Rightarrow (\mathbf{C} \Rightarrow \mathbf{C}) \Rightarrow R) \Rightarrow (\mathbf{C} \Rightarrow \mathbf{C} \Rightarrow \mathbf{C}) \Rightarrow \mathbf{C} \Rightarrow R \tag{10.3}$$

And again, we apply the rule Subst this time, $\mathbf{B}$ is $\mathbf{C}$ and X is the variable R yielding the first formula in our proof on the slide. To conserve space, we have combined these three steps into one in the slide. The next steps are done in exactly the same way.

In general, formulae can be used to represent facts about the world as propositions; they have a semantics that is a mapping of formulae into the real world (propositions are mapped to truth values.) We have seen two relations on formulae: the entailment relation and the derivation relation. The first one is defined purely in terms of the semantics, the second one is given by a calculus, i.e. purely syntactically. Is there any relation between these relations?

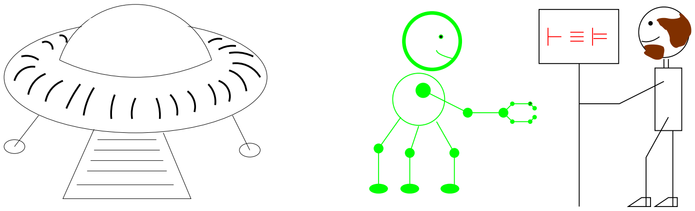
Soundness and Completeness

▷ **Definition 10.3.4.** Let $\mathcal{L} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system, then we call a calculus $\mathcal{C}$ for $\mathcal{L}$,

- ▷ sound (or correct), iff $\mathcal{H} \models \mathbf{A}$, whenever $\mathcal{H} \vdash_{\mathcal{C}} \mathbf{A}$, and
- ▷ complete, iff $\mathcal{H} \vdash_{\mathcal{C}} \mathbf{A}$, whenever $\mathcal{H} \models \mathbf{A}$.

▷ **Goal:** Find calculi C, such that $\vdash_C \mathbf{A}$ iff $\models \mathbf{A}$ (provability and validity coincide)

- ▷ To TRUTH through PROOF (CALCULEMUS [Leibniz ∼1680])



▷

Ideally, both relations would be the same, then the calculus would allow us to infer all facts that can be represented in the given formal language and that are true in the real world, and only those. In other words, our representation and inference is faithful to the world.

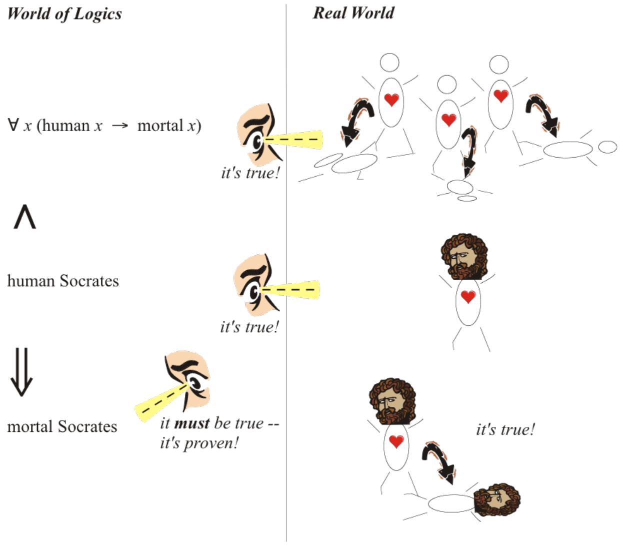
A consequence of this is that we can rely on purely syntactical means to make predictions about the world. Computers rely on formal representations of the world; if we want to solve a problem on our computer, we first represent it in the computer (as data structures, which can be seen as a formal language) and do syntactic manipulations on these structures (a form of calculus). Now, if the provability relation induced by the calculus and the validity relation coincide (this will be quite difficult to establish in general), then the solutions of the program will be correct, and we will find all possible ones. Of course, the logics we have studied so far are very simple, and not able to express interesting facts about the world, but we will study them as a simple example of the fundamental problem of computer science: How do the formal representations correlate with the real world.

Within the world of logics, one can derive new propositions (the *conclusions*, here: *Socrates is mortal*) from given ones (the *premises*, here: *Every human is mortal* and *Sokrates is human*). Such derivations are *proofs*.

In particular, logics can describe the internal structure of real-life facts; e.g. individual things, actions, properties. A famous example, which is in fact as old as it appears, is illustrated in the slide below.

The Miracle of Logic

▷ Purely formal derivations are true in the real world!



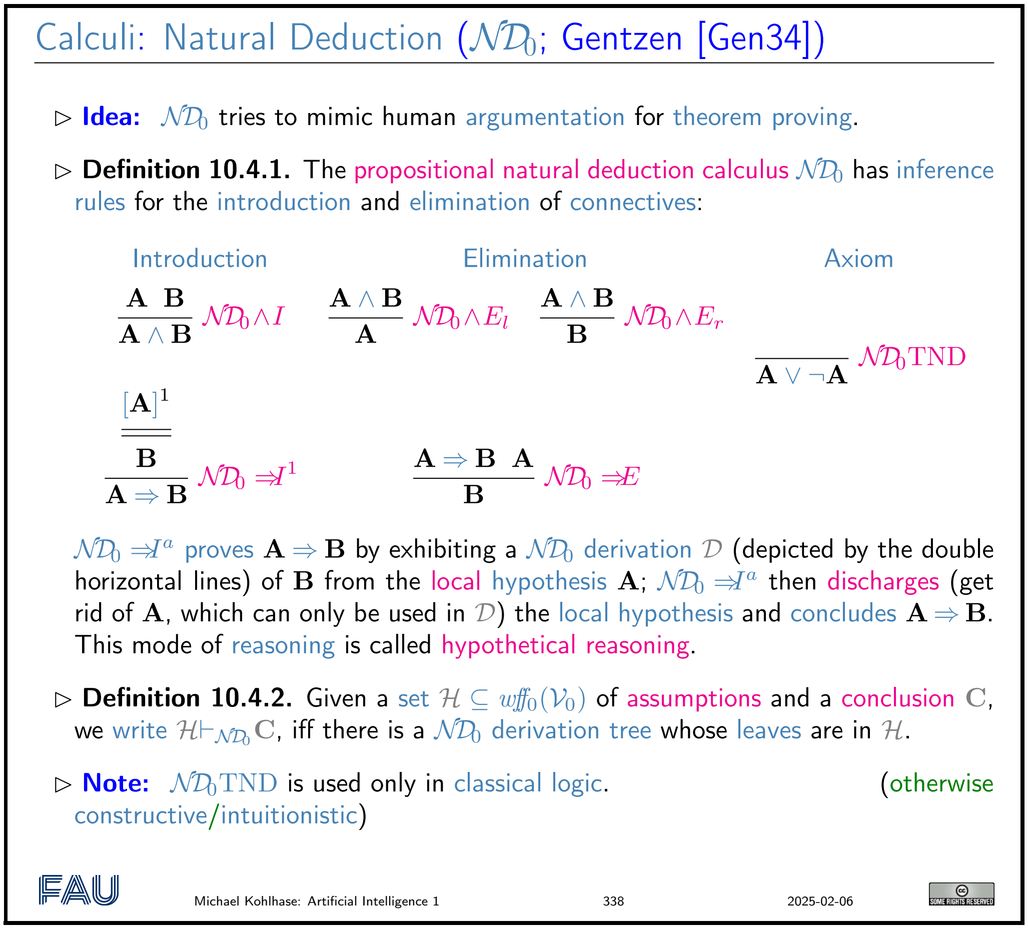
If a formal system is correct, the conclusions one can prove are true (= hold in the real world) whenever the premises are true. This is a miraculous fact (think about it!)

10.4 Propositional Natural Deduction Calculus

Video Nuggets covering this section can be found at `https://fau.tv/clip/id/22520` and `https://fau.tv/clip/id/22525`.

We will now introduce the "natural deduction" calculus for propositional logic. The calculus was created to model the natural mode of reasoning e.g. in everyday mathematical practice. In particular, it was intended as a counter-approach to the well-known Hilbert style calculi, which were mainly used as theoretical devices for studying reasoning in principle, not for modeling particular reasoning styles. We will introduce natural deduction in two styles/notations, both were invented by Gerhard Gentzen in the 1930's and are very much related. The Natural Deduction style (ND) uses local hypotheses in proofs for hypothetical reasoning, while the "sequent style" is a rationalized version and extension of the ND calculus that makes certain meta-proofs simpler to push through by making the context of local hypotheses explicit in the notation. The sequent notation also constitutes a more adequate data struture for implementations, and user interfaces.

Rather than using a minimal set of inference rules, we introduce a natural deduction calculus that provides two/three inference rules for every logical constant, one "introduction rule" (an inference rule that derives a formula with that logical constant at the head) and one "elimination rule" (an inference rule that acts on a formula with this head and derives a set of subformulae).



Calculi: Natural Deduction ($\mathcal{ND}_0$; Gentzen [Gen34])

- **Idea:** $\mathcal{ND}_0$ tries to mimic human argumentation for theorem proving.
- **Definition 10.4.1.** The propositional natural deduction calculus $\mathcal{ND}_0$ has inference rules for the introduction and elimination of connectives:

Introduction	Elimination	Axiom
$\dfrac{\mathbf{A}\quad\mathbf{B}}{\mathbf{A}\wedge\mathbf{B}}\,\mathcal{ND}_0{\wedge}I$	$\dfrac{\mathbf{A}\wedge\mathbf{B}}{\mathbf{A}}\,\mathcal{ND}_0{\wedge}E_l$ $\quad\dfrac{\mathbf{A}\wedge\mathbf{B}}{\mathbf{B}}\,\mathcal{ND}_0{\wedge}E_r$	$\dfrac{}{\mathbf{A}\vee\neg\mathbf{A}}\,\mathcal{ND}_0\mathrm{TND}$
$\dfrac{\begin{array}{c}[\mathbf{A}]^1\\ \hline\hline \mathbf{B}\end{array}}{\mathbf{A}\Rightarrow\mathbf{B}}\,\mathcal{ND}_0{\Rightarrow}I^1$	$\dfrac{\mathbf{A}\Rightarrow\mathbf{B}\quad\mathbf{A}}{\mathbf{B}}\,\mathcal{ND}_0{\Rightarrow}E$	

  $\mathcal{ND}_0{\Rightarrow}I^a$ proves $\mathbf{A}\Rightarrow\mathbf{B}$ by exhibiting a $\mathcal{ND}_0$ derivation $\mathcal{D}$ (depicted by the double horizontal lines) of $\mathbf{B}$ from the local hypothesis $\mathbf{A}$; $\mathcal{ND}_0{\Rightarrow}I^a$ then discharges (get rid of $\mathbf{A}$, which can only be used in $\mathcal{D}$) the local hypothesis and concludes $\mathbf{A}\Rightarrow\mathbf{B}$. This mode of reasoning is called hypothetical reasoning.

- **Definition 10.4.2.** Given a set $\mathcal{H}\subseteq wff_0(\mathcal{V}_0)$ of assumptions and a conclusion $\mathbf{C}$, we write $\mathcal{H}\vdash_{\mathcal{ND}_0}\mathbf{C}$, iff there is a $\mathcal{ND}_0$ derivation tree whose leaves are in $\mathcal{H}$.
- **Note:** $\mathcal{ND}_0\mathrm{TND}$ is used only in classical logic. (otherwise constructive/intuitionistic)

The most characteristic rule in the natural deduction calculus is the $\mathcal{ND}_0{\Rightarrow}I^a$ rule and the hypothetical reasoning it introduce. $\mathcal{ND}_0{\Rightarrow}I^a$ corresponds to the mathematical way of proving an implication $\mathbf{A}\Rightarrow\mathbf{B}$: We assume that $\mathbf{A}$ is true and show $\mathbf{B}$ from this local hypothesis. When we can do this we discharge the assumption and conclude $\mathbf{A}\Rightarrow\mathbf{B}$.

Note that the local hypothesis is discharged by the rule $\mathcal{ND}_0{\Rightarrow}I^a$, i.e. it cannot be used in any other part of the proof. As the $\mathcal{ND}_0{\Rightarrow}I^a$ rules may be nested, we decorate both the rule and the corresponding local hypothesis with a marker (here the number 1).

Let us now consider an example of hypothetical reasoning in action.

Natural Deduction: Examples

▷ **Example 10.4.3 (Inference with Local Hypotheses).**

$$\dfrac{\dfrac{\dfrac{[\mathbf{A}\wedge\mathbf{B}]^1}{\mathbf{B}}\mathcal{ND}_0\wedge E_r \quad \dfrac{[\mathbf{A}\wedge\mathbf{B}]^1}{\mathbf{A}}\mathcal{ND}_0\wedge E_l}{\mathbf{B}\wedge\mathbf{A}}\mathcal{ND}_0\wedge I}{\mathbf{A}\wedge\mathbf{B}\Rightarrow\mathbf{B}\wedge\mathbf{A}}\mathcal{ND}_0\Rightarrow I^1 \qquad \dfrac{\dfrac{\begin{array}{c}[A]^1\\ [B]^2\\ A\end{array}}{B\Rightarrow A}\mathcal{ND}_0\Rightarrow I^2}{A\Rightarrow B\Rightarrow A}\mathcal{ND}_0\Rightarrow I^1$$

Here we see hypothetical reasoning with local local hypotheses at work. In the left example, we assume the formula $\mathbf{A}\wedge\mathbf{B}$ and can use it in the proof until it is discharged by the rule $\mathcal{ND}_0\wedge E_l$ on the bottom – therefore we decorate the hypothesis and the rule by corresponding numbers (here the label "1"). Note the local assumption $\mathbf{A}\wedge\mathbf{B}$ is *local to the proof fragment* delineated by the corresponding (local) hypothesis and the discharging rule, i.e. even if this derivation is only a fragment of a larger proof, then we cannot use its (local) hypothesis anywhere else.

Note also that we can use as many copies of the local hypothesis as we need; they are all discharged at the same time.

In the right example we see that local hypotheses can be nested as long as they are kept local. In particular, we may not use the hypothesis $\mathbf{B}$ after the $\mathcal{ND}_0\Rightarrow I^2$, e.g. to continue with a $\mathcal{ND}_0\Rightarrow E$.

One of the nice things about the natural deduction calculus is that the deduction theorem is almost trivial to prove. In a sense, the triviality of the deduction theorem is the central idea of the calculus and the feature that makes it so natural.

A Deduction Theorem for $\mathcal{ND}_0$

▷ **Theorem 10.4.4.** $\mathcal{H},\mathbf{A}\vdash_{\mathcal{ND}_0}\mathbf{B}$, *iff* $\mathcal{H}\vdash_{\mathcal{ND}_0}\mathbf{A}\Rightarrow\mathbf{B}$.

▷ *Proof:* We show the two directions separately

1. If $\mathcal{H},\mathbf{A}\vdash_{\mathcal{ND}_0}\mathbf{B}$, then $\mathcal{H}\vdash_{\mathcal{ND}_0}\mathbf{A}\Rightarrow\mathbf{B}$ by $\mathcal{ND}_0\Rightarrow I$, and
2. If $\mathcal{H}\vdash_{\mathcal{ND}_0}\mathbf{A}\Rightarrow\mathbf{B}$, then $\mathcal{H},\mathbf{A}\vdash_{\mathcal{ND}_0}\mathbf{A}\Rightarrow\mathbf{B}$ by weakening and $\mathcal{H},\mathbf{A}\vdash_{\mathcal{ND}_0}\mathbf{B}$ by $\mathcal{ND}_0\Rightarrow E$.

Another characteristic of the natural deduction calculus is that it has inference rules (introduction and elimination rules) for all connectives. So we extend the set of rules from **??** for disjunction, negation and falsity.

More Rules for Natural Deduction

▷ **Note:** $\mathcal{ND}_0$ does not try to be minimal, but comfortable to work in!

▷ **Definition 10.4.5.** $\mathcal{ND}_0$ has the following additional inference rules for the remaining connectives.

$$\frac{\mathbf{A}}{\mathbf{A}\vee\mathbf{B}}\,\mathcal{ND}_0\vee I_l \qquad \frac{\mathbf{B}}{\mathbf{A}\vee\mathbf{B}}\,\mathcal{ND}_0\vee I_r \qquad \frac{\mathbf{A}\vee\mathbf{B}\quad\begin{array}{c}[\mathbf{A}]^1\\ \vdots\\ \mathbf{C}\end{array}\quad\begin{array}{c}[\mathbf{B}]^1\\ \vdots\\ \mathbf{C}\end{array}}{\mathbf{C}}\,\mathcal{ND}_0\vee E^1$$

$$\frac{\begin{array}{c}[\mathbf{A}]^1\\ \vdots\\ \mathbf{C}\end{array}\quad\begin{array}{c}[\mathbf{A}]^1\\ \vdots\\ \neg\mathbf{C}\end{array}}{\neg\mathbf{A}}\,\mathcal{ND}_0\neg I^1 \qquad \frac{\neg\neg\mathbf{A}}{\mathbf{A}}\,\mathcal{ND}_0\neg E$$

$$\frac{\neg\mathbf{A}\quad\mathbf{A}}{F}\,\mathcal{ND}_0 FI \qquad \frac{F}{\mathbf{A}}\,\mathcal{ND}_0 FE$$

- **Again:** $\mathcal{ND}_0\neg E$ is used only in classical logic (otherwise constructive/intuitionistic)

Natural Deduction in Sequent Calculus Formulation

- **Idea:** Represent hypotheses explicitly. (lift calculus to judgments)
- **Definition 10.4.6.** A judgment is a meta-statement about the provability of propositions.
- **Definition 10.4.7.** A sequent is a judgment of the form $\mathcal{H}\vdash\mathbf{A}$ about the provability of the formula $\mathbf{A}$ from the set $\mathcal{H}$ of hypotheses. We write $\vdash\mathbf{A}$ for $\emptyset\vdash\mathbf{A}$.
- **Idea:** Reformulate $\mathcal{ND}_0$ inference rules so that they act on sequents.
- **Example 10.4.8.** We give the sequent style version of ??:

$$\dfrac{\dfrac{\dfrac{}{\mathbf{A}\wedge\mathbf{B}\vdash\mathbf{A}\wedge\mathbf{B}}\,\mathcal{ND}^0_\vdash\mathrm{Ax}}{\mathbf{A}\wedge\mathbf{B}\vdash\mathbf{B}}\,\mathcal{ND}^0_\vdash\wedge E_r\qquad\dfrac{\dfrac{}{\mathbf{A}\wedge\mathbf{B}\vdash\mathbf{A}\wedge\mathbf{B}}\,\mathcal{ND}^0_\vdash\mathrm{Ax}}{\mathbf{A}\wedge\mathbf{B}\vdash\mathbf{A}}\,\mathcal{ND}^0_\vdash\wedge E_l}{\dfrac{\mathbf{A}\wedge\mathbf{B}\vdash\mathbf{B}\wedge\mathbf{A}}{\vdash\mathbf{A}\wedge\mathbf{B}\Rightarrow\mathbf{B}\wedge\mathbf{A}}\,\mathcal{ND}^0_\vdash\Rightarrow I}\,\mathcal{ND}^0_\vdash\wedge I \qquad \dfrac{\dfrac{\dfrac{}{\mathbf{A},\mathbf{B}\vdash\mathbf{A}}\,\mathcal{ND}^0_\vdash\mathrm{Ax}}{\mathbf{A}\vdash\mathbf{B}\Rightarrow\mathbf{A}}\,\mathcal{ND}^0_\vdash\Rightarrow I}{\vdash\mathbf{A}\Rightarrow\mathbf{B}\Rightarrow\mathbf{A}}\,\mathcal{ND}^0_\vdash\Rightarrow I$$

- **Note:** Even though the antecedent of a sequent is written like a sequences, it is actually a set. In particular, we can permute and duplicate members at will.

Sequent-Style Rules for Natural Deduction

▷ **Definition 10.4.9.** The following inference rules make up the propositional sequent style natural deduction calculus $\mathcal{ND}^0_\vdash$:

$$\frac{}{\Gamma, \mathbf{A}\vdash\mathbf{A}}\,\mathcal{ND}^0_\vdash\mathrm{Ax} \qquad \frac{\Gamma\vdash\mathbf{B}}{\Gamma, \mathbf{A}\vdash\mathbf{B}}\,\mathcal{ND}^0_\vdash\mathrm{weaken} \qquad \frac{}{\Gamma\vdash\mathbf{A}\vee\neg\mathbf{A}}\,\mathcal{ND}^0_\vdash\mathrm{TND}$$

$$\frac{\Gamma\vdash\mathbf{A}\quad\Gamma\vdash\mathbf{B}}{\Gamma\vdash\mathbf{A}\wedge\mathbf{B}}\,\mathcal{ND}^0_\vdash\wedge I \qquad \frac{\Gamma\vdash\mathbf{A}\wedge\mathbf{B}}{\Gamma\vdash\mathbf{A}}\,\mathcal{ND}^0_\vdash\wedge E_l \qquad \frac{\Gamma\vdash\mathbf{A}\wedge\mathbf{B}}{\Gamma\vdash\mathbf{B}}\,\mathcal{ND}^0_\vdash\wedge E_r$$

$$\frac{\Gamma\vdash\mathbf{A}}{\Gamma\vdash\mathbf{A}\vee\mathbf{B}}\,\mathcal{ND}^0_\vdash\vee I_l \qquad \frac{\Gamma\vdash\mathbf{B}}{\Gamma\vdash\mathbf{A}\vee\mathbf{B}}\,\mathcal{ND}^0_\vdash\vee I_r \qquad \frac{\Gamma\vdash\mathbf{A}\vee\mathbf{B}\quad\Gamma, \mathbf{A}\vdash\mathbf{C}\quad\Gamma, \mathbf{B}\vdash\mathbf{C}}{\Gamma\vdash\mathbf{C}}\,\mathcal{ND}^0_\vdash\vee E$$

$$\frac{\Gamma, \mathbf{A}\vdash\mathbf{B}}{\Gamma\vdash\mathbf{A}\Rightarrow\mathbf{B}}\,\mathcal{ND}^0_\vdash\Rightarrow I \qquad \frac{\Gamma\vdash\mathbf{A}\Rightarrow\mathbf{B}\quad\Gamma\vdash\mathbf{A}}{\Gamma\vdash\mathbf{B}}\,\mathcal{ND}^0_\vdash\Rightarrow E$$

$$\frac{\Gamma, \mathbf{A}\vdash F}{\Gamma\vdash\neg\mathbf{A}}\,\mathcal{ND}^0_\vdash\neg I \qquad \frac{\Gamma\vdash\neg\neg\mathbf{A}}{\Gamma\vdash\mathbf{A}}\,\mathcal{ND}^0_\vdash\neg E$$

$$\mathcal{ND}^0_\vdash FI\,\frac{\Gamma\vdash\neg\mathbf{A}\quad\Gamma\vdash\mathbf{A}}{\Gamma\vdash F} \qquad \mathcal{ND}^0_\vdash FE\,\frac{\Gamma\vdash F}{\Gamma\vdash\mathbf{A}}$$

Linearized Notation for (Sequent-Style) ND Proofs

▷ **Definition 10.4.10.** Linearized notation for sequent-style ND proofs

1. $\mathcal{H}_1 \vdash \mathbf{A}_1 \; (\mathcal{J}_1)$
2. $\mathcal{H}_2 \vdash \mathbf{A}_2 \; (\mathcal{J}_2)$
3. $\mathcal{H}_3 \vdash \mathbf{A}_3 \; (\mathcal{J}_3 1, 2)$

corresponds to

$$\frac{\mathcal{H}_1\vdash\mathbf{A}_1\quad\mathcal{H}_2\vdash\mathbf{A}_2}{\mathcal{H}_3\vdash\mathbf{A}_3}\,\mathcal{R}$$

▷ **Example 10.4.11.** We show a linearized version of the $\mathcal{ND}_0$ examples ??

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{A}\wedge\mathbf{B}\vdash\mathbf{A}\wedge\mathbf{B}}\,\mathcal{ND}^0_\vdash\mathrm{Ax}}{\mathbf{A}\wedge\mathbf{B}\vdash\mathbf{B}}\,\mathcal{ND}^0_\vdash\wedge E_r \quad \dfrac{\dfrac{}{\mathbf{A}\wedge\mathbf{B}\vdash\mathbf{A}\wedge\mathbf{B}}\,\mathcal{ND}^0_\vdash\mathrm{Ax}}{\mathbf{A}\wedge\mathbf{B}\vdash\mathbf{A}}\,\mathcal{ND}^0_\vdash\wedge E_l}{\mathbf{A}\wedge\mathbf{B}\vdash\mathbf{B}\wedge\mathbf{A}}\,\mathcal{ND}^0_\vdash\wedge I}{\vdash\mathbf{A}\wedge\mathbf{B}\Rightarrow\mathbf{B}\wedge\mathbf{A}}\,\mathcal{ND}^0_\vdash\Rightarrow I \qquad \dfrac{\dfrac{\dfrac{}{\mathbf{A}, \mathbf{B}\vdash\mathbf{A}}\,\mathcal{ND}^0_\vdash\mathrm{Ax}}{\mathbf{A}\vdash\mathbf{B}\Rightarrow\mathbf{A}}\,\mathcal{ND}^0_\vdash\Rightarrow I}{\vdash\mathbf{A}\Rightarrow\mathbf{B}\Rightarrow\mathbf{A}}\,\mathcal{ND}^0_\vdash\Rightarrow I$$

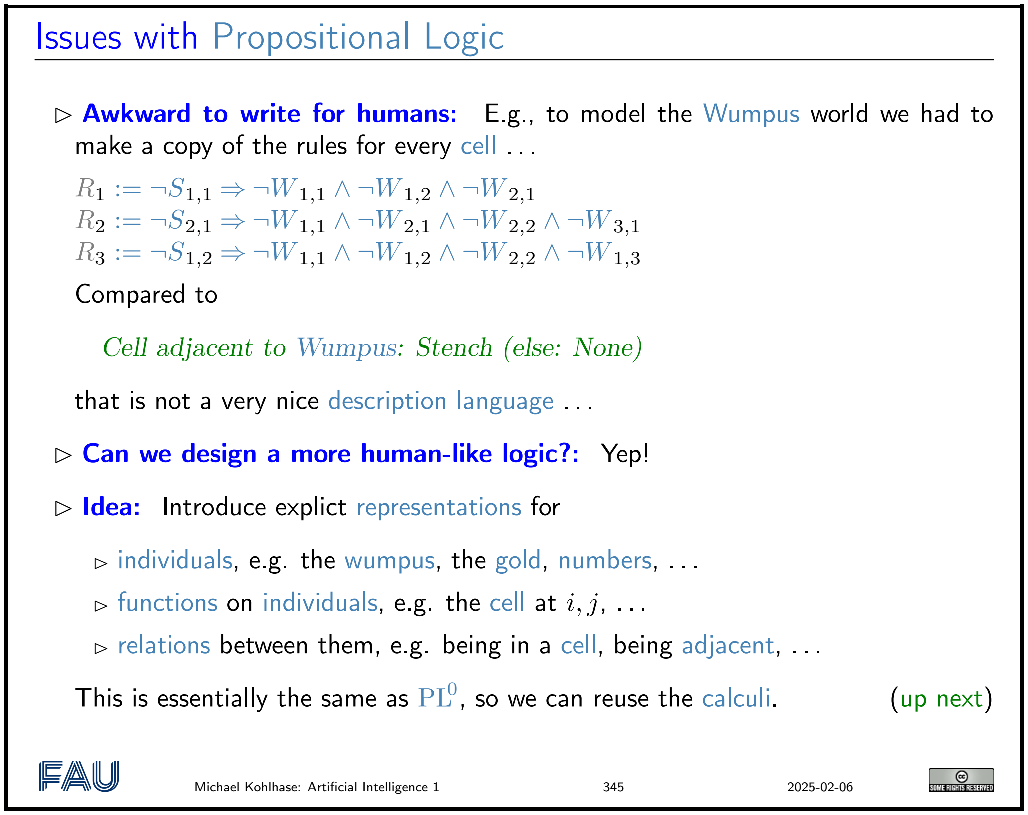
$\#$	hyp	$\vdash$	$formula$	$NDjust$
1.	1	$\vdash$	$\mathbf{A}\wedge\mathbf{B}$	$\mathcal{ND}^0_\vdash\mathrm{Ax}$
2.	1	$\vdash$	$\mathbf{B}$	$\mathcal{ND}^0_\vdash\wedge E_r\ 1$
3.	1	$\vdash$	$\mathbf{A}$	$\mathcal{ND}^0_\vdash\wedge E_l\ 1$
4.	1	$\vdash$	$\mathbf{B}\wedge\mathbf{A}$	$\mathcal{ND}^0_\vdash\wedge I\ 2, 3$
5.		$\vdash$	$\mathbf{A}\wedge\mathbf{B}\Rightarrow\mathbf{B}\wedge\mathbf{A}$	$\mathcal{ND}^0_\vdash\Rightarrow I\ 4$

$\#$	hyp	$\vdash$	$formula$	$NDjust$
1.	1	$\vdash$	$\mathbf{A}$	$\mathcal{ND}^0_\vdash\mathrm{Ax}$
2.	2	$\vdash$	$\mathbf{B}$	$\mathcal{ND}^0_\vdash\mathrm{Ax}$
3.	1, 2	$\vdash$	$\mathbf{A}$	$\mathcal{ND}^0_\vdash\mathrm{weaken}\ 1, 2$
4.	1	$\vdash$	$\mathbf{B}\Rightarrow\mathbf{A}$	$\mathcal{ND}^0_\vdash\Rightarrow I\ 3$
5.		$\vdash$	$\mathbf{A}\Rightarrow\mathbf{B}\Rightarrow\mathbf{A}$	$\mathcal{ND}^0_\vdash\Rightarrow I\ 4$

Each row in the table represents one inference step in the proof. It consists of line number (for referencing), a formula for the statement, a justification via a ND inference rule (and the rows this one is derived from), and finally a sequence of row numbers of proof steps that are local hypotheses in effect for the current row.

10.5 Predicate Logic Without Quantifiers

In the hair-color example we have seen that we are able to model complex situations in PL^0. The trick of using variables with fancy names like $bla(N)$ is a bit dubious, and we can already imagine that it will be difficult to support programmatically unless we make names like $bla(N)$ into first-class citizens i.e. expressions of the logic language themselves.



Issues with Propositional Logic

- ▷ **Awkward to write for humans:** E.g., to model the Wumpus world we had to make a copy of the rules for every cell ...

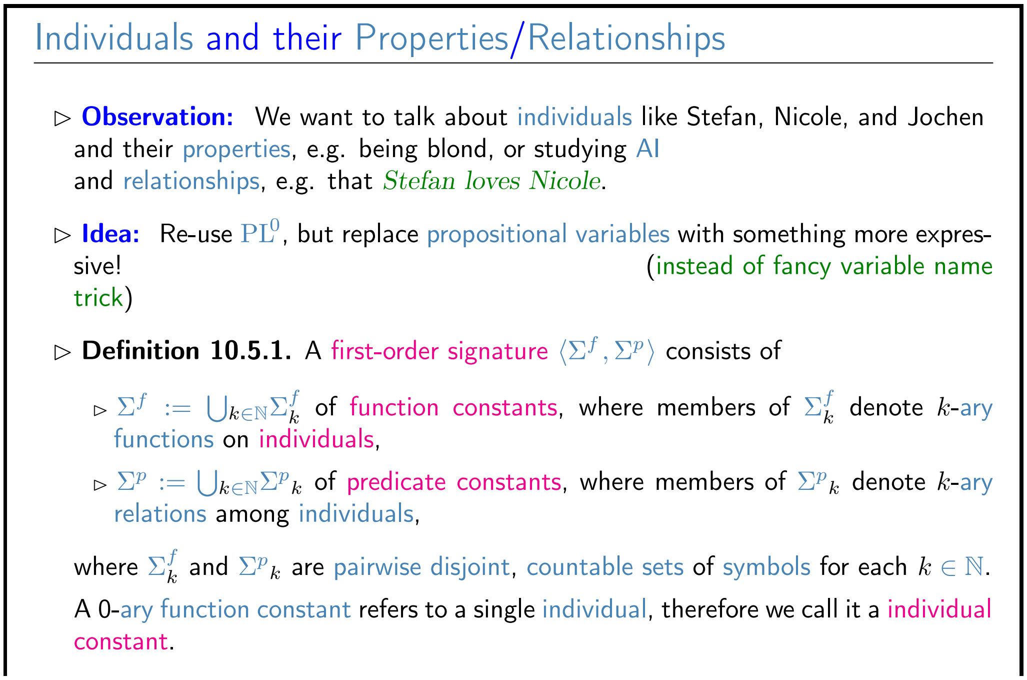
  $R_1 := \neg S_{1,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,1}$
  $R_2 := \neg S_{2,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{2,1} \wedge \neg W_{2,2} \wedge \neg W_{3,1}$
  $R_3 := \neg S_{1,2} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,2} \wedge \neg W_{1,3}$

  Compared to

  Cell adjacent to Wumpus: Stench (else: None)

  that is not a very nice description language ...
- ▷ **Can we design a more human-like logic?:** Yep!
- ▷ **Idea:** Introduce explict representations for
  - ▷ individuals, e.g. the wumpus, the gold, numbers, ...
  - ▷ functions on individuals, e.g. the cell at i, j, ...
  - ▷ relations between them, e.g. being in a cell, being adjacent, ...

  This is essentially the same as PL^0, so we can reuse the calculi. (up next)



Individuals and their Properties/Relationships

- ▷ **Observation:** We want to talk about individuals like Stefan, Nicole, and Jochen and their properties, e.g. being blond, or studying AI
  and relationships, e.g. that *Stefan loves Nicole*.
- ▷ **Idea:** Re-use PL^0, but replace propositional variables with something more expressive! (instead of fancy variable name trick)
- ▷ **Definition 10.5.1.** A first-order signature $\langle \Sigma^f, \Sigma^p \rangle$ consists of
  - ▷ $\Sigma^f := \bigcup_{k\in\mathbb{N}} \Sigma^f_k$ of function constants, where members of Σ^f_k denote k-ary functions on individuals,
  - ▷ $\Sigma^p := \bigcup_{k\in\mathbb{N}} {\Sigma^p}_k$ of predicate constants, where members of ${\Sigma^p}_k$ denote k-ary relations among individuals,

  where Σ^f_k and ${\Sigma^p}_k$ are pairwise disjoint, countable sets of symbols for each $k \in \mathbb{N}$.

  A 0-ary function constant refers to a single individual, therefore we call it a individual constant.

A Grammar for $\mathrm{PL}^{\mathrm{nq}}$

▷ **Definition 10.5.2.** The formulae of $\mathrm{PL}^{\mathrm{nq}}$ are given by the following grammar

function constants	f^k	$\in$	Σ^f_k	
predicate constants	p^k	$\in$	$\Sigma^p{}_k$	
terms	t	$::=$	f^0	individualconstant
		$\mid$	$f^k(t_1,\ldots,t_k)$	application
formulae	$\mathbf{A}$	$::=$	$p^k(t_1,\ldots,t_k)$	atomic
		$\mid$	$\neg\mathbf{A}$	negation
		$\mid$	$\mathbf{A}_1 \wedge \mathbf{A}_2$	conjunction

$\mathrm{PL}^{\mathrm{nq}}$ Semantics

▷ **Definition 10.5.3.** Domains $\mathcal{D}_0 = \{\mathsf{T},\mathsf{F}\}$ of truth values and $\mathcal{D}_\iota \neq \emptyset$ of individuals.

▷ **Definition 10.5.4.** Interpretation $\mathcal{I}$ assigns values to constants, e.g.

- ▷ $\mathcal{I}(\neg)\colon \mathcal{D}_0 \to \mathcal{D}_0; \mathsf{T} \mapsto \mathsf{F}; \mathsf{F} \mapsto \mathsf{T}$ and $\mathcal{I}(\wedge) = \ldots$ (as in PL^0)
- ▷ $\mathcal{I}\colon \Sigma^f_0 \to \mathcal{D}_\iota$ (interpret individual constants as individuals)
- ▷ $\mathcal{I}\colon \Sigma^f_k \to {\mathcal{D}_\iota}^k \to \mathcal{D}_\iota$ (interpret function constants as functions)
- ▷ $\mathcal{I}\colon \Sigma^p{}_k \to \mathcal{P}({\mathcal{D}_\iota}^k)$ (interpret predicate constants as relations)

▷ **Definition 10.5.5.** The value function $\mathcal{I}$ assigns values to formulae: (recursively)

- ▷ $\mathcal{I}(f(\mathbf{A}^1,\ldots,\mathbf{A}^k)) := \mathcal{I}(f)(\mathcal{I}(\mathbf{A}^1),\ldots,\mathcal{I}(\mathbf{A}^k))$
- ▷ $\mathcal{I}(p(\mathbf{A}^1,\ldots,\mathbf{A}^k)) := \mathsf{T}$, iff $\langle\mathcal{I}(\mathbf{A}^1),\ldots,\mathcal{I}(\mathbf{A}^k)\rangle \in \mathcal{I}(p)$
- ▷ $\mathcal{I}(\neg\mathbf{A}) = \mathcal{I}(\neg)(\mathcal{I}(\mathbf{A}))$ and $\mathcal{I}(\mathbf{A}\wedge\mathbf{B}) = \mathcal{I}(\wedge)(\mathcal{I}(\mathbf{A}),\mathcal{I}(\mathbf{G}))$ (just as in PL^0)

▷ **Definition 10.5.6.** Model: $\mathcal{M} = \langle\mathcal{D}_\iota,\mathcal{I}\rangle$ varies in $\mathcal{D}_\iota$ and $\mathcal{I}$.

▷ **Theorem 10.5.7.** *$\mathrm{PL}^{\mathrm{nq}}$ is isomorphic to PL^0* (*interpret atoms as prop. variables*)

All of the definitions above are quite abstract, we now look at them again using a very concrete – if somewhat contrived – example: The relevant parts are a universe $\mathcal{D}$ with four elements, and an interpretation that maps the signature into individuals, functions, and predicates over $\mathcal{D}$, which are given as concrete sets.

A Model for $\mathrm{PL}^{\mathrm{nq}}$

▷ **Example 10.5.8.** Let $L := \{a,b,c,d,e,P,Q,R,S\}$, we set the universe $\mathcal{D} := \{\clubsuit,\spadesuit,\heartsuit,\diamondsuit\}$, and specify the interpretation function $\mathcal{I}$ by setting

- ▷ $a \mapsto \clubsuit$, $b \mapsto \spadesuit$, $c \mapsto \heartsuit$, $d \mapsto \diamondsuit$, and $e \mapsto \diamondsuit$ for constants,

- ▷ $P \mapsto \{\clubsuit, \spadesuit\}$ and $Q \mapsto \{\spadesuit, \diamondsuit\}$, for unary predicate constants.
- ▷ $R \mapsto \{\langle\heartsuit, \diamondsuit\rangle, \langle\diamondsuit, \heartsuit\rangle\}$, and $S \mapsto \{\langle\diamondsuit, \spadesuit\rangle, \langle\spadesuit, \clubsuit\rangle\}$ for binary predicate constants.

▷ **Example 10.5.9 (Computing Meaning in this Model).**

- ▷ $\mathcal{I}(R(a,b) \wedge P(c)) = \mathsf{T}$, iff
- ▷ $\mathcal{I}(R(a,b)) = \mathsf{T}$ and $\mathcal{I}(P(c)) = \mathsf{T}$, iff
- ▷ $\langle\mathcal{I}(a), \mathcal{I}(b)\rangle \in \mathcal{I}(R)$ and $\mathcal{I}(c) \in \mathcal{I}(P)$, iff
- ▷ $\langle\clubsuit, \spadesuit\rangle \in \{\langle\heartsuit, \diamondsuit\rangle, \langle\diamondsuit, \heartsuit\rangle\}$ and $\heartsuit \in \{\clubsuit, \spadesuit\}$

So, $\mathcal{I}(R(a,b) \wedge P(c)) = \mathsf{F}$.

The example above also shows how we can compute of meaning by in a concrete model: we just follow the evaluation rules to the letter.

We now come to the central technical result about $\mathrm{PL}^{\mathrm{nq}}$: it is essentially the same as propositional logic (PL^0). We say that the two logic are isomorphic. Technically, this means that the formulae of $\mathrm{PL}^{\mathrm{nq}}$ can be translated to PL^0 and there is a corresponding model translation from the models of PL^0 to those of $\mathrm{PL}^{\mathrm{nq}}$ such that the respective notions of evaluation are assignped to each other.

$\mathrm{PL}^{\mathrm{nq}}$ and PL^0 are Isomorphic

▷ **Observation:** For every choice of Σ of signature, the set $\mathcal{A}_\Sigma$ of atomic $\mathrm{PL}^{\mathrm{nq}}$ formulae is countable, so there is a $\mathcal{V}_\Sigma \subseteq \mathcal{V}_0$ and a bijection $\theta_\Sigma \colon \mathcal{A}_\Sigma \to \mathcal{V}_\Sigma$.

θ_Σ can be extended to formulae as $\mathrm{PL}^{\mathrm{nq}}$ and PL^0 share connectives.

▷ **Lemma 10.5.10.** *For every model $\mathcal{M} = \langle\mathcal{D}_\iota, \mathcal{I}\rangle$, there is a variable assignment $\varphi_\mathcal{M}$, such that $\mathcal{I}_{\varphi_\mathcal{M}}(\mathbf{A}) = \mathcal{I}(\mathbf{A})$.*

▷ *Proof sketch:* We just define $\varphi_\mathcal{M}(X) := \mathcal{I}(\theta_\Sigma^{-1}(X))$

▷ **Lemma 10.5.11.** *For every variable assignment $\psi \colon \mathcal{V}_\Sigma \to \{\mathsf{T}, \mathsf{F}\}$ there is a model $\mathcal{M}^\psi = \langle\mathcal{D}^\psi, \mathcal{I}^\psi\rangle$, such that $\mathcal{I}_\psi(\mathbf{A}) = \mathcal{I}^\psi(\mathbf{A})$.*

▷ *Proof sketch:* see next slide

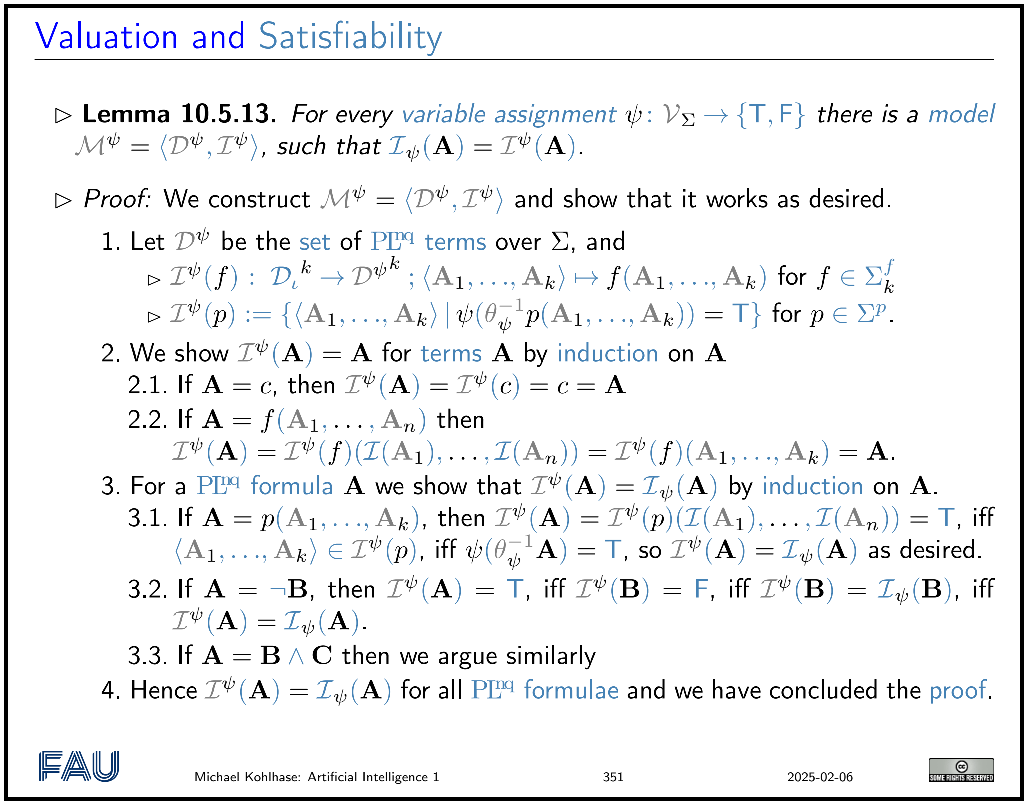
▷ **Corollary 10.5.12.** *$\mathrm{PL}^{\mathrm{nq}}$ is isomorphic to PL^0, i.e. the following diagram commutes:*

$$\begin{tikzcd}
\langle\mathcal{D}^\psi, \mathcal{I}^\psi\rangle & \mathcal{V}_\Sigma \to \{\mathsf{T}, \mathsf{F}\} \arrow[l, "\psi \mapsto \mathcal{M}^\psi"'] \\
\mathrm{PL}^{\mathrm{nq}}(\Sigma) \arrow[u, "\mathcal{I}^\psi()"] \arrow[r, "\theta_\Sigma"] & \mathrm{PL}^0(\mathcal{A}_\Sigma) \arrow[u, "\mathcal{I}_{\varphi_\mathcal{M}}()"']
\end{tikzcd}$$

▷ **Note:** This constellation with a language isomorphism and a corresponding model isomorphism (in converse direction) is typical for a logic isomorphism.

The practical upshot of the commutative diagram from **??** is that if we have a way of computing evaluation (or entailment for that matter) in PL^0, then we can "borrow" it for $\mathrm{PL}^{\mathrm{nq}}$ by composing it with the language and model translations. In other words, we can reuse calculi and automated

theorem provers from PL^0 for $\mathrm{PL}^{\mathrm{nq}}$.
But we still have to provide the proof for ??, which we do now.



Valuation and Satisfiability

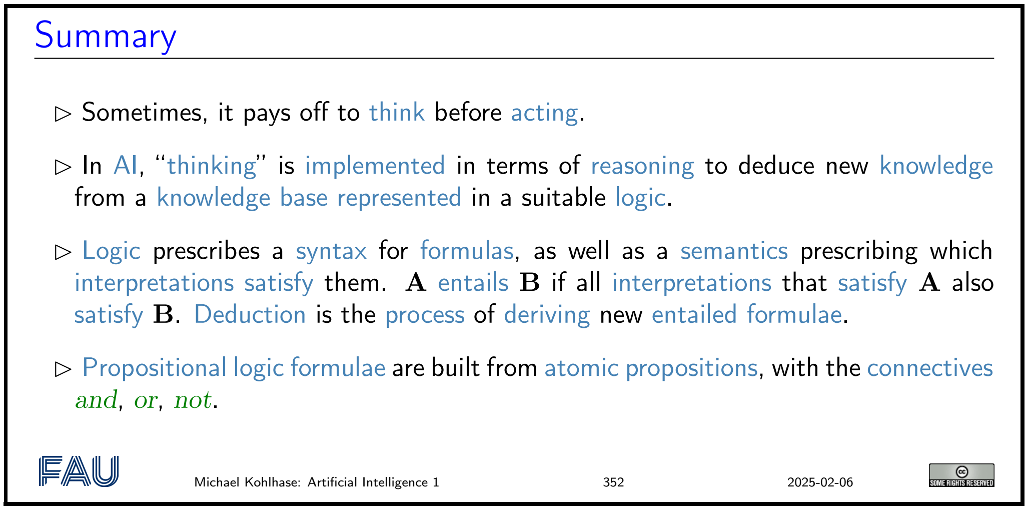
▷ **Lemma 10.5.13.** *For every variable assignment $\psi\colon \mathcal{V}_\Sigma \to \{\mathsf{T},\mathsf{F}\}$ there is a model $\mathcal{M}^\psi = \langle \mathcal{D}^\psi, \mathcal{I}^\psi \rangle$, such that $\mathcal{I}_\psi(\mathbf{A}) = \mathcal{I}^\psi(\mathbf{A})$.*

▷ *Proof:* We construct $\mathcal{M}^\psi = \langle \mathcal{D}^\psi, \mathcal{I}^\psi \rangle$ and show that it works as desired.

1. Let $\mathcal{D}^\psi$ be the set of $\mathrm{PL}^{\mathrm{nq}}$ terms over Σ, and
   ▷ $\mathcal{I}^\psi(f) : {\mathcal{D}_\iota}^k \to {\mathcal{D}^\psi}^k ; \langle \mathbf{A}_1, \ldots, \mathbf{A}_k \rangle \mapsto f(\mathbf{A}_1, \ldots, \mathbf{A}_k)$ for $f \in \Sigma^f_k$
   ▷ $\mathcal{I}^\psi(p) := \{\langle \mathbf{A}_1, \ldots, \mathbf{A}_k \rangle \mid \psi(\theta_\psi^{-1} p(\mathbf{A}_1, \ldots, \mathbf{A}_k)) = \mathsf{T}\}$ for $p \in \Sigma^p$.
2. We show $\mathcal{I}^\psi(\mathbf{A}) = \mathbf{A}$ for terms $\mathbf{A}$ by induction on $\mathbf{A}$
   2.1. If $\mathbf{A} = c$, then $\mathcal{I}^\psi(\mathbf{A}) = \mathcal{I}^\psi(c) = c = \mathbf{A}$
   2.2. If $\mathbf{A} = f(\mathbf{A}_1, \ldots, \mathbf{A}_n)$ then
   $\mathcal{I}^\psi(\mathbf{A}) = \mathcal{I}^\psi(f)(\mathcal{I}(\mathbf{A}_1), \ldots, \mathcal{I}(\mathbf{A}_n)) = \mathcal{I}^\psi(f)(\mathbf{A}_1, \ldots, \mathbf{A}_k) = \mathbf{A}$.
3. For a $\mathrm{PL}^{\mathrm{nq}}$ formula $\mathbf{A}$ we show that $\mathcal{I}^\psi(\mathbf{A}) = \mathcal{I}_\psi(\mathbf{A})$ by induction on $\mathbf{A}$.
   3.1. If $\mathbf{A} = p(\mathbf{A}_1, \ldots, \mathbf{A}_k)$, then $\mathcal{I}^\psi(\mathbf{A}) = \mathcal{I}^\psi(p)(\mathcal{I}(\mathbf{A}_1), \ldots, \mathcal{I}(\mathbf{A}_n)) = \mathsf{T}$, iff $\langle \mathbf{A}_1, \ldots, \mathbf{A}_k \rangle \in \mathcal{I}^\psi(p)$, iff $\psi(\theta_\psi^{-1}\mathbf{A}) = \mathsf{T}$, so $\mathcal{I}^\psi(\mathbf{A}) = \mathcal{I}_\psi(\mathbf{A})$ as desired.
   3.2. If $\mathbf{A} = \neg\mathbf{B}$, then $\mathcal{I}^\psi(\mathbf{A}) = \mathsf{T}$, iff $\mathcal{I}^\psi(\mathbf{B}) = \mathsf{F}$, iff $\mathcal{I}^\psi(\mathbf{B}) = \mathcal{I}_\psi(\mathbf{B})$, iff $\mathcal{I}^\psi(\mathbf{A}) = \mathcal{I}_\psi(\mathbf{A})$.
   3.3. If $\mathbf{A} = \mathbf{B} \wedge \mathbf{C}$ then we argue similarly
4. Hence $\mathcal{I}^\psi(\mathbf{A}) = \mathcal{I}_\psi(\mathbf{A})$ for all $\mathrm{PL}^{\mathrm{nq}}$ formulae and we have concluded the proof.

10.6 Conclusion

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/25027`.



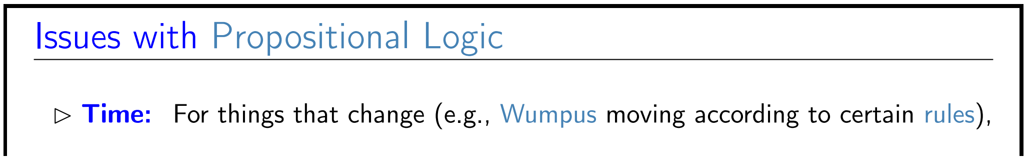
Summary

▷ Sometimes, it pays off to think before acting.

▷ In AI, "thinking" is implemented in terms of reasoning to deduce new knowledge from a knowledge base represented in a suitable logic.

▷ Logic prescribes a syntax for formulas, as well as a semantics prescribing which interpretations satisfy them. $\mathbf{A}$ entails $\mathbf{B}$ if all interpretations that satisfy $\mathbf{A}$ also satisfy $\mathbf{B}$. Deduction is the process of deriving new entailed formulae.

▷ Propositional logic formulae are built from atomic propositions, with the connectives *and*, *or*, *not*.



Issues with Propositional Logic

▷ **Time:** For things that change (e.g., Wumpus moving according to certain rules), we need time-indexed propositions (like, $S_{2,1}^{t=7}$) to represent validity over time $\rightsquigarrow$ further expansion of the rules.

- ▷ **Can we design a more human-like logic?:** Yep
  - ▷ Predicate logic: quantification of variables ranging over individuals. (cf. **??** and **??**)
  - ▷ . . . and a whole zoo of logics much more powerful still.
  - ▷ Note: In applications, propositional CNF encodings are generated by computer programs. This mitigates (but does not remove!) the inconveniences of propositional modeling.

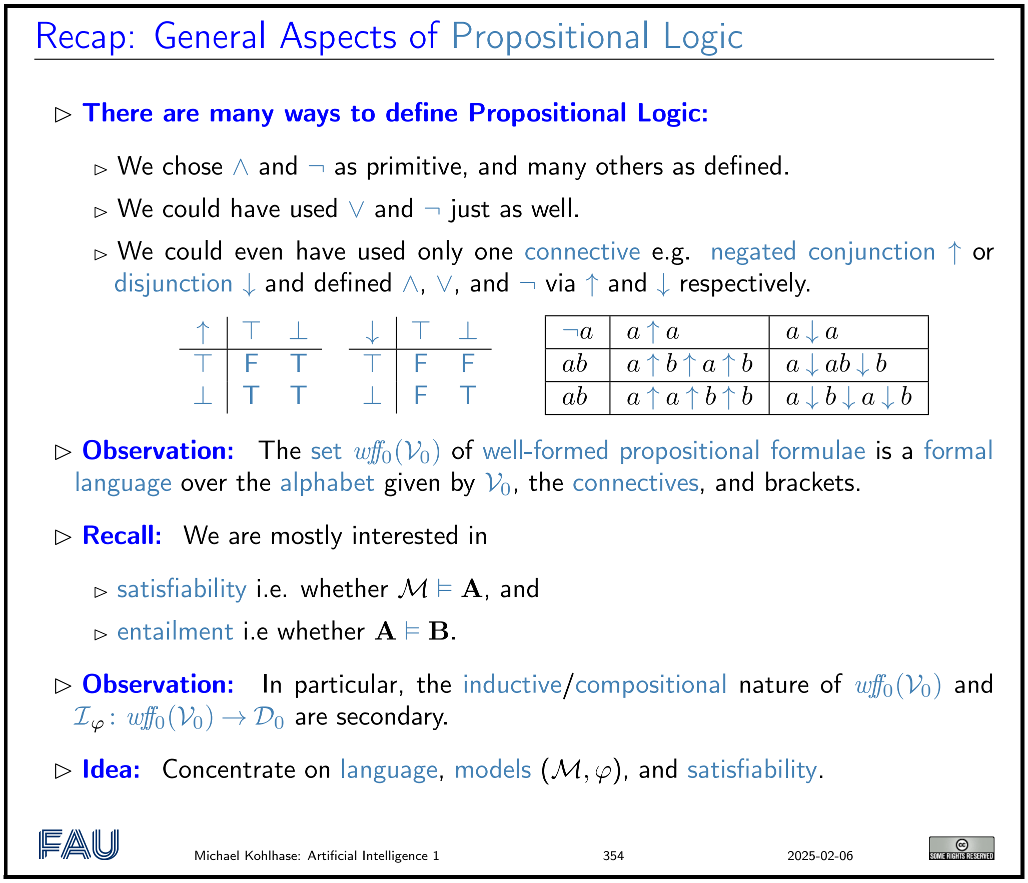
Suggested Reading:

- *Chapter 7: Logical Agents*, Sections 7.1 – 7.5 [RN09].
  - Sections 7.1 and 7.2 roughly correspond to my “Introduction”, Section 7.3 roughly corresponds to my “Logic (in AI)”, Section 7.4 roughly corresponds to my “Propositional Logic”, Section 7.5 roughly corresponds to my “Resolution” and “Killing a Wumpus”.
  - Overall, the content is quite similar. I have tried to add some additional clarifying illustrations. RN gives many complementary explanations, nice as additional background reading.
  - I would note that RN’s presentation of resolution seems a bit awkward, and Section 7.5 contains some additional material that is imho not interesting (alternate inference rules, forward and backward chaining). Horn clauses and unit resolution (also in Section 7.5), on the other hand, are quite relevant.

Chapter 11

Formal Systems: Syntax, Semantics, Entailment, and Derivation in General

We will now take a more abstract view and introduce the necessary prerequisites of abstract rule systems. We will also take the opportunity to discuss the quality criteria for calculi.



Recap: General Aspects of Propositional Logic

- ▷ **There are many ways to define Propositional Logic:**
  - ▷ We chose $\wedge$ and $\neg$ as primitive, and many others as defined.
  - ▷ We could have used $\vee$ and $\neg$ just as well.
  - ▷ We could even have used only one connective e.g. negated conjunction $\uparrow$ or disjunction $\downarrow$ and defined $\wedge$, $\vee$, and $\neg$ via $\uparrow$ and $\downarrow$ respectively.

$\uparrow$	$\top$	$\bot$
$\top$	F	T
$\bot$	T	T

$\downarrow$	$\top$	$\bot$
$\top$	F	F
$\bot$	F	T

$\neg a$	$a \uparrow a$	$a \downarrow a$
ab	$a \uparrow b \uparrow a \uparrow b$	$a \downarrow ab \downarrow b$
ab	$a \uparrow a \uparrow b \uparrow b$	$a \downarrow b \downarrow a \downarrow b$

- ▷ **Observation:** The set $\mathit{wff}_0(\mathcal{V}_0)$ of well-formed propositional formulae is a formal language over the alphabet given by $\mathcal{V}_0$, the connectives, and brackets.
- ▷ **Recall:** We are mostly interested in
  - ▷ satisfiability i.e. whether $\mathcal{M} \models \mathbf{A}$, and
  - ▷ entailment i.e whether $\mathbf{A} \models \mathbf{B}$.
- ▷ **Observation:** In particular, the inductive/compositional nature of $\mathit{wff}_0(\mathcal{V}_0)$ and $\mathcal{I}_\varphi \colon \mathit{wff}_0(\mathcal{V}_0) \to \mathcal{D}_0$ are secondary.
- ▷ **Idea:** Concentrate on language, models $(\mathcal{M}, \varphi)$, and satisfiability.

FAU Michael Kohlhase: Artificial Intelligence 1 354 2025-02-06

The notion of a logical system is at the basis of the field of logic. In its most abstract form, a logical system consists of a formal language, a class of models, and a satisfaction relation between models and expressions of the formal language. The satisfaction relation tells us when an expression is deemed true in this model.

Logical Systems

▷ **Definition 11.0.1.** A logical system (or simply a logic) is a triple $\mathcal{L}:=\langle\mathcal{L},\mathcal{K},\models\rangle$, where the language $\mathcal{L}$ is a formal language, the model class $\mathcal{K}$ is a set, and $\models\subseteq\mathcal{K}\times\mathcal{L}$. Members of $\mathcal{L}$ are called formulae of $\mathcal{L}$, members of $\mathcal{K}$ models for $\mathcal{L}$, and $\models$ the satisfaction relation.

▷ **Example 11.0.2 (Propositional Logic).** $\langle \mathit{wff}(\Sigma_{PL^0},\mathcal{V}_{PL^0}),\mathcal{K}_o,\models\rangle$ is a logical system, if we define $\mathcal{K}_o:=\mathcal{V}_0\rightharpoonup\mathcal{D}_0$ (the set of variable assignments) and $\varphi\models\mathbf{A}$ iff $\mathcal{I}_\varphi(\mathbf{A})=\mathsf{T}$.

▷ **Definition 11.0.3.** Let $\langle\mathcal{L},\mathcal{K},\models\rangle$ be a logical system, $\mathbf{M}\in\mathcal{K}$ a model and $\mathbf{A}\in\mathcal{L}$ a formula. Then we say that $\mathbf{A}$ is

- ▷ satisfied by $\mathbf{M}$ iff $\mathbf{M}\models\mathbf{A}$.
- ▷ satisfiable iff $\mathbf{A}$ is satisfied by some model.
- ▷ unsatisfiable iff $\mathbf{A}$ is not satisfiable.
- ▷ falsified by $\mathbf{M}$ iff $\mathbf{M}\not\models\mathbf{A}$.
- ▷ valid or unfalsifiable (write $\models\mathbf{A}$) iff $\mathbf{A}$ is satisfied by every model.
- ▷ invalid or falsifiable (write $\not\models\mathbf{A}$) iff $\mathbf{A}$ is not valid.

Let us now turn to the syntactical counterpart of the entailment relation: derivability in a calculus. Again, we take care to define the concepts at the general level of logical systems.
The intuition of a calculus is that it provides a set of syntactic rules that allow to reason by considering the form of propositions alone. Such rules are called inference rules, and they can be strung together to derivations — which can alternatively be viewed either as sequences of formulae where all formulae are justified by prior formulae or as trees of inference rule applications. But we can also define a calculus in the more general setting of logical systems as an arbitrary relation on formulae with some general properties. That allows us to abstract away from the homomorphic setup of logics and calculi and concentrate on the basics.

Derivation Relations and Inference Rules

▷ **Definition 11.0.4.** Let $\mathcal{L}$ be a formal language, then we call a relation $\vdash\subseteq\mathcal{P}(\mathcal{L})\times\mathcal{L}$ a derivation relation for $\mathcal{L}$, if

- ▷ $\mathcal{H}\vdash\mathbf{A}$, if $\mathbf{A}\in\mathcal{H}$ ($\vdash$ is proof reflexive),
- ▷ $\mathcal{H}\vdash\mathbf{A}$ and $(\mathcal{H}'\cup\{\mathbf{A}\})\vdash\mathbf{B}$ imply $(\mathcal{H}\cup\mathcal{H}')\vdash\mathbf{B}$ ($\vdash$ is proof transitive),
- ▷ $\mathcal{H}\vdash\mathbf{A}$ and $\mathcal{H}\subseteq\mathcal{H}'$ imply $\mathcal{H}'\vdash\mathbf{A}$ ($\vdash$ is monotonic or admits weakening).

▷ **Definition 11.0.5.** Let $\mathcal{L}$ be a formal language, then an inference rule over $\mathcal{L}$ is a decidable $n+1$ ary relation on $\mathcal{L}$. Inference rules are traditionally written as

$$\frac{\mathbf{A}_1\quad\ldots\quad\mathbf{A}_n}{\mathbf{C}}\,\mathcal{N}$$

where $\mathbf{A}_1,\ldots,\mathbf{A}_n$ and $\mathbf{C}$ are schemata for words in $\mathcal{L}$ and $\mathcal{N}$ is a name. The $\mathbf{A}_i$ are called assumptions of $\mathcal{N}$, and $\mathbf{C}$ is called its conclusion.

Any $n+1$-tuple

$$\frac{\mathbf{a}_1 \quad \ldots \quad \mathbf{a}_n}{\mathbf{c}}$$

in $\mathcal{N}$ is called an application of $\mathcal{N}$ and we say that we apply $\mathcal{N}$ to a set M of words with $\mathbf{a}_1,\ldots,\mathbf{a}_n \in M$ to obtain $\mathbf{c}$.

- ▷ **Definition 11.0.6.** An inference rule without assumptions is called an axiom.
- ▷ **Definition 11.0.7.** A calculus (or inference system) is a formal language $\mathcal{L}$ equipped with a set $\mathcal{C}$ of inference rules over $\mathcal{L}$.

With formula schemata we mean representations of sets of formulae, we use boldface uppercase letters as (meta)-variables for formulae, for instance the formula schema $\mathbf{A}\Rightarrow\mathbf{B}$ represents the set of formulae whose head is $\Rightarrow$.

Derivations

- ▷ **Definition 11.0.8.**Let $\mathcal{L} := \langle\mathcal{L},\mathcal{K},\models\rangle$ be a logical system and $\mathcal{C}$ a calculus for $\mathcal{L}$, then a $\mathcal{C}$-derivation of a formula $\mathbf{C}\in\mathcal{L}$ from a set $\mathcal{H}\subseteq\mathcal{L}$ of hypotheses (write $\mathcal{H}\vdash_{\mathcal{C}}\mathbf{C}$) is a sequence $\mathbf{A}_1,\ldots,\mathbf{A}_m$ of $\mathcal{L}$-formulae, such that
  - ▷ $\mathbf{A}_m=\mathbf{C}$, (derivation culminates in $\mathbf{C}$)
  - ▷ for all $1\le i\le m$, either $\mathbf{A}_i\in\mathcal{H}$, or (hypothesis)
  - ▷ there is an inference rule $\dfrac{\mathbf{A}_{l_1} \quad \ldots \quad \mathbf{A}_{l_k}}{\mathbf{A}_i}$ in $\mathcal{C}$ with $l_j<i$ for all $j\le k$. (rule application)

  We can also see a derivation as a derivation tree, where the $\mathbf{A}_{l_j}$ are the children of the node $\mathbf{A}_i$.
- ▷ **Example 11.0.9.**

  In the propositional Hilbert calculus $\mathcal{H}^0$ we have the derivation $P\vdash_{\mathcal{H}^0}Q\Rightarrow P$: the sequence is $P\Rightarrow Q\Rightarrow P,P,Q\Rightarrow P$ and the corresponding tree on the right.

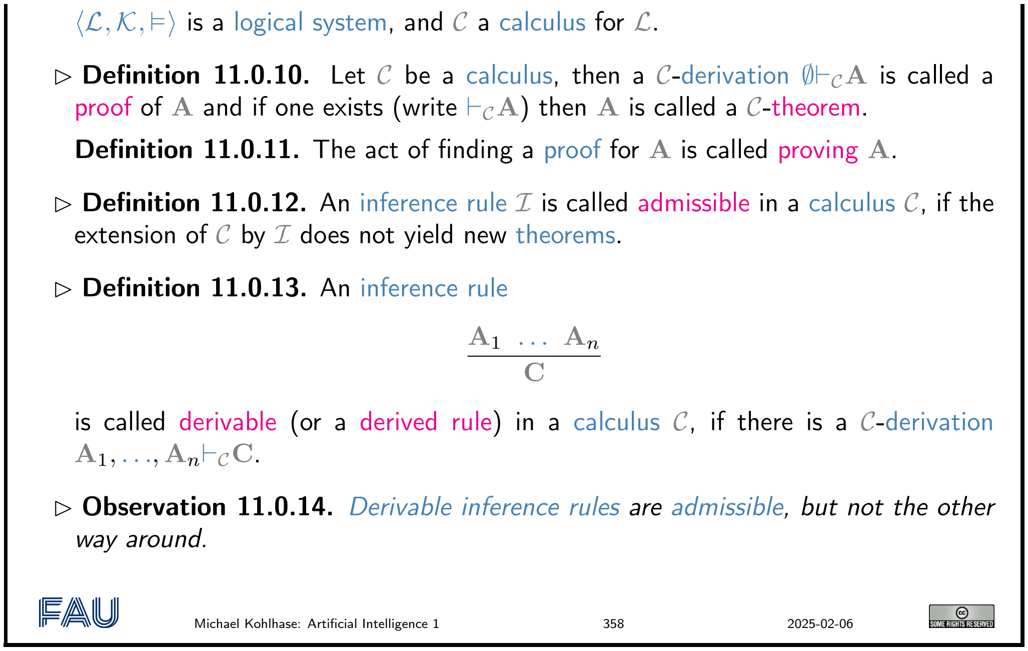
$$\dfrac{\dfrac{}{P\Rightarrow Q\Rightarrow P}K \qquad P}{Q\Rightarrow P}MP$$

Inference rules are relations on formulae represented by formula schemata (where boldface, uppercase letters are used as metavariables for formulae). For instance, in ?? the inference rule $\dfrac{\mathbf{A}\Rightarrow\mathbf{B} \quad \mathbf{A}}{\mathbf{B}}$ was applied in a situation, where the metavariables $\mathbf{A}$ and $\mathbf{B}$ were instantiated by the formulae P and $Q\Rightarrow P$.

As axioms do not have assumptions, they can be added to a derivation at any time. This is just what we did with the axioms in ??.

Formal Systems

- ▷ Let $\langle\mathcal{L},\mathcal{K},\models\rangle$ be a logical system and $\mathcal{C}$ a calculus, then $\vdash_{\mathcal{C}}$ is a derivation relation and thus $\langle\mathcal{L},\mathcal{K},\models,\vdash_{\mathcal{C}}\rangle$ a derivation system.
- ▷ Therefore we will sometimes also call $\langle\mathcal{L},\mathcal{C},\mathcal{K},\models\rangle$ a formal system, iff $\mathcal{L} :=$



$\langle\mathcal{L},\mathcal{K},\models\rangle$ is a logical system, and $\mathcal{C}$ a calculus for $\mathcal{L}$.

- ▷ **Definition 11.0.10.** Let $\mathcal{C}$ be a calculus, then a $\mathcal{C}$-derivation $\emptyset\vdash_{\mathcal{C}}\mathbf{A}$ is called a proof of $\mathbf{A}$ and if one exists (write $\vdash_{\mathcal{C}}\mathbf{A}$) then $\mathbf{A}$ is called a $\mathcal{C}$-theorem.

  Definition 11.0.11. The act of finding a proof for $\mathbf{A}$ is called proving $\mathbf{A}$.

- ▷ **Definition 11.0.12.** An inference rule $\mathcal{I}$ is called admissible in a calculus $\mathcal{C}$, if the extension of $\mathcal{C}$ by $\mathcal{I}$ does not yield new theorems.

- ▷ **Definition 11.0.13.** An inference rule

$$\frac{\mathbf{A}_1 \quad \ldots \quad \mathbf{A}_n}{\mathbf{C}}$$

  is called derivable (or a derived rule) in a calculus $\mathcal{C}$, if there is a $\mathcal{C}$-derivation $\mathbf{A}_1,\ldots,\mathbf{A}_n\vdash_{\mathcal{C}}\mathbf{C}$.

- ▷ **Observation 11.0.14.** *Derivable inference rules are admissible, but not the other way around.*

The notion of a formal system encapsulates the most general way we can conceptualize a logical system with a calculus, i.e. a system in which we can do “formal reasoning”.

Chapter 12

Machine-Oriented Calculi for Propositional Logic

A Video Nugget covering this chapter can be found at `https://fau.tv/clip/id/22531`.

12.1 Test Calculi

Automated Deduction as an Agent Inference Procedure

▷ **Recall:** Our knowledge of the cave entails a definite Wumpus position!(slide 316)

▷ **Problem:** That was human reasoning, can we build an agent function that does this?

▷ **Answer:** As for constraint networks, we use inference, here resolution/tableaux.

FAU Michael Kohlhase: Artificial Intelligence 1 359 2025-02-06

The following theorem is simple, but will be crucial later on.

Unsatisfiability Theorem

▷ **Theorem 12.1.1 (Unsatisfiability Theorem).** *$\mathcal{H} \vDash \mathbf{A}$ iff $\mathcal{H} \cup \{\neg\mathbf{A}\}$ is unsatisfiable.*

▷ *Proof:* We prove both directions separately

1. "⇒": Say $\mathcal{H} \vDash \mathbf{A}$
   1.1. For any φ with $\varphi{\models}\mathcal{H}$ we have $\varphi{\models}\mathbf{A}$ and thus $\varphi{\not\models}(\neg\mathbf{A})$.
2. "⇐": Say $\mathcal{H} \cup \{\neg\mathbf{A}\}$ is unsatisfiable.
   2.1. For any φ with $\varphi{\models}\mathcal{H}$ we have $\varphi{\not\models}(\neg\mathbf{A})$ and thus $\varphi{\models}\mathbf{A}$.

▷ **Observation 12.1.2.** *Entailment can be tested via satisfiability.*

FAU Michael Kohlhase: Artificial Intelligence 1 360 2025-02-06

Test Calculi: A Paradigm for Automating Inference

- **Definition 12.1.3.** Given a formal system $\langle\mathcal{L},\mathcal{C},\mathcal{K},\models\rangle$, the task of theorem proving consists in determining whether $\mathcal{H}\vdash_{\mathcal{C}}C$ for a conjecture $C\in\mathcal{L}$ and hypotheses $\mathcal{H}\subseteq\mathcal{L}$.
- **Definition 12.1.4.** Automated theorem proving (ATP) is the automation of theorem proving
- **Idea:** A set $\mathcal{H}$ of hypotheses and a conjecture $\mathbf{A}$ induce a search problem $\Pi_{\mathcal{C}}^{\mathcal{H}\models\mathbf{A}}:=\langle\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{I},\mathcal{G}\rangle$, where the states $\mathcal{S}$ are sets of formulae, the actions $\mathcal{A}$ are the inference rules from $\mathcal{C}$, the initial state $\mathcal{I}=\mathcal{H}$, and the goal states are those with $\mathbf{A}\in\mathcal{S}$.
- **Problem:** ATP as a search problem does not admit good heuristics, since these need to take the conjecture $\mathcal{A}$ into account.
- **Idea:** Turn the search around – using the unsatisfiability theorem (**??**).
- **Definition 12.1.5.** For a given conjecture A and hypotheses $\mathcal{H}$ a test calculus $\mathcal{T}$ tries to derive a refutation $\mathcal{H},\overline{A}\vdash_{\mathcal{T}}\bot$ instead of $\mathcal{H}\vdash A$, where $\overline{A}$ is unsatisfiable iff A is valid and $\bot$, an "obviously" unsatisfiable formula.
- **Observation:** A test calculus $\mathcal{C}$ induces a search problem where the initial state is $\mathcal{H}\cup\{\neg\mathbf{A}\}$ and $S\in\mathcal{S}$ is a goal state iff $\bot\in S$.(proximity of $\bot$ easier for heuristics)
- Searching for $\bot$ admits simple heuristics, e.g. size reduction. ($\bot$ minimal)

Michael Kohlhase: Artificial Intelligence 1 361 2025-02-06

12.1.1 Normal Forms

Before we can start, we will need to recap some nomenclature on formulae.

Recap: Atoms and Literals

- **Definition 12.1.6.** A formula is called atomic (or an atom) if it does not contain logical constants, else it is called complex.
- **Definition 12.1.7.** Let $\langle\mathcal{L},\mathcal{K},\models\rangle$ be a logical system and $\mathbf{A}\in\mathcal{L}$, then we call a pair $\mathbf{A}^\alpha$ of a formula and a truth value $\alpha\in\{\mathsf{T},\mathsf{F}\}$ a labeled formula. For a set Φ of formulae we use $\Phi^\alpha:=\{\mathbf{A}^\alpha\,|\,\mathbf{A}\in\Phi\}$.

  We call a labeled formula $\mathbf{A}^\mathsf{T}$ positive and $\mathbf{A}^\mathsf{F}$ negative.

  Definition 12.1.8. Let $\langle\mathcal{L},\mathcal{K},\models\rangle$ be a logical system and $\mathbf{A}^\alpha$ a labeled formula. Then we say that $\mathcal{M}\in\mathcal{K}$ satisfies $\mathbf{A}^\alpha$ (written $\mathcal{M}\models\mathbf{A}$), iff $\alpha=\mathsf{T}$ and $\mathcal{M}\models\mathbf{A}$ or $\alpha=\mathsf{F}$ and $\mathcal{M}\not\models\mathbf{A}$.
- **Definition 12.1.9.** Let $\langle\mathcal{L},\mathcal{K},\models\rangle$ be a logical system, $\mathbf{A}\in\mathcal{L}$ atomic, and $\alpha\in\{\mathsf{T},\mathsf{F}\}$, then we call a $\mathbf{A}^\alpha$ a literal.
- **Intuition:** To satisfy a formula, we make it "true". To satisfy a labeled formula $\mathbf{A}^\alpha$, it must have the truth value α.

▷ **Definition 12.1.10.** For a literal $\mathbf{A}^{\alpha}$, we call the literal $\mathbf{A}^{\beta}$ with $\alpha \neq \beta$ the opposite literal (or partner literal).

The idea about literals is that they are atoms (the simplest formulae) that carry around their intended truth value.

Alternative Definition: Literals

▷ **Note:** Literals are often defined without recurring to labeled formulae:

▷ **Definition 12.1.11.** A literal is an atom $\mathbf{A}$ (positive literal) or negated atom $\neg\mathbf{A}$ (negative literal). $\mathbf{A}$ and $\neg\mathbf{A}$ are opposite literals.

▷ **Note:** This notion of literal is equivalent to the labeled formulae-notion of literal, but does not generalize as well to logics with more than two truth values.

Normal Forms

▷ There are two quintessential normal forms for propositional formulae: (there are others as well)

▷ **Definition 12.1.12.** A formula is in conjunctive normal form (CNF) if it is T or a conjunction of disjunctions of literals: i.e. if it is of the form $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} l_{ij}$

▷ **Definition 12.1.13.** A formula is in disjunctive normal form (DNF) if it is F or a disjunction of conjunctions of literals: i.e. if it is of the form $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} l_{ij}$

▷ **Observation 12.1.14.** *Every formula has equivalent formulae in CNF and DNF.*

Video Nuggets covering this chapter can be found at `https://fau.tv/clip/id/23705` and `https://fau.tv/clip/id/23708`.

12.2 Analytical Tableaux

Video Nuggets covering this section can be found at `https://fau.tv/clip/id/23705` and `https://fau.tv/clip/id/23708`.

12.2.1 Analytical Tableaux

Test Calculi: Tableaux and Model Generation

▷ **Idea:** A tableau calculus is a test calculus that

- ▷ analyzes a labeled formulae in a tree to determine satisfiability,
- ▷ its branches correspond to valuations (⇝ models).

▷ **Example 12.2.1.** Tableau calculi try to construct models for labeled formulae:

Tableau refutation (Validity)	Model generation (Satisfiability)
$\models P\wedge Q\Rightarrow Q\wedge P$	$\models P\wedge(Q\vee\neg R)\wedge\neg Q$
$(P\wedge Q\Rightarrow Q\wedge P)^{\mathsf{F}}$ $(P\wedge Q)^{\mathsf{T}}$ $(Q\wedge P)^{\mathsf{F}}$ P^{T} Q^{T} $\begin{array}{c\|c} P^{\mathsf{F}} & Q^{\mathsf{F}} \\ \bot & \bot \end{array}$	$(P\wedge(Q\vee\neg R)\wedge\neg Q)^{\mathsf{T}}$ $(P\wedge(Q\vee\neg R))^{\mathsf{T}}$ $\neg Q^{\mathsf{T}}$ Q^{F} P^{T} $(Q\vee\neg R)^{\mathsf{T}}$ $\begin{array}{c\|c} Q^{\mathsf{T}} & \neg R^{\mathsf{T}} \\ \bot & R^{\mathsf{F}} \end{array}$
No Model	Herbrand model $\{P^{\mathsf{T}}, Q^{\mathsf{F}}, R^{\mathsf{F}}\}$ $\varphi := \{P\mapsto \mathsf{T}, Q\mapsto \mathsf{F}, R\mapsto \mathsf{F}\}$

▷ **Idea:** Open branches in saturated tableaux yield models.

▷ **Algorithm:** Fully expand all possible tableaux, (no rule can be applied)

- ▹ Satisfiable, iff there are open branches (correspond to models)

Tableau calculi develop a formula in a tree-shaped arrangement that represents a case analysis on when a formula can be made true (or false). Therefore the formulae are decorated with upper indices that hold the intended truth value.

On the left we have a refutation tableau that analyzes a negated formula (it is decorated with the intended truth value F). Both branches contain an elementary contradiction ⊥.

On the right we have a model generation tableau, which analyzes a positive formula (it is decorated with the intended truth value T). This tableau uses the same rules as the refutation tableau, but makes a case analysis of when this formula can be satisfied. In this case we have a closed branch and an open one. The latter corresponds a model.

Now that we have seen the examples, we can write down the tableau rules formally.

Analytical Tableaux (Formal Treatment of $\mathcal{T}_0$)

▷ **Idea:** A test calculus where

- ▹ A labeled formula is analyzed in a tree to determine satisfiability,
- ▹ branches correspond to valuations (models)

▷ **Definition 12.2.2.** The propositional tableau calculus $\mathcal{T}_0$ has two inference rules per connective (one for each possible label)

$$\frac{(\mathbf{A}\wedge\mathbf{B})^{\mathsf{T}}}{\begin{array}{c}\mathbf{A}^{\mathsf{T}}\\ \mathbf{B}^{\mathsf{T}}\end{array}}\mathcal{T}_0\wedge \qquad \frac{(\mathbf{A}\wedge\mathbf{B})^{\mathsf{F}}}{\mathbf{A}^{\mathsf{F}}\;\big|\;\mathbf{B}^{\mathsf{F}}}\mathcal{T}_0\vee \qquad \frac{\neg\mathbf{A}^{\mathsf{T}}}{\mathbf{A}^{\mathsf{F}}}\mathcal{T}_0\neg^{\mathsf{T}} \qquad \frac{\neg\mathbf{A}^{\mathsf{F}}}{\mathbf{A}^{\mathsf{T}}}\mathcal{T}_0\neg^{\mathsf{F}} \qquad \frac{\begin{array}{c}\mathbf{A}^{\alpha}\\ \mathbf{A}^{\beta}\end{array}\quad \alpha\neq\beta}{\bot}\mathcal{T}_0\bot$$

Use rules exhaustively as long as they contribute new material (⇝ termination)

▷ **Definition 12.2.3.** We call any tree ($|$ introduces branches) produced by the $\mathcal{T}_0$ inference rules from a set Φ of labeled formulae a tableau for Φ.

▷ **Definition 12.2.4.** Call a tableau saturated, iff no rule adds new material and a branch closed, iff it ends in $\bot$, else open. A tableau is closed, iff all of its branches are.

In analogy to the $\bot$ at the end of closed branches, we sometimes decorate open branches with a □ symbol.

These inference rules act on tableaux have to be read as follows: if the formulae over the line appear in a tableau branch, then the branch can be extended by the formulae or branches below the line. There are two rules for each primary connective, and a branch closing rule that adds the special symbol $\bot$ (for unsatisfiability) to a branch.

We use the tableau rules with the convention that they are only applied, if they contribute new material to the branch. This ensures termination of the tableau procedure for propositional logic (every rule eliminates one primary connective).

Definition 12.2.5. We will call a closed tableau with the labeled formula $\mathbf{A}^\alpha$ at the root a tableau refutation for $\mathcal{A}^\alpha$.

The saturated tableau represents a full case analysis of what is necessary to give $\mathbf{A}$ the truth value α; since all branches are closed (contain contradictions) this is impossible.

Analytical Tableaux ($\mathcal{T}_0$ continued)

▷ **Definition 12.2.6 ($\mathcal{T}_0$-Theorem/Derivability).** $\mathbf{A}$ is a $\mathcal{T}_0$-theorem ($\vdash_{\mathcal{T}_0}\mathbf{A}$), iff there is a closed tableau with $\mathbf{A}^{\mathsf{F}}$ at the root.

$\Phi \subseteq wff_0(\mathcal{V}_0)$ derives $\mathbf{A}$ in $\mathcal{T}_0$ ($\Phi\vdash_{\mathcal{T}_0}\mathbf{A}$), iff there is a closed tableau starting with $\mathbf{A}^{\mathsf{F}}$ and Φ^{T}. The tableau with only a branch of $\mathbf{A}^{\mathsf{F}}$ and Φ^{T} is called initial for $\Phi\vdash_{\mathcal{T}_0}\mathbf{A}$.

Definition 12.2.7. We will call a tableau refutation for $\mathbf{A}^{\mathsf{F}}$ a tableau proof for $\mathbf{A}$, since it refutes the possibility of finding a model where $\mathbf{A}$ evaluates to F. Thus $\mathbf{A}$ must evaluate to T in all models, which is just our definition of validity.

Thus the tableau procedure can be used as a calculus for propositional logic. In contrast to the propositional Hilbert calculus it does not prove a theorem $\mathbf{A}$ by deriving it from a set of axioms, but it proves it by refuting its negation. Such calculi are called negative or test calculi. Generally negative calculi have computational advantages over positive ones, since they have a built-in sense of direction.

We have rules for all the necessary connectives (we restrict ourselves to $\wedge$ and $\neg$, since the others can be expressed in terms of these two via the propositional identities above. For instance, we can write $\mathbf{A} \vee \mathbf{B}$ as $\neg(\neg\mathbf{A} \wedge \neg\mathbf{B})$, and $\mathbf{A} \Rightarrow \mathbf{B}$ as $\neg\mathbf{A} \vee \mathbf{B}, \ldots$.)

We now look at a formulation of propositional logic with fancy variable names. Note that $\mathrm{loves}(\mathrm{mary}, \mathrm{bill})$ is just a variable name like P or X, which we have used earlier.

A Valid Real-World Example

▷ **Example 12.2.8.** *If Mary loves Bill and John loves Mary, then John loves Mary*

$$\begin{array}{c}
(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \land \mathrm{loves}(\mathrm{john},\mathrm{mary}) \Rightarrow \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}} \\
\neg(\neg\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \land \mathrm{loves}(\mathrm{john},\mathrm{mary})) \land \neg\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}} \\
(\neg\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \land \mathrm{loves}(\mathrm{john},\mathrm{mary})) \land \neg\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}} \\
\neg\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \land \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}} \\
\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \land \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}} \\
(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \land \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}} \\
\neg\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}} \\
\mathrm{loves}(\mathrm{mary},\mathrm{bill})^{\mathsf{T}} \\
\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}} \\
\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{F}} \\
\bot
\end{array}$$

This is a closed tableau, so the $\mathrm{loves}(\mathrm{mary},\mathrm{bill})\land\mathrm{loves}(\mathrm{john},\mathrm{mary})\Rightarrow\mathrm{loves}(\mathrm{john},\mathrm{mary})$ is a $\mathcal{T}_0$-theorem.

As we will see, $\mathcal{T}_0$ is sound and complete, so

$$\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \land \mathrm{loves}(\mathrm{john},\mathrm{mary}) \Rightarrow \mathrm{loves}(\mathrm{john},\mathrm{mary})$$

is valid.

We could have used the unsatisfiability theorem (**??**) here to show that *If Mary loves Bill and John loves Mary* entails *John loves Mary*. But there is a better way to show entailment: we directly use derivability in $\mathcal{T}_0$.

Deriving Entailment in $\mathcal{T}_0$

▷ **Example 12.2.9.** *Mary loves Bill* and *John loves Mary* together entail that *John loves Mary*

$$\begin{array}{c}
\mathrm{loves}(\mathrm{mary},\mathrm{bill})^{\mathsf{T}} \\
\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}} \\
\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{F}} \\
\bot
\end{array}$$

This is a closed tableau, so $\{\mathrm{loves}(\mathrm{mary},\mathrm{bill}),\mathrm{loves}(\mathrm{john},\mathrm{mary})\}\vdash_{\mathcal{T}_0}\mathrm{loves}(\mathrm{john},\mathrm{mary})$.

Again, as $\mathcal{T}_0$ is sound and complete we have

$$\{\mathrm{loves}(\mathrm{mary},\mathrm{bill}),\mathrm{loves}(\mathrm{john},\mathrm{mary})\} \models \mathrm{loves}(\mathrm{john},\mathrm{mary})$$

Note: We can also use the tableau calculus to try and show entailment (and fail). The nice thing is that the failed proof, we can see what went wrong.

A Falsifiable Real-World Example

▷ **Example 12.2.10.** * *If Mary loves Bill or John loves Mary, then John loves Mary*

Try proving the implication (this fails)

$$\begin{array}{c}
((\mathrm{loves}(\mathrm{mary},\mathrm{bill})\lor\mathrm{loves}(\mathrm{john},\mathrm{mary}))\Rightarrow\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}}\\
\neg(\neg\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill})\lor\mathrm{loves}(\mathrm{john},\mathrm{mary}))\land\neg\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}}\\
(\neg\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill})\lor\mathrm{loves}(\mathrm{john},\mathrm{mary}))\land\neg\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}}\\
\neg\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}}\\
\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{F}}\\
\neg\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill})\lor\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}}\\
\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill})\lor\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}}\\
(\mathrm{loves}(\mathrm{mary},\mathrm{bill})\lor\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}}\\
\begin{array}{c|c}
\mathrm{loves}(\mathrm{mary},\mathrm{bill})^{\mathsf{T}} & \mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}}\\
 & \bot
\end{array}
\end{array}$$

Indeed we can make $\mathcal{I}_\varphi(\mathrm{loves}(\mathrm{mary},\mathrm{bill}))=\mathsf{T}$ but $\mathcal{I}_\varphi(\mathrm{loves}(\mathrm{john},\mathrm{mary}))=\mathsf{F}$.

Michael Kohlhase: Artificial Intelligence 1 370 2025-02-06

Obviously, the tableau above is saturated, but not closed, so it is not a tableau proof for our initial entailment conjecture. We have marked the literal on the open branch green, since they allow us to read of the conditions of the situation, in which the entailment fails to hold. As we intuitively argued above, this is the situation, where *Mary loves Bill*. In particular, the open branch gives us a variable assignment (marked in green) that satisfies the initial formula. In this case, *Mary loves Bill*, which is a situation, where the entailment fails.

Again, the derivability version is much simpler:

Testing for Entailment in $\mathcal{T}_0$

▷ **Example 12.2.11.** Does *Mary loves Bill or John loves Mary* entail that *John loves Mary*?

$$\begin{array}{c}
(\mathrm{loves}(\mathrm{mary},\mathrm{bill})\lor\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}}\\
\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{F}}\\
\begin{array}{c|c}
\mathrm{loves}(\mathrm{mary},\mathrm{bill})^{\mathsf{T}} & \mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}}\\
 & \bot
\end{array}
\end{array}$$

This saturated tableau has an open branch that shows that the interpretation with $\mathcal{I}_\varphi(\mathrm{loves}(\mathrm{mary},\mathrm{bill}))=\mathsf{T}$ but $\mathcal{I}_\varphi(\mathrm{loves}(\mathrm{john},\mathrm{mary}))=\mathsf{F}$ falsifies the derivability/entailment conjecture.

Michael Kohlhase: Artificial Intelligence 1 371 2025-02-06

We have seen in the examples above that while it is possible to get by with only the connectives ∨ and ¬, it is a bit unnatural and tedious, since we need to eliminate the other connectives first. In this section, we will make the calculus less frugal by adding rules for the other connectives, without losing the advantage of dealing with a small calculus, which is good making statements about the calculus itself.

12.2.2 Practical Enhancements for Tableaux

The main idea here is to add the new rules as derivable inference rules, i.e. rules that only abbreviate derivations in the original calculus. Generally, adding derivable inference rules does not change the derivation relation of the calculus, and is therefore a safe thing to do. In particular, we will add the following rules to our tableau calculus.

We will convince ourselves that the first rule is derivable, and leave the other ones as an exercise.

Derived Rules of Inference

- ▷ **Definition 12.2.12.** An inference rule

$$\frac{\mathbf{A}_1 \quad \ldots \quad \mathbf{A}_n}{\mathbf{C}}$$

is called derivable (or a derived rule) in a calculus $\mathcal{C}$, if there is a $\mathcal{C}$-derivation $\mathbf{A}_1, \ldots, \mathbf{A}_n \vdash_{\mathcal{C}} \mathbf{C}$.

- ▷ **Definition 12.2.13.** We have the following derivable inference rules in $\mathcal{T}_0$:

$$\frac{(\mathbf{A} \Rightarrow \mathbf{B})^{\mathsf{T}}}{\mathbf{A}^{\mathsf{F}} \mid \mathbf{B}^{\mathsf{T}}} \qquad \frac{(\mathbf{A} \Rightarrow \mathbf{B})^{\mathsf{F}}}{\begin{array}{c}\mathbf{A}^{\mathsf{T}}\\ \mathbf{B}^{\mathsf{F}}\end{array}} \qquad \frac{\begin{array}{c}\mathbf{A}^{\mathsf{T}}\\ (\mathbf{A} \Rightarrow \mathbf{B})^{\mathsf{T}}\end{array}}{\mathbf{B}^{\mathsf{T}}}$$

$$\frac{(\mathbf{A} \vee \mathbf{B})^{\mathsf{T}}}{\mathbf{A}^{\mathsf{T}} \mid \mathbf{B}^{\mathsf{T}}} \qquad \frac{(\mathbf{A} \vee \mathbf{B})^{\mathsf{F}}}{\begin{array}{c}\mathbf{A}^{\mathsf{F}}\\ \mathbf{B}^{\mathsf{F}}\end{array}} \qquad \frac{(\mathbf{A} \Leftrightarrow \mathbf{B})^{\mathsf{T}}}{\begin{array}{c|c}\mathbf{A}^{\mathsf{T}} & \mathbf{A}^{\mathsf{F}}\\ \mathbf{B}^{\mathsf{T}} & \mathbf{B}^{\mathsf{F}}\end{array}} \qquad \frac{(\mathbf{A} \Leftrightarrow \mathbf{B})^{\mathsf{F}}}{\begin{array}{c|c}\mathbf{A}^{\mathsf{T}} & \mathbf{A}^{\mathsf{F}}\\ \mathbf{B}^{\mathsf{F}} & \mathbf{B}^{\mathsf{T}}\end{array}}$$

$$\begin{array}{c}
\mathbf{A}^{\mathsf{T}}\\
(\mathbf{A} \Rightarrow \mathbf{B})^{\mathsf{T}}\\
(\neg\mathbf{A} \vee \mathbf{B})^{\mathsf{T}}\\
\neg(\neg\neg\mathbf{A} \wedge \neg\mathbf{B})^{\mathsf{T}}\\
(\neg\neg\mathbf{A} \wedge \neg\mathbf{B})^{\mathsf{F}}\\
\begin{array}{c|c}
\neg\neg\mathbf{A}^{\mathsf{F}} & \neg\mathbf{B}^{\mathsf{F}}\\
\neg\mathbf{A}^{\mathsf{T}} & \mathbf{B}^{\mathsf{T}}\\
\mathbf{A}^{\mathsf{F}} & \\
\bot &
\end{array}
\end{array}$$

With these derived rules, theorem proving becomes quite efficient. With these rules, the tableau (??) would have the following simpler form:

Tableaux with derived Rules (example)

Example 12.2.14.

$$\begin{array}{c}
(\mathrm{loves}(\mathrm{mary}, \mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john}, \mathrm{mary}) \Rightarrow \mathrm{loves}(\mathrm{john}, \mathrm{mary}))^{\mathsf{F}}\\
(\mathrm{loves}(\mathrm{mary}, \mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john}, \mathrm{mary}))^{\mathsf{T}}\\
\mathrm{loves}(\mathrm{john}, \mathrm{mary})^{\mathsf{F}}\\
\mathrm{loves}(\mathrm{mary}, \mathrm{bill})^{\mathsf{T}}\\
\mathrm{loves}(\mathrm{john}, \mathrm{mary})^{\mathsf{T}}\\
\bot
\end{array}$$

12.2.3 Soundness and Termination of Tableaux

As always we need to convince ourselves that the calculus is sound, otherwise, tableau proofs do not guarantee validity, which we are after. Since we are now in a refutation setting we cannot just show that the inference rules preserve validity: we care about unsatisfiability (which is the dual notion to validity), as we want to show the initial labeled formula to be unsatisfiable. Before we can do this, we have to ask ourselves, what it means to be (un)-satisfiable for a labeled formula or a tableau.

Soundness (Tableau)

- ▷ **Idea:** A test calculus is refutation sound, iff its inference rules preserve satisfiability and the goal formulae are unsatisfiable.
- ▷ **Definition 12.2.15.** A labeled formula $\mathbf{A}^\alpha$ is valid under φ, iff $\mathcal{I}_\varphi(\mathbf{A}) = \alpha$.
- ▷ **Definition 12.2.16.** A tableau $\mathcal{T}$ is satisfiable, iff there is a satisfiable branch $\mathcal{P}$ in $\mathcal{T}$, i.e. if the set of formulae on $\mathcal{P}$ is satisfiable.
- ▷ **Lemma 12.2.17.** *$\mathcal{T}_0$ rules transform satisfiable tableaux into satisfiable ones.*
- ▷ **Theorem 12.2.18 (Soundness).** *$\mathcal{T}_0$ is sound, i.e. $\Phi \subseteq wf\!f_0(\mathcal{V}_0)$ valid, if there is a closed tableau $\mathcal{T}$ for Φ^F.*
- ▷ *Proof:* by contradiction
  1. Suppose Φ isfalsifiable $\widehat{=}$ not valid.
  2. Then the initial tableau is satisfiable, (Φ^F satisfiable)
  3. so $\mathcal{T}$ is satisfiable, by **??**.
  4. Thus there is a satisfiable branch (by definition)
  5. but all branches are closed ($\mathcal{T}$ closed)
- ▷ **Theorem 12.2.19 (Completeness).** *$\mathcal{T}_0$ is complete, i.e. if $\Phi \subseteq wf\!f_0(\mathcal{V}_0)$ is valid, then there is a closed tableau $\mathcal{T}$ for Φ^F.*

  Proof sketch: Proof difficult/interesting; see **??**

Thus we only have to prove **??**, this is relatively easy to do. For instance for the first rule: if we have a tableau that contains $(\mathbf{A} \wedge \mathbf{B})^\mathsf{T}$ and is satisfiable, then it must have a satisfiable branch. If $(\mathbf{A} \wedge \mathbf{B})^\mathsf{T}$ is not on this branch, the tableau extension will not change satisfiability, so we can assume that it is on the satisfiable branch and thus $\mathcal{I}_\varphi(\mathbf{A} \wedge \mathbf{B}) = \mathsf{T}$ for some variable assignment φ. Thus $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$ and $\mathcal{I}_\varphi(\mathbf{B}) = \mathsf{T}$, so after the extension (which adds the formulae $\mathbf{A}^\mathsf{T}$ and $\mathbf{B}^\mathsf{T}$ to the branch), the branch is still satisfiable. The cases for the other rules are similar.
The next result is a very important one, it shows that there is a procedure (the tableau procedure) that will always terminate and answer the question whether a given propositional formula is valid or not. This is very important, since other logics (like the often-studied first-order logic) does not enjoy this property.

▷ Termination for Tableaux

- ▷ **Lemma 12.2.20.** *$\mathcal{T}_0$ terminates, i.e. every $\mathcal{T}_0$ tableau becomes saturated after finitely many rule applications.*
- ▷ *Proof:* By examining the rules wrt. a measure μ
  1. Let us call a labeled formulae $\mathbf{A}^\alpha$ worked off in a tableau $\mathcal{T}$, if a $\mathcal{T}_0$ rule has already been applied to it.
  2. It is easy to see that applying rules to worked off formulae will only add formulae that are already present in its branch.
  3. Let $\mu(\mathcal{T})$ be the number of connectives in labeled formulae in $\mathcal{T}$ that are not worked off.
  4. Then each rule application to a labeled formula in $\mathcal{T}$ that is not worked off reduces $\mu(\mathcal{T})$ by at least one. (inspect the rules)
  5. At some point the tableau only contains worked off formulae and literals.
  6. Since there are only finitely many literals in $\mathcal{T}$, so we can only apply $\mathcal{T}_0\bot$ a finite number of times.

▷ **Corollary 12.2.21.** *$\mathcal{T}_0$ induces a decision procedure for validity in PL^0.*

Proof: We combine the results so far

▷ 1. By **??** it is decidable whether $\vdash_{\mathcal{T}_0}\mathbf{A}$
2. By soundness (**??**) and completeness (**??**), $\vdash_{\mathcal{T}_0}\mathbf{A}$ iff $\mathbf{A}$ is valid.

Michael Kohlhase: Artificial Intelligence 1 375 2025-02-06

Note: The proof above only works for the "base $\mathcal{T}_0$" because (only) there the rules do not "copy". A rule like

$$\frac{(\mathbf{A}\Leftrightarrow\mathbf{B})^{\mathsf{T}}}{\begin{array}{c|c}\mathbf{A}^{\mathsf{T}} & \mathbf{A}^{\mathsf{F}}\\ \mathbf{B}^{\mathsf{T}} & \mathbf{B}^{\mathsf{F}}\end{array}}$$

does, and in particular the number of non-worked-off variables below the line is larger than above the line. For such rules, we would have a more intricate version of μ which – instead of returning a natural number – returns a more complex object; a multiset of numbers. would work here. In our proof we are just assuming that the defined connectives have already eliminated. The tableau calculus basically computes the disjunctive normal form: every branch is a disjunct that is a conjunction of literals. The method relies on the fact that a DNF is unsatisfiable, iff each literal is, i.e. iff each branch contains a contradiction in form of a pair of opposite literals.

12.3 Resolution for Propositional Logic

12.3.1 Resolution for Propositional Logic

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/23712`. The next calculus is a test calculus based on the conjunctive normal form: the resolution calculus. In contrast to the tableau method, it does not compute the normal form as it goes along, but has a pre-processing step that does this and a single inference rule that maintains the normal form. The goal of this calculus is to derive the empty clause, which is unsatisfiable.

Another Test Calculus: Resolution

▷ **Definition 12.3.1.** A clause is a disjunction $l_1^{\alpha_1}\vee\ldots\vee l_n^{\alpha_n}$ of literals. We will use $\Box$ for the "empty" disjunction (no disjuncts) and call it the empty clause. A clause with exactly one literal is called a unit clause.

▷ **Definition 12.3.2 (Resolution Calculus).** The resolution calculus $\mathcal{R}_0$ operates a clause sets via a single inference rule:

$$\frac{P^{\mathsf{T}}\vee\mathbf{A}\quad P^{\mathsf{F}}\vee\mathbf{B}}{\mathbf{A}\vee\mathbf{B}}\,\mathcal{R}$$

This rule allows to add the resolvent (the clause below the line) to a clause set which contains the two clauses above. The literals P^{T} and P^{F} are called cut literals.

▷ **Definition 12.3.3 (Resolution Refutation).** Let S be a clause set, then we call an $\mathcal{R}_0$-derivation of $\Box$ from S $\mathcal{R}_0$-refutation and write $\mathcal{D}\colon S\vdash_{\mathcal{R}_0}\Box$.

Michael Kohlhase: Artificial Intelligence 1 376 2025-02-06

Clause Normal Form Transformation (A calculus)

- ▷ **Definition 12.3.4.** We will often write a clause set $\{C_1, \ldots, C_n\}$ as $C_1 ; \ldots ; C_n$, use $S ; T$ for the union of the clause sets S and T, and $S ; C$ for the extension by a clause C.
- ▷ **Definition 12.3.5 (Transformation into Clause Normal Form).** The CNF transformation calculus CNF_0 consists of the following four inference rules on sets of labeled formulae.

$$\frac{\mathbf{C} \vee (\mathbf{A} \vee \mathbf{B})^{\mathsf{T}}}{\mathbf{C} \vee \mathbf{A}^{\mathsf{T}} \vee \mathbf{B}^{\mathsf{T}}} \qquad \frac{\mathbf{C} \vee (\mathbf{A} \vee \mathbf{B})^{\mathsf{F}}}{\mathbf{C} \vee \mathbf{A}^{\mathsf{F}} ; \mathbf{C} \vee \mathbf{B}^{\mathsf{F}}} \qquad \frac{\mathbf{C} \vee \neg \mathbf{A}^{\mathsf{T}}}{\mathbf{C} \vee \mathbf{A}^{\mathsf{F}}} \qquad \frac{\mathbf{C} \vee \neg \mathbf{A}^{\mathsf{F}}}{\mathbf{C} \vee \mathbf{A}^{\mathsf{T}}}$$

- ▷ **Definition 12.3.6.** We write $\mathit{CNF}_0(\mathbf{A}^\alpha)$ for the set of all clauses derivable from $\mathbf{A}^\alpha$ via the rules above.

that the **C**-terms in the definition of the inference rules are necessary, since we assumed that the assumptions of the inference rule must match full clauses. The **C** terms are used with the convention that they are optional. So that we can also simplify $(\mathbf{A} \vee \mathbf{B})^{\mathsf{T}}$ to $\mathbf{A}^{\mathsf{T}} \vee \mathbf{B}^{\mathsf{T}}$.

Background: The background behind this notation is that $\mathbf{A}$ and $T \vee \mathbf{A}$ are equivalent for any $\mathbf{A}$. That allows us to interpret the **C**-terms in the assumptions as T and thus leave them out.

The clause normal form translation as we have formulated it here is quite frugal; we have left out rules for the connectives $\vee$, $\Rightarrow$, and $\Leftrightarrow$, relying on the fact that formulae containing these connectives can be translated into ones without before CNF transformation. The advantage of having a calculus with few inference rules is that we can prove meta properties like soundness and completeness with less effort (these proofs usually require one case per inference rule). On the other hand, adding specialized inference rules makes proofs shorter and more readable.

Fortunately, there is a way to have your cake and eat it. Derivable inference rules have the property that they are formally redundant, since they do not change the expressive power of the calculus. Therefore we can leave them out when proving meta-properties, but include them when actually using the calculus.

Derived Rules of Inference

- ▷ **Definition 12.3.7.** An inference rule

$$\frac{\mathbf{A}_1 \quad \ldots \quad \mathbf{A}_n}{\mathbf{C}}$$

is called derivable (or a derived rule) in a calculus $\mathcal{C}$, if there is a $\mathcal{C}$-derivation $\mathbf{A}_1, \ldots, \mathbf{A}_n \vdash_{\mathcal{C}} \mathbf{C}$.

- ▷ **Idea:** Derived rules make derivations shorter.
- ▷ **Example 12.3.8.**

$$\cfrac{\cfrac{\cfrac{\mathbf{C} \vee (\mathbf{A} \Rightarrow \mathbf{B})^{\mathsf{T}}}{\mathbf{C} \vee (\neg \mathbf{A} \vee \mathbf{B})^{\mathsf{T}}}}{\mathbf{C} \vee \neg \mathbf{A}^{\mathsf{T}} \vee \mathbf{B}^{\mathsf{T}}}}{\mathbf{C} \vee \mathbf{A}^{\mathsf{F}} \vee \mathbf{B}^{\mathsf{T}}} \quad \rightsquigarrow \quad \frac{\mathbf{C} \vee (\mathbf{A} \Rightarrow \mathbf{B})^{\mathsf{T}}}{\mathbf{C} \vee \mathbf{A}^{\mathsf{F}} \vee \mathbf{B}^{\mathsf{T}}}$$

▷ **Other Derived CNF Rules:**

$$\frac{\mathbf{C}\lor(\mathbf{A}\Rightarrow\mathbf{B})^{\mathsf{T}}}{\mathbf{C}\lor\mathbf{A}^{\mathsf{F}}\lor\mathbf{B}^{\mathsf{T}}} \qquad \frac{\mathbf{C}\lor(\mathbf{A}\Rightarrow\mathbf{B})^{\mathsf{F}}}{\mathbf{C}\lor\mathbf{A}^{\mathsf{T}}\,;\,\mathbf{C}\lor\mathbf{B}^{\mathsf{F}}} \qquad\qquad \frac{\mathbf{C}\lor(\mathbf{A}\land\mathbf{B})^{\mathsf{T}}}{\mathbf{C}\lor\mathbf{A}^{\mathsf{T}}\,;\,\mathbf{C}\lor\mathbf{B}^{\mathsf{T}}} \qquad \frac{\mathbf{C}\lor(\mathbf{A}\land\mathbf{B})^{\mathsf{F}}}{\mathbf{C}\lor\mathbf{A}^{\mathsf{F}}\lor\mathbf{B}^{\mathsf{F}}}$$

With these derivable rules, theorem proving becomes quite efficient. To get a better understanding of the calculus, we look at an example: we prove an axiom of the Hilbert Calculus we have studied above.

Example: Proving Axiom S with Resolution

▷ **Example 12.3.9.** Clause Normal Form transformation

$$\begin{array}{c}
((P\Rightarrow Q\Rightarrow R)\Rightarrow(P\Rightarrow Q)\Rightarrow P\Rightarrow R)^{\mathsf{F}}\\ \hline
(P\Rightarrow Q\Rightarrow R)^{\mathsf{T}}\,;\,((P\Rightarrow Q)\Rightarrow P\Rightarrow R)^{\mathsf{F}}\\ \hline
P^{\mathsf{F}}\lor(Q\Rightarrow R)^{\mathsf{T}}\,;\,(P\Rightarrow Q)^{\mathsf{T}}\,;\,(P\Rightarrow R)^{\mathsf{F}}\\ \hline
P^{\mathsf{F}}\lor Q^{\mathsf{F}}\lor R^{\mathsf{T}}\,;\,P^{\mathsf{F}}\lor Q^{\mathsf{T}}\,;\,P^{\mathsf{T}}\,;\,R^{\mathsf{F}}
\end{array}$$

Result $\{P^{\mathsf{F}}\lor Q^{\mathsf{F}}\lor R^{\mathsf{T}}\,,\,P^{\mathsf{F}}\lor Q^{\mathsf{T}}\,,\,P^{\mathsf{T}}\,,\,R^{\mathsf{F}}\}$

▷ **Example 12.3.10.** Resolution Proof

1	$P^{\mathsf{F}}\lor Q^{\mathsf{F}}\lor R^{\mathsf{T}}$	initial
2	$P^{\mathsf{F}}\lor Q^{\mathsf{T}}$	initial
3	P^{T}	initial
4	R^{F}	initial
5	$P^{\mathsf{F}}\lor Q^{\mathsf{F}}$	resolve 1.3 with 4.1
6	Q^{F}	resolve 5.1 with 3.1
7	P^{F}	resolve 2.2 with 6.1
8	$\square$	resolve 7.1 with 3.1

Clause Set Simplification

▷ **Observation:** Let Δ be a clause set, l a literal with $l\in\Delta$ (unit clause), and Δ' be Δ where

- ▷ all clauses $l\lor C$ have been removed and
- ▷ and all clauses $\overline{l}\lor C$ have been shortened to C.

Then Δ is satisfiable, iff Δ' is. We call Δ' the clause set simplification of Δ wrt. l.

▷ **Corollary 12.3.11.** *Adding clause set simplification wrt. unit clauses to $\mathcal{R}_0$ does not affect soundness and completeness.*

▷ This is almost always a good idea! (clause set simplification is cheap)

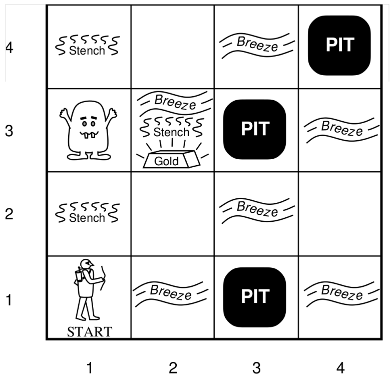
12.3.2 Killing a Wumpus with Propositional Inference

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/23713`.

Let us now consider an extended example, where we also address the question how inference in PL^0 – here resolution is embedded into the rational agent metaphor we use in AI-1: we come back to the Wumpus world.

Applying Propositional Inference: Where is the Wumpus?

- **Example 12.3.12 (Finding the Wumpus).** The situation and what the agent knows



1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 A S OK	2,2	3,2	4,2
1,1 V OK	2,1 B V OK	3,1	4,1

- What should the agent do next and why?
- **One possibility**: Convince yourself that the Wumpus is in $[1,3]$ and shoot it.
- What is the general mechanism here? (for the agent function)

Before we come to the general mechanism, we will go into how we would “convince ourselves that the Wumpus is in $[1,3]$.

Where is the Wumpus? Our Knowledge

- **Idea:** We formalize the knowledge about the Wumpus world in PL^0 and use a test calculus to check for entailment.
- **Simplification:** We worry only about the Wumpus and stench: $S_{i,j} \widehat{=}$ *stench in* $[i,j]$, $W_{i,j} \widehat{=}$ Wumpus in $[i,j]$.

- **Propositions whose value we know:** $\neg S_{1,1}$, $\neg W_{1,1}$, $\neg S_{2,1}$, $\neg W_{2,1}$, $S_{1,2}$, $\neg W_{1,2}$.
- **Knowledge about the Wumpus and smell:**
  From *Cell adjacent to Wumpus: Stench (else: None)*, we get

$$\begin{aligned} R_1 &:= \neg S_{1,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,1}\\ R_2 &:= \neg S_{2,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{2,1} \wedge \neg W_{2,2} \wedge \neg W_{3,1}\\ R_3 &:= \neg S_{1,2} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,2} \wedge \neg W_{1,3}\\ R_4 &:= S_{1,2} \Rightarrow (W_{1,3} \vee W_{2,2} \vee W_{1,1})\\ &\vdots \end{aligned}$$

- **To show:**

  $R_1, R_2, R_3, R_4 \vDash W_{1,3}$ (we will use resolution)

The first in is to compute the clause normal form of the relevant knowledge.

And Now Using Resolution Conventions

- We obtain the clause set Δ composed of the following clauses:
  - **Propositions whose value we know**: $S_{1,1}^{\mathsf{F}}$, $W_{1,1}^{\mathsf{F}}$, $S_{2,1}^{\mathsf{F}}$, $W_{2,1}^{\mathsf{F}}$, $S_{1,2}^{\mathsf{T}}$, $W_{1,2}^{\mathsf{F}}$
  - **Knowledge about the Wumpus and smell**:

from	clauses
R_1	$S_{1,1}^{\mathsf{T}} \vee W_{1,1}^{\mathsf{F}}$, $S_{1,1}^{\mathsf{T}} \vee W_{1,2}^{\mathsf{F}}$, $S_{1,1}^{\mathsf{T}} \vee W_{2,1}^{\mathsf{F}}$
R_2	$S_{2,1}^{\mathsf{T}} \vee W_{1,1}^{\mathsf{F}}$, $S_{2,1}^{\mathsf{T}} \vee W_{2,1}^{\mathsf{F}}$, $S_{2,1}^{\mathsf{T}} \vee W_{2,2}^{\mathsf{F}}$, $S_{2,1}^{\mathsf{T}} \vee W_{3,1}^{\mathsf{F}}$
R_3	$S_{1,2}^{\mathsf{T}} \vee W_{1,1}^{\mathsf{F}}$, $S_{1,2}^{\mathsf{T}} \vee W_{1,2}^{\mathsf{F}}$, $S_{1,2}^{\mathsf{T}} \vee W_{2,2}^{\mathsf{F}}$, $S_{1,2}^{\mathsf{T}} \vee W_{1,3}^{\mathsf{F}}$
R_4	$S_{1,2}^{\mathsf{F}} \vee W_{1,3}^{\mathsf{T}} \vee W_{2,2}^{\mathsf{T}} \vee W_{1,1}^{\mathsf{T}}$

  - **Negated goal formula**: $W_{1,3}^{\mathsf{F}}$

Given this clause normal form, we only need to find generate empty clause via repeated applications of the resolution rule.

Resolution Proof Killing the Wumpus!

- **Example 12.3.13 (Where is the Wumpus).** We show a derivation that proves that he is in $(1,3)$.
  - *Assume the Wumpus is not in $(1,3)$. Then either there's no stench in $(1,2)$, or the Wumpus is in some other neigbor cell of $(1,2)$.*
    - Parents: $W_{1,3}^{\mathsf{F}}$ and $S_{1,2}^{\mathsf{F}} \vee W_{1,3}^{\mathsf{T}} \vee W_{2,2}^{\mathsf{T}} \vee W_{1,1}^{\mathsf{T}}$.
    - Resolvent: $S_{1,2}^{\mathsf{F}} \vee W_{2,2}^{\mathsf{T}} \vee W_{1,1}^{\mathsf{T}}$.
  - *There's a stench in $(1,2)$, so it must be another neighbor.*
    - Parents: $S_{1,2}^{\mathsf{T}}$ and $S_{1,2}^{\mathsf{F}} \vee W_{2,2}^{\mathsf{T}} \vee W_{1,1}^{\mathsf{T}}$.
    - Resolvent: $W_{2,2}^{\mathsf{T}} \vee W_{1,1}^{\mathsf{T}}$.

- *We've been to $(1,1)$, and there's no Wumpus there, so it can't be $(1,1)$.*
  - Parents: $W_{1,1}{}^{\mathsf{F}}$ and $W_{2,2}{}^{\mathsf{T}} \vee W_{1,1}{}^{\mathsf{T}}$.
  - Resolvent: $W_{2,2}{}^{\mathsf{T}}$.
- *There is no stench in $(2,1)$ so it can't be $(2,2)$ either, in contradiction.*
  - Parents: $S_{2,1}{}^{\mathsf{F}}$ and $S_{2,1}{}^{\mathsf{T}} \vee W_{2,2}{}^{\mathsf{F}}$.
  - Resolvent: $W_{2,2}{}^{\mathsf{F}}$.
  - Parents: $W_{2,2}{}^{\mathsf{F}}$ and $W_{2,2}{}^{\mathsf{T}}$.
  - Resolvent: $\Box$.

As resolution is sound, we have shown that indeed $R_1, R_2, R_3, R_4 \models W_{1,3}$.

Now that we have seen how we can use propositional inference to derive consequences of the percepts and world knowledge, let us come back to the question of a general mechanism for agent functions with propositional inference.

Where does the Conjecture $W_{1,3}{}^{\mathsf{F}}$ come from?

- **Question:** Where did the $W_{1,3}{}^{\mathsf{F}}$ come from?
- **Observation 12.3.14.** *We need a general mechanism for making conjectures.*
- **Idea:** Interpret the Wumpus world as a search problem $\mathcal{P} := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$ where
  - the states $\mathcal{S}$ are given by the cells (and agent orientation) and
  - the actions $\mathcal{A}$ by the possible actions of the agent.

  Use tree search as the main agent function and a test calculus for testing all dangers (pits), opportunities (gold) and the Wumpus.
- **Example 12.3.15 (Back to the Wumpus).** In ??, the agent is in $[1,2]$, it has perceived stench, and the possible actions include shoot, and goForward. Evaluating either of these leads to the conjecture $W_{1,3}$. And since $W_{1,3}$ is entailed, the action shoot probably comes out best, heuristically.
- **Remark:** Analogous to the backtracking with inference algorithm from CSP.

Admittedly, the search framework from **??** does not quite cover the agent function we have here, since that assumes that the world is fully observable, which the Wumpus world is emphatically not. But it already gives us a good impression of what would be needed for the "general mechanism".

12.4 Conclusion

Summary

- Every propositional formula can be brought into conjunctive normal form (CNF), which can be identified with a set of clauses.

▷ The tableau and resolution calculi are deduction procedures based on trying to derive a contradiction from the negated theorem (a closed tableau or the empty clause). They are refutation complete, and can be used to prove $\mathrm{KB} \models \mathbf{A}$ by showing that $\mathrm{KB} \cup \{\neg\mathbf{A}\}$ is unsatisfiable.

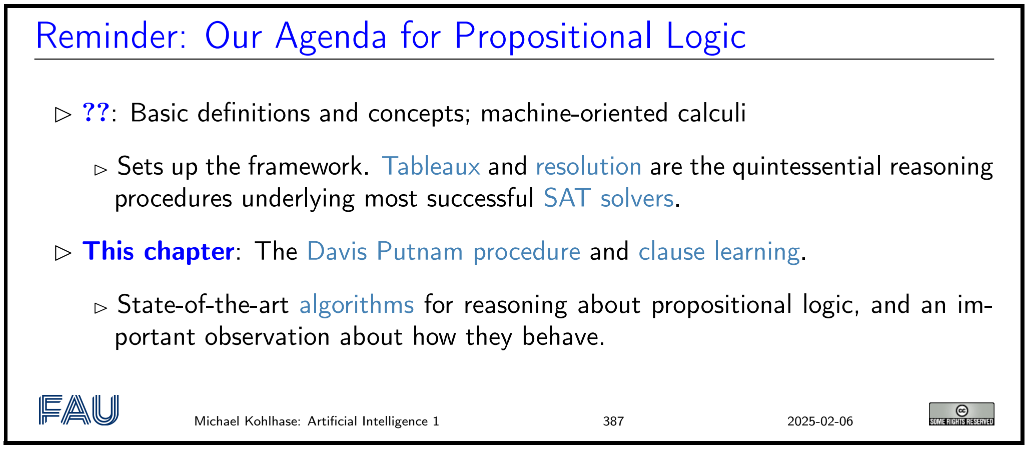
Excursion: A full analysis of any calculus needs a completeness proof. We will not cover this in AI-1, but provide one for the calculi introduced so far in**??**.

Chapter 13

Propositional Reasoning: SAT Solvers

13.1 Introduction

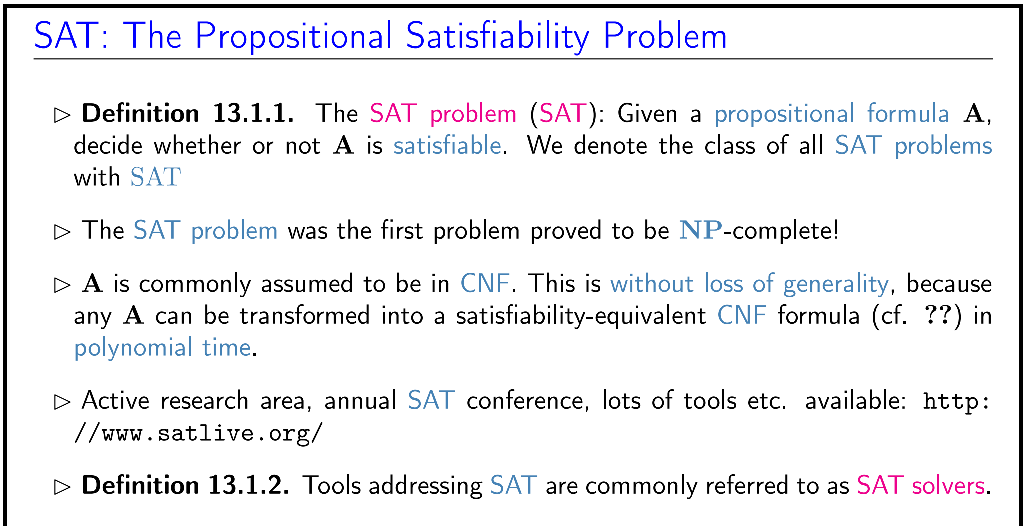
A Video Nugget covering this section can be found at `https://fau.tv/clip/id/25019`.



Reminder: Our Agenda for Propositional Logic

- ??: Basic definitions and concepts; machine-oriented calculi
  - Sets up the framework. Tableaux and resolution are the quintessential reasoning procedures underlying most successful SAT solvers.
- **This chapter**: The Davis Putnam procedure and clause learning.
  - State-of-the-art algorithms for reasoning about propositional logic, and an important observation about how they behave.

FAU Michael Kohlhase: Artificial Intelligence 1 387 2025-02-06



SAT: The Propositional Satisfiability Problem

- **Definition 13.1.1.** The SAT problem (SAT): Given a propositional formula $\mathbf{A}$, decide whether or not $\mathbf{A}$ is satisfiable. We denote the class of all SAT problems with SAT
- The SAT problem was the first problem proved to be **NP**-complete!
- $\mathbf{A}$ is commonly assumed to be in CNF. This is without loss of generality, because any $\mathbf{A}$ can be transformed into a satisfiability-equivalent CNF formula (cf. ??) in polynomial time.
- Active research area, annual SAT conference, lots of tools etc. available: `http://www.satlive.org/`
- **Definition 13.1.2.** Tools addressing SAT are commonly referred to as SAT solvers.

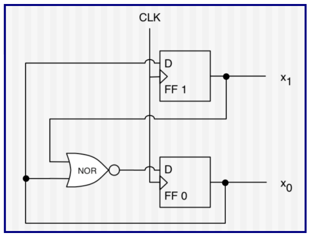
- ▷ **Recall:** To decide whether $\mathrm{KB} \models \mathbf{A}$, decide satisfiability of $\theta := \mathrm{KB} \cup \{\neg\mathbf{A}\}$: θ is unsatisfiable iff $\mathrm{KB} \models \mathbf{A}$.
- ▷ **Consequence:** Deduction can be performed using SAT solvers.

SAT vs. CSP

- ▷ **Recall:** Constraint network $\langle V, D, C\rangle$ has variables $v \in V$ with finite domains $D_v \in D$, and binary constraints $C_{uv} \in C$ which are relations over u, and v specifying the permissible combined assignments to u and v. One extension is to allow constraints of higher arity.
- ▷ **Observation 13.1.3 (SAT: A kind of CSP).** *SAT can be viewed as a CSP problem in which all variable domains are Boolean, and the constraints have unbounded arity.*
- ▷ **Theorem 13.1.4 (Encoding CSP as SAT).** *Given any constraint network $\mathcal{C}$, we can in low order polynomial time construct a CNF formula $\mathbf{A}(\mathcal{C})$ that is satisfiable iff $\mathcal{C}$ is solvable.*
- ▷ *Proof:* We design a formula, relying on known transformation to CNF
  1. encode multi-XOR for each variable
  2. encode each constraint by DNF over relation
  3. Running time: $\mathcal{O}(nd^2 + md^2)$ where n is the number of variables, d the domain size, and m the number of constraints.
- ▷ **Upshot:** Anything we can do with CSP, we can (in principle) do with SAT.

Example Application: Hardware Verification

- ▷ **Example 13.1.5 (Hardware Verification).**



- ▷ Counter, repeatedly from $c = 0$ to $c = 2$.
- ▷ 2 bits x_1 and x_0; $c = 2 * x_1 + x_0$.
- ▷ (FF$\widehat{=}$ Flip-Flop, D $\widehat{=}$ Data IN, CLK $\widehat{=}$ Clock)
- ▷ **To Verify**: If $c < 3$ in current clock cycle, then $c < 3$ in next clock cycle.

- ▷ **Step 1:** Encode into propositional logic.
  - ▷ **Propositions**: x_1, x_0; and y_1, y_0 (value in next cycle).
  - ▷ **Transition relation**: $y_1 \Leftrightarrow y_0$; $y_0 \Leftrightarrow \neg(x_1 \vee x_0)$.
  - ▷ **Initial state**: $\neg(x_1 \wedge x_0)$.
  - ▷ **Error property**: $x_1 \wedge y_0$.
- ▷ **Step 2:** Transform to CNF, encode as a clause set Δ.

- **Clauses**: $y_1^{\mathsf{F}} \vee x_0^{\mathsf{T}}$, $y_1^{\mathsf{T}} \vee x_0^{\mathsf{F}}$, $y_0^{\mathsf{T}} \vee x_1^{\mathsf{T}} \vee x_0^{\mathsf{T}}$, $y_0^{\mathsf{F}} \vee x_1^{\mathsf{F}}$, $y_0^{\mathsf{F}} \vee x_0^{\mathsf{F}}$, $x_1^{\mathsf{F}} \vee x_0^{\mathsf{F}}$, y_1^{T}, y_0^{T}.

▷ **Step 3:** Call a SAT solver (up next).

Our Agenda for This Chapter

▷ **The Davis-Putnam (Logemann-Loveland) Procedure:** How to systematically test satisfiability?

- The quintessential SAT solving procedure, DPLL.

▷ **DPLL is (A Restricted Form of) Resolution:** How does this relate to what we did in the last chapter?

- mathematical understanding of DPLL.

▷ **Why Did Unit Propagation Yield a Conflict?:** How can we analyze which mistakes were made in “dead” search branches?

- Knowledge is power, see next.

▷ **Clause Learning:** How can we learn from our mistakes?

- One of the key concepts, perhaps *the* key concept, underlying the success of SAT.

▷ **Phase Transitions – Where the Really Hard Problems Are:** Are *all* formulas “hard” to solve?

- The answer is “no”. And in some cases we can figure out exactly when they are/aren’t hard to solve.

13.2 The Davis-Putnam (Logemann-Loveland) Procedure

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/25026`.

The DPLL Procedure

▷ **Definition 13.2.1.** The Davis Putnam procedure (DPLL) is a SAT solver called on a clause set Δ and the empty assignment ϵ. It interleaves unit propagation (UP) and splitting:

```
function DPLL($\Delta$,$I$) returns a partial assignment $I$, or ‘‘unsatisfiable’’
  /* Unit Propagation (UP) Rule: */
  $\Delta'$ := a copy of $\Delta$; $I'$ := $I$
  while $\Delta'$ contains a unit clause $C = P^\alpha$ do
    extend $I'$ with $[\alpha/P]$, clause−set simplify $\Delta'$
   /* Termination Test: */
   if $\Box \in \Delta'$ then return ‘‘unsatisfiable’’
```

```
 if $\Delta' = \{\}$ then return $I'$
 /* Splitting Rule: */
select some proposition $P$ for which $I'$ is not defined
$I''$ := $I'$ extended with one truth value for $P$; $\Delta''$ := a copy of $\Delta'$; simplify $\Delta''$
if $I'''$ := DPLL($\Delta''$,$I''$) $\neq$ ‘‘unsatisfiable’’ then return $I'''$
$I''$ := $I'$ extended with the other truth value for $P$; $\Delta''$ := $\Delta'$; simplify $\Delta''$
return DPLL($\Delta''$,$I''$)
```

▷ In practice, of course one uses flags etc. instead of “copy”.

DPLL: Example (Vanilla1)

▷ **Example 13.2.2 (UP and Splitting).** Let $\Delta := P^{\mathsf{T}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{F}} ; P^{\mathsf{F}} \vee Q^{\mathsf{F}} ; R^{\mathsf{T}} ; P^{\mathsf{T}} \vee Q^{\mathsf{F}}$

1. UP Rule: $R \mapsto \mathsf{T}$
   $P^{\mathsf{T}} \vee Q^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{F}} ; P^{\mathsf{T}} \vee Q^{\mathsf{F}}$
2. Splitting Rule:

2a. $P \mapsto \mathsf{F}$ $Q^{\mathsf{T}} ; Q^{\mathsf{F}}$	2b. $P \mapsto \mathsf{T}$ Q^{F}
3a. UP Rule: $Q \mapsto \mathsf{T}$ □ **returning “unsatisfiable”**	3b. UP Rule: $Q \mapsto \mathsf{F}$ clause set empty **returning “$R \mapsto \mathsf{T}, P \mapsto \mathsf{T}, Q \mapsto \mathsf{F}$**

DPLL: Example (Vanilla2)

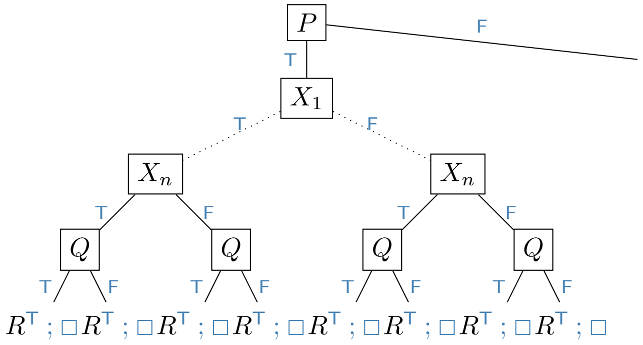
▷ **Observation:** Sometimes UP is all we need.

▷ **Example 13.2.3.** Let $\Delta := Q^{\mathsf{F}} \vee P^{\mathsf{F}} ; P^{\mathsf{T}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} \vee S^{\mathsf{F}} ; Q^{\mathsf{T}} \vee S^{\mathsf{F}} ; R^{\mathsf{T}} \vee S^{\mathsf{F}} ; S^{\mathsf{T}}$

1. UP Rule: $S \mapsto \mathsf{T}$
   $Q^{\mathsf{F}} \vee P^{\mathsf{F}} ; P^{\mathsf{T}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} ; Q^{\mathsf{T}} ; R^{\mathsf{T}}$
2. UP Rule: $Q \mapsto \mathsf{T}$
   $P^{\mathsf{F}} ; P^{\mathsf{T}} \vee R^{\mathsf{F}} ; R^{\mathsf{T}}$
3. UP Rule: $R \mapsto \mathsf{T}$
   $P^{\mathsf{F}} ; P^{\mathsf{T}}$
4. UP Rule: $P \mapsto \mathsf{T}$
   □

DPLL: Example (Redundance1)

▷ **Example 13.2.4.** We introduce some nasty redundance to make DPLL slow.
$\Delta := P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{F}}$
DPLL on $\Delta ; \Theta$ with $\Theta := {X_1}^{\mathsf{T}} \vee \ldots \vee {X_n}^{\mathsf{T}} ; {X_1}^{\mathsf{F}} \vee \ldots \vee {X_n}^{\mathsf{F}}$


P
F
T
X1
T
F
Xn
Xn
T
F
T
F
Q
Q
Q
Q
T
F
T
F
T
F
T
F
R^T ; □ R^T ; □ R^T ; □ R^T ; □ R^T ; □ R^T ; □ R^T ; □ R^T ; □

Properties of DPLL

▷ **Unsatisfiable case:** What can we say if "unsatisfiable" is returned?

- ▹ In this case, we know that Δ is unsatisfiable: Unit propagation is *sound*, in the sense that it does not reduce the set of solutions.

▷ **Satisfiable case:** What can we say when a partial interpretation I is returned?

- ▹ Any extension of I to a complete interpretation satisfies Δ. (By construction, I suffices to satisfy all clauses.)

▷ Déjà Vu, Anybody?

▷ DPLL $\widehat{=}$ backtracking with inference, where inference $\widehat{=}$ unit propagation.

- ▹ Unit propagation is sound: It does not reduce the set of solutions.
- ▹ Running time is exponential in worst case, good variable/value selection strategies required.

13.3 DPLL $\widehat{=}$ (A Restricted Form of) Resolution

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/27022`.

In the last slide we have discussed the semantic properties of the DPLL procedure: DPLL is (refutation) sound and complete. Note that this is a theoretical resultin the sense that the algorithm is, but that does not mean that a particular implementation of DPLL might not contain bugs that affect sounds and completeness.

In the satisfiable case, DPLL returns a satisfying variable assignment, which we can check (in low-order polynomial time) but in the unsatisfiable case, it just reports on the fact that it has tried all branches and found nothing. This is clearly unsatisfactory, and we will address this situation now by presenting a way that DPLL can output a resolution proof in the unsatisfiable case.

UP $\widehat{=}$ Unit Resolution

- **Observation:** The unit propagation (UP) rule corresponds to a calculus:

```
while $\Delta'$ contains a unit clause $\{l\}$ do
  extend $I'$ with the respective truth value for the proposition underlying $l$
  simplify $\Delta'$ /* remove false literals */
```

- **Definition 13.3.1 (Unit Resolution).** Unit resolution (UR) is the test calculus consisting of the following inference rule:

$$\frac{C \vee P^{\alpha} \quad P^{\beta} \quad \alpha \neq \beta}{C}\,\mathrm{UR}$$

- Unit propagation $\widehat{=}$ resolution restricted to cases where one parent is unit clause.
- **Observation 13.3.2 (Soundness).** UR *is refutation sound.* (*since resolution is*)
- **Observation 13.3.3 (Completeness).** UR *is not refutation complete (alone).*
- **Example 13.3.4.** $P^{\mathsf{T}} \vee Q^{\mathsf{T}}\,;\, P^{\mathsf{T}} \vee Q^{\mathsf{F}}\,;\, P^{\mathsf{F}} \vee Q^{\mathsf{T}}\,;\, P^{\mathsf{F}} \vee Q^{\mathsf{F}}$ is unsatisfiable but UR cannot derive the empty clause $\square$.
- UR makes only limited inferences, as long as there are unit clauses. It does not guarantee to infer everything that can be inferred.

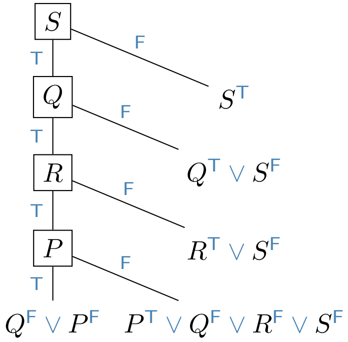
DPLL vs. Resolution

- **Definition 13.3.5.** We define the number of decisions of a DPLL run as the total number of times a truth value was set by either unit propagation or splitting.
- **Theorem 13.3.6.** *If DPLL returns "unsatisfiable" on Δ, then $\Delta \vdash_{\mathcal{R}_0} \square$ with a resolution proof whose length is at most the number of decisions.*
- *Proof:* Consider first DPLL without UP
  1. Consider any leaf node N, for proposition X, both of whose truth values directly result in a clause C that has become empty.
  2. Then for $X = \mathsf{F}$ the respective clause C must contain X^{T}; and for $X = \mathsf{T}$ the respective clause C must contain X^{F}. Thus we can resolve these two clauses to a clause $C(N)$ that does not contain X.
  3. $C(N)$ can contain only the negations of the decision literals $l_1, \ldots, l_k$ above N. Remove N from the tree, then iterate the argument. Once the tree is empty, we have derived the empty clause.
  4. Unit propagation can be simulated via applications of the splitting rule, choosing a proposition that is constrained by a unit clause: One of the two truth values then immediately yields an empty clause.

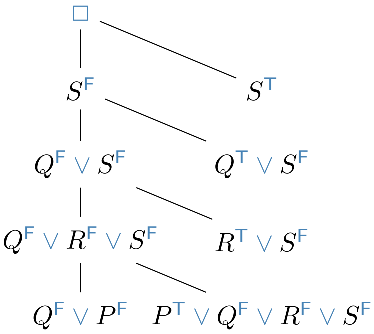
DPLL vs. Resolution: Example (Vanilla2)

- **Observation:** The proof of **??** is constructive, so we can use it as a method to read of a resolution proof from a DPLL trace.
- **Example 13.3.7.** We follow the steps in the proof of **??** for $\Delta := Q^{\mathsf{F}} \vee P^{\mathsf{F}} ; P^{\mathsf{T}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} \vee S^{\mathsf{F}} ; Q^{\mathsf{T}} \vee S^{\mathsf{F}} ; R^{\mathsf{T}} \vee S^{\mathsf{F}} ; S^{\mathsf{T}}$

DPLL: (Without UP; leaves annotated with clauses that became empty)


S
F
T
S^{T}
Q
F
T
$Q^{\mathsf{T}} \vee S^{\mathsf{F}}$
R
F
T
$R^{\mathsf{T}} \vee S^{\mathsf{F}}$
P
F
T
$Q^{\mathsf{F}} \vee P^{\mathsf{F}}$
$P^{\mathsf{T}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} \vee S^{\mathsf{F}}$

Resolution proof from that DPLL tree:


$\Box$
S^{F}
S^{T}
$Q^{\mathsf{F}} \vee S^{\mathsf{F}}$
$Q^{\mathsf{T}} \vee S^{\mathsf{F}}$
$Q^{\mathsf{F}} \vee R^{\mathsf{F}} \vee S^{\mathsf{F}}$
$R^{\mathsf{T}} \vee S^{\mathsf{F}}$
$Q^{\mathsf{F}} \vee P^{\mathsf{F}}$
$P^{\mathsf{T}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} \vee S^{\mathsf{F}}$

- **Intuition:** From a (top-down) DPLL tree, we generate a (bottom-up) resolution proof.

For reference, we give the full proof here.

Theorem 13.3.8. *If DPLL returns "unsatisfiable" on a clause set Δ, then $\Delta \vdash_{\mathcal{R}_0} \Box$ with a $\mathcal{R}_0$-derivation whose length is at most the number of decisions.*

Proof: Consider first DPLL with no unit propagation.

1. If the search tree is not empty, then there exists a leaf node N, i.e., a node associated to proposition X so that, for each value of X, the partial assignment directly results in an empty clause.
2. Denote the parent decisions of N by $L_1, \ldots, L_k$, where L_i is a literal for proposition X_i and the search node containing X_i is N_i.
3. Denote the empty clause for X by $C(N,X)$, and denote the empty clause for X^{F} by $C(N,X^{\mathsf{F}})$.
4. For each $x \in \{X^{\mathsf{T}}, X^{\mathsf{F}}\}$ we have the following properties:
   1. $x^{\mathsf{F}} \in C(N,x)$; and
   2. $C(N,x) \subseteq \{x^{\mathsf{F}}, \overline{L_1}, \ldots, \overline{L_k}\}$.

   Due to , we can resolve $C(N,X)$ with $C(N,X^{\mathsf{F}})$; denote the outcome clause by $C(N)$.
5. We obviously have that (1) $C(N) \subseteq \{\overline{L_1}, \ldots, \overline{L_k}\}$.
6. The proof now proceeds by removing N from the search tree and attaching $C(N)$ at the L_k branch of N_k, in the role of $C(N_k, L_k)$ as above. Then we select the next leaf node N' and iterate the argument; once the tree is empty, by (1) we have derived the empty clause. What we need to show is that, in each step of this iteration, we preserve the properties (a) and (b) for all leaf nodes. Since we did not change anything in other parts of the tree, the only node we need to show this for is $N' := N_k$.
7. Due to (1), we have (b) for N_k. But we do not necessarily have (a): $C(N) \subseteq \{\overline{L_1}, \ldots, \overline{L_k}\}$, but there are cases where $\overline{L_k} \notin C(N)$ (e.g., if X_k is not contained in any clause and thus

branching over it was completely unnecessary). If so, however, we can simply remove N_k and all its descendants from the tree as well. We attach $C(N)$ at the $L_{(k-1)}$ branch of $N_{(k-1)}|$, in the role of $C(N_{(k-1)}, L_{(k-1)})$. If $\overline{L_{(k-1)}} \in C(N)$ then we have (a) for $N' := N_{(k-1)}$ and can stop. If $L_{(k-1)}^{\mathsf{F}} \notin C(N)$, then we remove $N_{(k-1)}$ and so forth, until either we stop with (a), or have removed N_1 and thus must already have derived the empty clause (because $C(N) \subseteq \{\overline{L_1}, \ldots, \overline{L_k}\} \backslash \{\overline{L_1}, \ldots, \overline{L_k}\}$).

8. Unit propagation can be simulated via applications of the splitting rule, choosing a proposition that is constrained by a unit clause: One of the two truth values then immediately yields an empty clause.

DPLL vs. Resolution: Discussion

- **So What?:** The theorem we just proved helps to *understand* DPLL: DPLL is an efficient practical method for conducting resolution proofs.
- **In fact:** DPLL $\widehat{=}$ tree resolution.
- **Definition 13.3.9.** In a tree resolution, each derived clause C is used only once (at its parent).
- **Problem:** The same C must be derived anew every time it is used!
- **This is a fundamental weakness:** There are inputs Δ whose shortest tree resolution proof is exponentially longer than their shortest (general) resolution proof.
- **Intuitively:** DPLL makes the same mistakes over and over again.
- **Idea:** DPLL should learn from its mistakes on one search branch, and apply the learned knowledge to other branches.
- **To the rescue:** clause learning (up next)

FAU Michael Kohlhase: Artificial Intelligence 1 400 2025-02-06

Excursion: Practical SAT solvers use a technique called CDCL that analyzes failure and learns from that in terms of inferred clauses. Unfortunately, we cannot cover this in AI-1.??.

13.4 Conclusion

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/25090`.

Summary

- SAT solvers decide satisfiability of CNF formulas. This can be used for deduction, and is highly successful as a general problem solving technique (e.g., in verification).
- DPLL $\widehat{=}$ backtracking with inference performed by unit propagation (UP), which iteratively instantiates unit clauses and simplifies the formula.
- DPLL proofs of unsatisfiability correspond to a restricted form of resolution. The restriction forces DPLL to "makes the same mistakes over again".
- Implication graphs capture how UP derives conflicts. Their analysis enables us to do clause learning. DPLL with clause learning is called CDCL. It corresponds to full

resolution, not “making the same mistakes over again”.

- ▷ CDCL is state of the art in applications, routinely solving formulas with millions of propositions.
- ▷ In particular random formula distributions, typical problem hardness is characterized by phase transitions.

State of the Art in SAT

- ▷ **SAT competitions:**
  - ▷ Since beginning of the 90s `http://www.satcompetition.org/`
  - ▷ *random* vs. *industrial* vs. *handcrafted* benchmarks.
  - ▷ Largest industrial instances: $> 1.000.000$ propositions.
- ▷ **State of the art is CDCL:**
  - ▷ Vastly superior on handcrafted and industrial benchmarks.
  - ▷ Key techniques: clause learning! Also: Efficient implementation (UP!), good branching heuristics, random restarts, portfolios.
- ▷ **What about local search?:**
  - ▷ Better on random instances.
  - ▷ No “dramatic” progress in last decade.
  - ▷ Parameters are difficult to adjust.

But – What About Local Search for SAT?

- ▷ **There’s a wealth of research on local search for SAT, e.g.:**
- ▷ **Definition 13.4.1.** The GSAT algorithm **OUTPUT**: a satisfying truth assignment of Δ, if found

```
function GSAT ($\Delta$, $MaxFlips$ $MaxTries$
  for $i$ :=1 to $MaxTries$
    $I$ := a randomly—generated truth assignment
    for $j$ :=1 to $MaxFlips$
    if $I$ satisfies $\Delta$ then return $I$
        $X$:= a proposition reversing whose truth assignment gives
        the largest increase in the number of satisfied clauses
       $I$ := $I$ with the truth assignment of $X$ reversed
    end for
  end for
  return ‘‘no satisfying assignment found’’
```

▷ local search is not as successful in SAT applications, and the underlying ideas are very similar to those presented in ?? (Not covered here)

Topics We Didn't Cover Here

▷ **Variable/value selection heuristics**: A whole zoo is out there.

▷ **Implementation techniques**: One of the most intensely researched subjects. Famous "watched literals" technique for UP had huge practical impact.

▷ **Local search**: In space of all truth value assignments. GSAT (slide 403) had huge impact at the time (1992), caused huge amount of follow-up work. Less intensely researched since clause learning hit the scene in the late 90s.

▷ **Portfolios**: How to combine several SAT solvers efficiently?

▷ **Random restarts**: Tackling heavy-tailed runtime distributions.

▷ **Tractable SAT**: Polynomial-time sub-classes (most prominent: 2-SAT, Horn formulas).

▷ **MaxSAT**: Assign weight to each clause, maximize weight of satisfied clauses (= optimization version of SAT).

▷ **Resolution special cases**: There's a universe in between unit resolution and full resolution: trade off inference vs. search.

▷ **Proof complexity**: Can one resolution special case X simulate another one Y polynomially? Or is there an exponential separation (example families where X is exponentially less efficient than Y)?

Suggested Reading:

- *Chapter 7: Logical Agents*, Section 7.6.1 [RN09].
  - Here, RN describe DPLL, i.e., basically what I cover under "The Davis-Putnam (Logemann-Loveland) Procedure".
  - That's the only thing they cover of this Chapter's material. (And they even mark it as "can be skimmed on first reading".)
  - This does not do the state of the art in SAT any justice.
- *Chapter 7: Logical Agents*, Sections 7.6.2, 7.6.3, and 7.7 [RN09].
  - Sections 7.6.2 and 7.6.3 say a few words on local search for SAT, which I recommend as additional background reading. Section 7.7 describes in quite some detail how to build an agent using propositional logic to take decisions; nice background reading as well.

Chapter 14

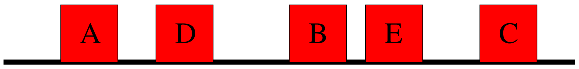
First-Order Predicate Logic

14.1 Motivation: A more Expressive Language

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/25091`.

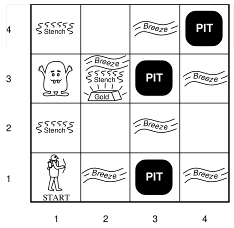
Let's Talk About Blocks, Baby . . .

- ▷ **Question:** What do you see here?



- ▷ **You say:** "All blocks are red"; "All blocks are on the table"; "A is a block".
- ▷ **And now:** Say it in propositional logic!
- ▷ **Answer:** "isRedA","isRedB", . . . , "onTableA", "onTableB", . . . , "isBlockA", . . .
- ▷ **Wait a sec!:** Why don't we just say, e.g., "AllBlocksAreRed" and "isBlockA"?
- ▷ **Problem:** Could we conclude that A is red? (No)
  These statements are atomic (just strings); their inner structure ("all blocks", "is a block") is not captured.
- ▷ **Idea:** Predicate Logic (PL^1) extends propositional logic with the ability to explicitly speak about objects and their properties.
- ▷ **How?:** Variables ranging over objects, predicates describing object properties, . . .
- ▷ **Example 14.1.1.** "$\forall x.\mathrm{block}(x) \Rightarrow \mathrm{red}(x)$"; "$\mathrm{block}(\mathbf{A})$"

Let's Talk About the Wumpus Instead?

▷ 

Percepts: $[\mathit{Stench}, \mathit{Breeze}, \mathit{Glitter}, \mathit{Bump}, \mathit{Scream}]$

- Cell adjacent to Wumpus: Stench (else: None).
- Cell adjacent to Pit: Breeze (else: None).
- Cell that contains gold: $\mathit{Glitter}$ (else: None).
- You walk into a wall: Bump (else: None).
- Wumpus shot by arrow: Scream (else: None).

- Say, in propositional logic: “Cell adjacent to Wumpus: Stench.”
  - $W_{1,1} \Rightarrow S_{1,2} \wedge S_{2,1}$
  - $W_{1,2} \Rightarrow S_{2,2} \wedge S_{1,1} \wedge S_{1,3}$
  - $W_{1,3} \Rightarrow S_{2,3} \wedge S_{1,2} \wedge S_{1,4}$
  - . . .
- **Note:** Even when we *can* describe the problem suitably, for the desired reasoning, the propositional formulation typically is way too large to write (by hand).
- **PL1 solution:** “$\forall x.\mathrm{Wumpus}(x) \Rightarrow (\forall y.\mathrm{adj}(x,y) \Rightarrow \mathrm{stench}(y))$”

Blocks/Wumpus, Who Cares? Let’s Talk About Numbers!

- Even worse!
- **Example 14.1.2 (Integers).** A limited vocabulary to talk about these
  - The objects: $\{1,2,3,\ldots\}$.
  - Predicate 1: “$\mathrm{even}(x)$” should be true iff x is even.
  - Predicate 2: “$\mathrm{eq}(x,y)$” should be true iff $x=y$.
  - Function: $\mathrm{succ}(x)$ maps x to $x+1$.
- **Old problem:** Say, in propositional logic, that “$1+1=2$”.
  - Inner structure of vocabulary is ignored (cf. “AllBlocksAreRed”).
  - PL1 solution: “$\mathrm{eq}(\mathrm{succ}(1),2)$”.
- **New Problem:** Say, in propositional logic, “if x is even, so is $x+2$”.
  - It is impossible to speak about infinite sets of objects!
  - PL1 solution: “$\forall x.\mathrm{even}(x) \Rightarrow \mathrm{even}(\mathrm{succ}(\mathrm{succ}(x)))$”.

Now We’re Talking

- **Example 14.1.3.**

$$\forall n.\mathrm{gt}(n,2)\Rightarrow\neg(\exists a,b,c.\mathrm{eq}(\mathrm{plus}(\mathrm{pow}(a,n),\mathrm{pow}(b,n)),\mathrm{pow}(c,n)))$$

Read: *Forall $n>2$, there are no a, b, c, such that $a^n+b^n=c^n$* (Fermat's last theorem)

- **Theorem proving in PL1:** Arbitrary theorems, in principle.
  - Fermat's last theorem is of course infeasible, but interesting theorems can and have been proved automatically.
  - See `http://en.wikipedia.org/wiki/Automated_theorem_proving`.
  - **Note**: Need to axiomatize "Plus", "PowerOf", "Equals". See `http://en.wikipedia.org/wiki/Peano_axioms`

What Are the Practical Relevance/Applications?

- **. . . even asking this question is a sacrilege:**
- (Quotes from Wikipedia)
  - *"In Europe, logic was first developed by Aristotle. Aristotelian logic became widely accepted in science and mathematics."*
  - *"The development of logic since Frege, Russell, and Wittgenstein had a profound influence on the practice of philosophy and the perceived nature of philosophical problems, and Philosophy of mathematics."*
  - *"During the later medieval period, major efforts were made to show that Aristotle's ideas were compatible with Christian faith."*
  - (In other words: the church issued for a long time that Aristotle's ideas were *in*compatible with Christian faith.)

What Are the Practical Relevance/Applications?

- **You're asking it anyhow:**
  - **Logic programming**. Prolog et al.
  - **Databases**. Deductive databases where elements of logic allow to conclude additional facts. Logic is tied deeply with database theory.
  - Semantic technology. Mega-trend since $>$ a decade. Use PL1 fragments to annotate data sets, facilitating their use and analysis.
- Prominent PL1 fragment: Web Ontology Language OWL.
- Prominent data set: The WWW. (semantic web)
- **Assorted quotes on Semantic Web and OWL:**

- ▷ *The brain of humanity.*
- ▷ *The Semantic Web will never work.*
- ▷ *A TRULY meaningful way of interacting with the Web may finally be here: the Semantic Web. The idea was proposed 10 years ago. A triumvirate of internet heavyweights – Google, Twitter, and Facebook – are making it real.*

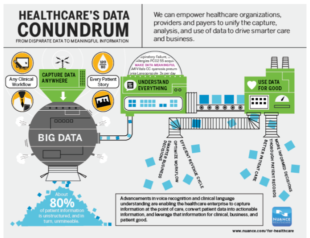
(A Few) Semantic Technology Applications

Web Queries

Jeopardy (IBM Watson)

Context-Aware Apps

Healthcare


HEALTHCARE'S DATA CONUNDRUM
FROM DISPARATE DATA TO MEANINGFUL INFORMATION
We can empower healthcare organizations, providers and payers to unify the capture, analysis, and use of data to drive smarter care and business.
CAPTURE DATA ANYWHERE
UNDERSTAND EVERYTHING
USE DATA FOR GOOD
BIG DATA
80%

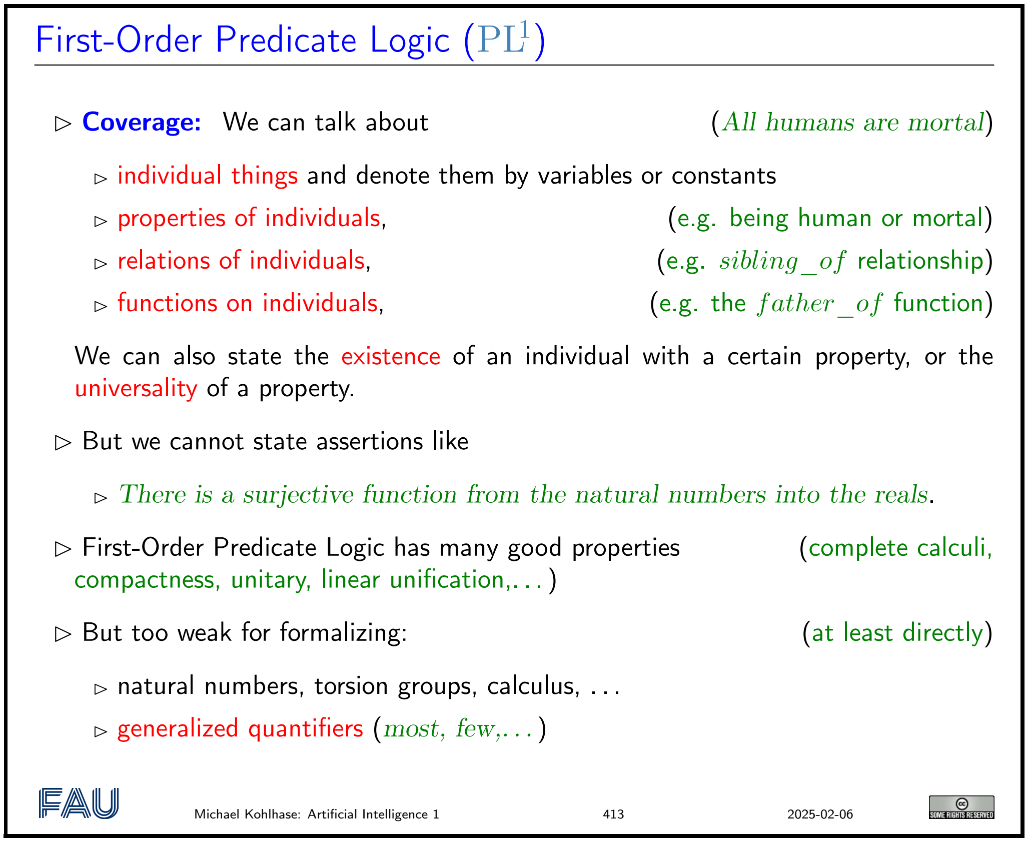
Our Agenda for This Topic

- ▷ **This Chapter**: Basic definitions and concepts; normal forms.
  - ▷ Sets up the framework and basic operations.
  - ▷ **Syntax**: How to write PL1 formulas? (Obviously required)
  - ▷ **Semantics**: What is the meaning of PL1 formulas? (Obviously required.)
  - ▷ **Normal Forms**: What are the basic normal forms, and how to obtain them? (Needed for algorithms, which are defined on these normal forms.)
- ▷ **Next Chapter**: Compilation to propositional reasoning; unification; lifted resolution/tableau.
  - ▷ Algorithmic principles for reasoning about predicate logic.

14.2 First-Order Logic

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/25093`.

First-order logic is the most widely used formal systems for modelling knowledge and inference processes. It strikes a very good bargain in the trade-off between expressivity and conceptual and computational complexity. To many people first-order logic is “the logic”, i.e. the only logic worth considering, its applications range from the foundations of mathematics to natural language semantics.



First-Order Predicate Logic (PL^1)

- **Coverage:** We can talk about (*All humans are mortal*)
  - individual things and denote them by variables or constants
  - properties of individuals, (e.g. being human or mortal)
  - relations of individuals, (e.g. *sibling_of* relationship)
  - functions on individuals, (e.g. the *father_of* function)

  We can also state the existence of an individual with a certain property, or the universality of a property.
- But we cannot state assertions like
  - *There is a surjective function from the natural numbers into the reals*.
- First-Order Predicate Logic has many good properties (complete calculi, compactness, unitary, linear unification,. . .)
- But too weak for formalizing: (at least directly)
  - natural numbers, torsion groups, calculus, . . .
  - generalized quantifiers (*most, few,*. . .)

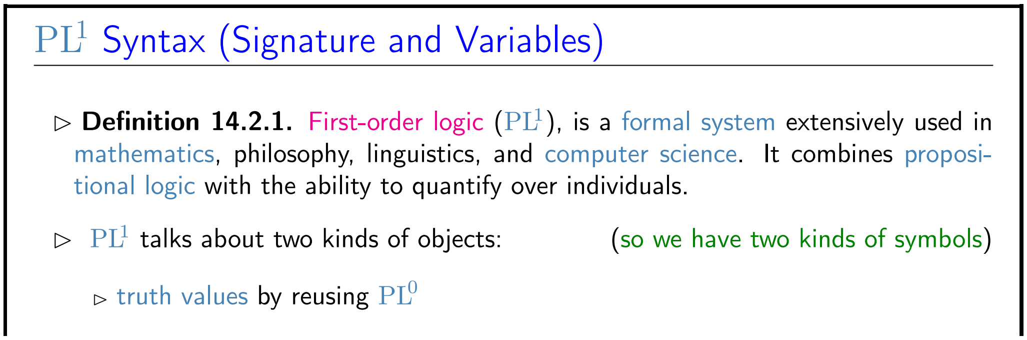
Michael Kohlhase: Artificial Intelligence 1 413 2025-02-06

14.2.1 First-Order Logic: Syntax and Semantics

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/25094`.

The syntax and semantics of first-order logic is systematically organized in two distinct layers: one for truth values (like in propositional logic) and one for individuals (the new, distinctive feature of first-order logic).

The first step of defining a formal language is to specify the alphabet, here the first-order signatures and their components.



PL^1 Syntax (Signature and Variables)

- **Definition 14.2.1.** First-order logic (PL^1), is a formal system extensively used in mathematics, philosophy, linguistics, and computer science. It combines propositional logic with the ability to quantify over individuals.
- PL^1 talks about two kinds of objects: (so we have two kinds of symbols)
  - truth values by reusing PL^0

- individuals, e.g. numbers, foxes, Pokémon,...

- **Definition 14.2.2.** A first-order signature consists of (all disjoint; $k \in \mathbb{N}$)
  - connectives: $\Sigma_0 = \{T, F, \neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, \ldots\}$ (functions on truth values)
  - function constants: $\Sigma^f_k = \{f, g, h, \ldots\}$ (k-ary functions on individuals)
  - predicate constants: $\Sigma^p{}_k = \{p, q, r, \ldots\}$ (k-ary relations among individuals.)
  - (Skolem constants: $\Sigma^{sk}_k = \{f^1_k, f^2_k, \ldots\}$) (witness constructors; countably ∞)
  - We take Σ_1 to be all of these together: $\Sigma_1 := \Sigma^f \cup \Sigma^p \cup \Sigma^{sk}$ and define $\Sigma := \Sigma_1 \cup \Sigma_0$.

- **Definition 14.2.3.** We assume a set of individual variables: $\mathcal{V}_\iota := \{X, Y, Z, \ldots\}$. (countably ∞)

We make the deliberate, but non-standard design choice here to include Skolem constants into the signature from the start. These are used in inference systems to give names to objects and construct witnesses. Other than the fact that they are usually introduced by need, they work exactly like regular constants, which makes the inclusion rather painless. As we can never predict how many Skolem constants we are going to need, we give ourselves countably infinitely many for every arity. Our supply of individual variables is countably infinite for the same reason.

The formulae of first-order logic are built up from the signature and variables as terms (to represent individuals) and proposition (to represent proposition). The latter include the connectives from PL^0, but also quantifiers.

PL^1 Syntax (Formulae)

- **Definition 14.2.4.** Terms: $\mathbf{A} \in \mathit{wff}_\iota(\Sigma_1, \mathcal{V}_\iota)$ (denote individuals)
  - $\mathcal{V}_\iota \subseteq \mathit{wff}_\iota(\Sigma_1, \mathcal{V}_\iota)$,
  - if $f \in \Sigma^f_k$ and $\mathbf{A}^i \in \mathit{wff}_\iota(\Sigma_1, \mathcal{V}_\iota)$ for $i \leq k$, then $f(\mathbf{A}^1, \ldots, \mathbf{A}^k) \in \mathit{wff}_\iota(\Sigma_1, \mathcal{V}_\iota)$.

- **Definition 14.2.5.** First-order propositions: $\mathbf{A} \in \mathit{wff}_o(\Sigma_1, \mathcal{V}_\iota)$: (denote truth values)
  - if $p \in \Sigma^p{}_k$ and $\mathbf{A}^i \in \mathit{wff}_\iota(\Sigma_1, \mathcal{V}_\iota)$ for $i \leq k$, then $p(\mathbf{A}^1, \ldots, \mathbf{A}^k) \in \mathit{wff}_o(\Sigma_1, \mathcal{V}_\iota)$,
  - if $\mathbf{A}, \mathbf{B} \in \mathit{wff}_o(\Sigma_1, \mathcal{V}_\iota)$ and $X \in \mathcal{V}_\iota$, then $T, \mathbf{A} \wedge \mathbf{B}, \neg \mathbf{A}, \forall X.\mathbf{A} \in \mathit{wff}_o(\Sigma_1, \mathcal{V}_\iota)$. $\forall$ is a binding operator called the universal quantifier.

- **Definition 14.2.6.** We define the connectives $F, \vee, \Rightarrow, \Leftrightarrow$ via the abbreviations $\mathbf{A} \vee \mathbf{B} := \neg(\neg \mathbf{A} \wedge \neg \mathbf{B})$, $\mathbf{A} \Rightarrow \mathbf{B} := \neg \mathbf{A} \vee \mathbf{B}$, $\mathbf{A} \Leftrightarrow \mathbf{B} := (\mathbf{A} \Rightarrow \mathbf{B}) \wedge (\mathbf{B} \Rightarrow \mathbf{A})$, and $F := \neg T$. We will use them like the primary connectives $\wedge$ and $\neg$

- **Definition 14.2.7.** We use $\exists X.\mathbf{A}$ as an abbreviation for $\neg(\forall X.\neg \mathbf{A})$. $\exists$ is a binding operator called the existential quantifier.

- **Definition 14.2.8.** Call formulae without connectives or quantifiers atomic else complex.

Note: We only need e.g. conjunction, negation, and universal quantifier, all other logical constants can be defined from them (as we will see when we have fixed their interpretations).

Alternative Notations for Quantifiers

Here	Elsewhere
$\forall x.\mathbf{A}$	$\bigwedge x.\mathbf{A}$ $\quad (x)\mathbf{A}$
$\exists x.\mathbf{A}$	$\bigvee x.\mathbf{A}$

FAU Michael Kohlhase: Artificial Intelligence 1 416 2025-02-06

The introduction of quantifiers to first-order logic brings a new phenomenon: variables that are under the scope of a quantifiers will behave very differently from the ones that are not. Therefore we build up a vocabulary that distinguishes the two.

Free and Bound Variables

- ▷ **Definition 14.2.9.** We call an occurrence of a variable X bound in a formula $\mathbf{A}$ (otherwise free), iff it occurs in a sub-formula $\forall X.\mathbf{B}$ of $\mathbf{A}$.

  For a formula $\mathbf{A}$, we will use $\mathrm{BVar}(\mathbf{A})$ (and $\mathrm{free}(\mathbf{A})$) for the set of bound (free) variables of $\mathbf{A}$, i.e. variables that have a free/bound occurrence in $\mathbf{A}$.

- ▷ **Definition 14.2.10.** We define the set $\mathrm{free}(\mathbf{A})$ of free variables of a formula $\mathbf{A}$:

$$\begin{array}{l}
\mathrm{free}(X) := \{X\}\\
\mathrm{free}(f(\mathbf{A}_1,\ldots,\mathbf{A}_n)) := \bigcup_{1\leq i\leq n}\mathrm{free}(\mathbf{A}_i)\\
\mathrm{free}(p(\mathbf{A}_1,\ldots,\mathbf{A}_n)) := \bigcup_{1\leq i\leq n}\mathrm{free}(\mathbf{A}_i)\\
\mathrm{free}(\neg\mathbf{A}) := \mathrm{free}(\mathbf{A})\\
\mathrm{free}(\mathbf{A}\wedge\mathbf{B}) := \mathrm{free}(\mathbf{A})\cup\mathrm{free}(\mathbf{B})\\
\mathrm{free}(\forall X.\mathbf{A}) := \mathrm{free}(\mathbf{A})\backslash\{X\}
\end{array}$$

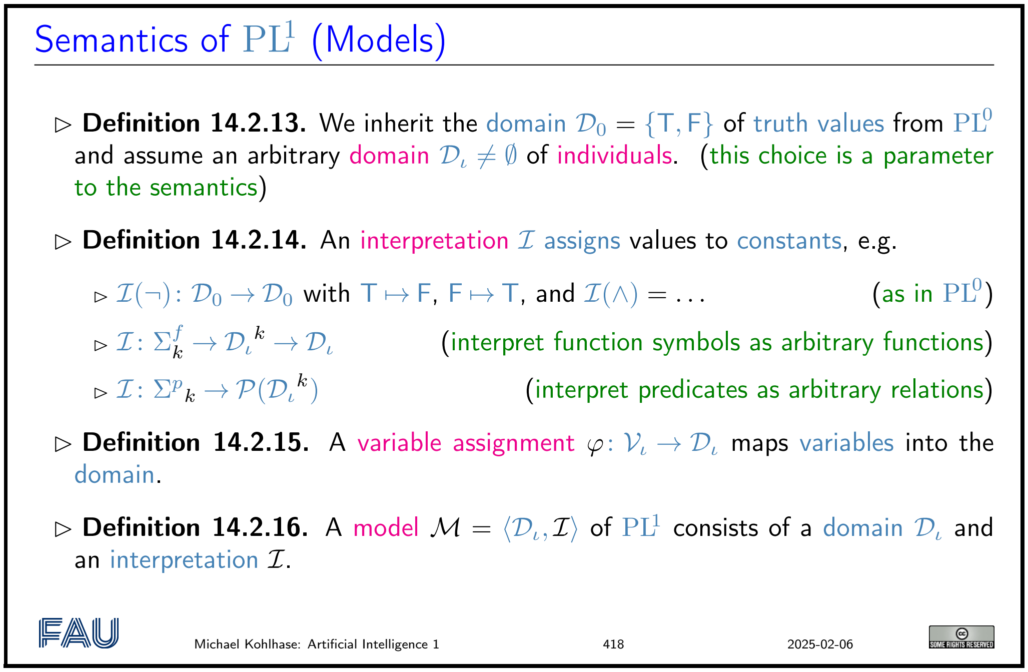
- ▷ **Definition 14.2.11.** We call a formula $\mathbf{A}$ closed or ground, iff $\mathrm{free}(\mathbf{A}) = \emptyset$. We call a closed proposition a sentence, and denote the set of all ground term with $\mathit{cwff}_\iota(\Sigma_\iota)$ and the set of sentences with $\mathit{cwff}_o(\Sigma_\iota)$.

- ▷ **Axiom 14.2.12.** *Bound variables can be renamed, i.e. any subterm $\forall X.\mathbf{B}$ of a formula $\mathbf{A}$ can be replaced by $\mathbf{A}' := (\forall Y.\mathbf{B}')$, where $\mathbf{B}'$ arises from $\mathbf{B}$ by replacing all $X \in \mathrm{free}(\mathbf{B})$ with a new variable Y that does not occur in $\mathbf{A}$. We call $\mathbf{A}'$ an alphabetical variant of $\mathbf{A}$ – and the other way around too.*

FAU Michael Kohlhase: Artificial Intelligence 1 417 2025-02-06

We will be mainly interested in (sets of) sentences – i.e. closed propositions – as the representations of meaningful statements about individuals. Indeed, we will see below that free variables do not gives us expressivity, since they behave like constants and could be replaced by them in all situations, except the recursive definition of quantified formulae. Indeed in all situations where variables occur freely, they have the character of metavariables, i.e. syntactic placeholders that can be instantiated with terms when needed in a calculus.

The semantics of first-order logic is a Tarski-style set-theoretic semantics where the atomic syntactic entities are interpreted by mapping them into a well-understood structure, a first-order universe that is just an arbitrary set.

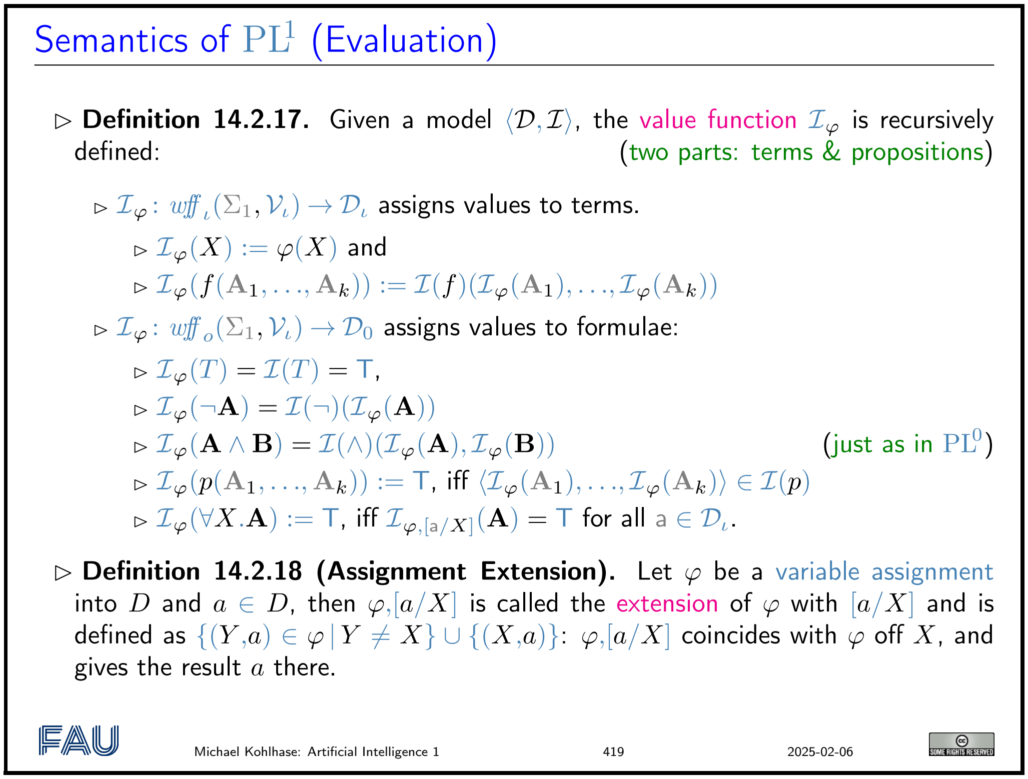


Semantics of PL^1 (Models)

- **Definition 14.2.13.** We inherit the domain $\mathcal{D}_0 = \{\mathsf{T}, \mathsf{F}\}$ of truth values from PL^0 and assume an arbitrary domain $\mathcal{D}_\iota \neq \emptyset$ of individuals. (this choice is a parameter to the semantics)
- **Definition 14.2.14.** An interpretation $\mathcal{I}$ assigns values to constants, e.g.
  - $\mathcal{I}(\neg)\colon \mathcal{D}_0 \to \mathcal{D}_0$ with $\mathsf{T} \mapsto \mathsf{F}$, $\mathsf{F} \mapsto \mathsf{T}$, and $\mathcal{I}(\wedge) = \ldots$ (as in PL^0)
  - $\mathcal{I}\colon \Sigma_k^f \to {\mathcal{D}_\iota}^k \to \mathcal{D}_\iota$ (interpret function symbols as arbitrary functions)
  - $\mathcal{I}\colon {\Sigma^p}_k \to \mathcal{P}({\mathcal{D}_\iota}^k)$ (interpret predicates as arbitrary relations)
- **Definition 14.2.15.** A variable assignment $\varphi\colon \mathcal{V}_\iota \to \mathcal{D}_\iota$ maps variables into the domain.
- **Definition 14.2.16.** A model $\mathcal{M} = \langle \mathcal{D}_\iota, \mathcal{I} \rangle$ of PL^1 consists of a domain $\mathcal{D}_\iota$ and an interpretation $\mathcal{I}$.

FAU Michael Kohlhase: Artificial Intelligence 1 418 2025-02-06 SOME RIGHTS RESERVED

We do not have to make the domain of truth values part of the model, since it is always the same; we determine the model by choosing a domain and an interpretation functiong.
Given a first-order model, we can define the evaluation function as a homomorphism over the construction of formulae.



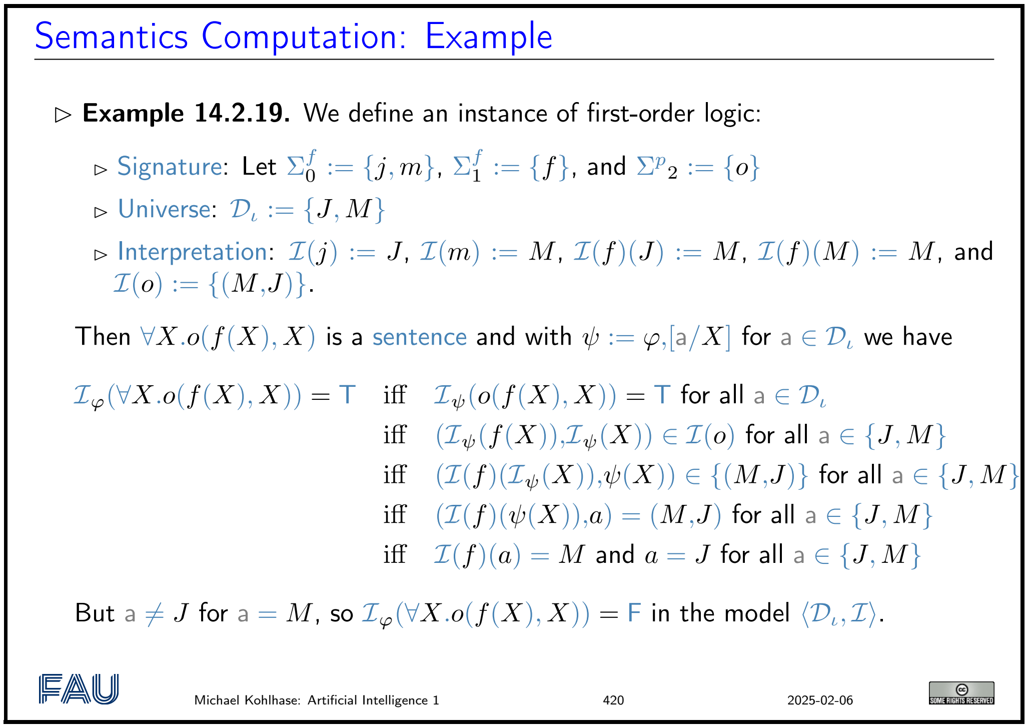
Semantics of PL^1 (Evaluation)

- **Definition 14.2.17.** Given a model $\langle \mathcal{D}, \mathcal{I} \rangle$, the value function $\mathcal{I}_\varphi$ is recursively defined: (two parts: terms & propositions)
  - $\mathcal{I}_\varphi\colon \mathit{wff}_\iota(\Sigma_1, \mathcal{V}_\iota) \to \mathcal{D}_\iota$ assigns values to terms.
    - $\mathcal{I}_\varphi(X) := \varphi(X)$ and
    - $\mathcal{I}_\varphi(f(\mathbf{A}_1, \ldots, \mathbf{A}_k)) := \mathcal{I}(f)(\mathcal{I}_\varphi(\mathbf{A}_1), \ldots, \mathcal{I}_\varphi(\mathbf{A}_k))$
  - $\mathcal{I}_\varphi\colon \mathit{wff}_o(\Sigma_1, \mathcal{V}_\iota) \to \mathcal{D}_0$ assigns values to formulae:
    - $\mathcal{I}_\varphi(T) = \mathcal{I}(T) = \mathsf{T}$,
    - $\mathcal{I}_\varphi(\neg \mathbf{A}) = \mathcal{I}(\neg)(\mathcal{I}_\varphi(\mathbf{A}))$
    - $\mathcal{I}_\varphi(\mathbf{A} \wedge \mathbf{B}) = \mathcal{I}(\wedge)(\mathcal{I}_\varphi(\mathbf{A}), \mathcal{I}_\varphi(\mathbf{B}))$ (just as in PL^0)
    - $\mathcal{I}_\varphi(p(\mathbf{A}_1, \ldots, \mathbf{A}_k)) := \mathsf{T}$, iff $\langle \mathcal{I}_\varphi(\mathbf{A}_1), \ldots, \mathcal{I}_\varphi(\mathbf{A}_k) \rangle \in \mathcal{I}(p)$
    - $\mathcal{I}_\varphi(\forall X.\mathbf{A}) := \mathsf{T}$, iff $\mathcal{I}_{\varphi,[\mathsf{a}/X]}(\mathbf{A}) = \mathsf{T}$ for all $\mathsf{a} \in \mathcal{D}_\iota$.
- **Definition 14.2.18 (Assignment Extension).** Let φ be a variable assignment into D and $a \in D$, then $\varphi,[a/X]$ is called the extension of φ with $[a/X]$ and is defined as $\{(Y,a) \in \varphi \,|\, Y \neq X\} \cup \{(X,a)\}$: $\varphi,[a/X]$ coincides with φ off X, and gives the result a there.

FAU Michael Kohlhase: Artificial Intelligence 1 419 2025-02-06 SOME RIGHTS RESERVED

The only new (and interesting) case in this definition is the quantifier case, there we define the value of a quantified formula by the value of its scope – *but with an extension of the incoming variable assignment*. Note that by passing to the scope $\mathbf{A}$ of $\forall x.\mathbf{A}$, the occurrences of the variable x in $\mathbf{A}$ that were bound in $\forall x.\mathbf{A}$ become free and are amenable to evaluation by the variable assignment $\psi := \varphi,[\mathsf{a}/X]$. Note that as an extension of φ, the assignment ψ supplies exactly the right value for x in $\mathbf{A}$. This variability of the variable assignment in the definition of the value function justifies the somewhat complex setup of first-order evaluation, where we have the (static) interpretation function for the symbols from the signature and the (dynamic) variable assignment for the variables.

Note furthermore, that the value $\mathcal{I}_\varphi(\exists x.\mathbf{A})$ of $\exists x.\mathbf{A}$, which we have defined to be $\neg(\forall x.\neg\mathbf{A})$ is true, iff it is not the case that $\mathcal{I}_\varphi(\forall x.\neg\mathbf{A}) = \mathcal{I}_\psi(\neg\mathbf{A}) = \mathsf{F}$ for all $\mathsf{a} \in \mathcal{D}_\iota$ and $\psi := \varphi,[\mathsf{a}/X]$. This is the case, iff $\mathcal{I}_\psi(\mathbf{A}) = \mathsf{T}$ for some $\mathsf{a} \in \mathcal{D}_\iota$. So our definition of the existential quantifier yields the appropriate semantics.



Semantics Computation: Example

- **Example 14.2.19.** We define an instance of first-order logic:
  - Signature: Let $\Sigma_0^f := \{j, m\}$, $\Sigma_1^f := \{f\}$, and $\Sigma^p{}_2 := \{o\}$
  - Universe: $\mathcal{D}_\iota := \{J, M\}$
  - Interpretation: $\mathcal{I}(j) := J$, $\mathcal{I}(m) := M$, $\mathcal{I}(f)(J) := M$, $\mathcal{I}(f)(M) := M$, and $\mathcal{I}(o) := \{(M,J)\}$.

  Then $\forall X.o(f(X), X)$ is a sentence and with $\psi := \varphi,[\mathsf{a}/X]$ for $\mathsf{a} \in \mathcal{D}_\iota$ we have

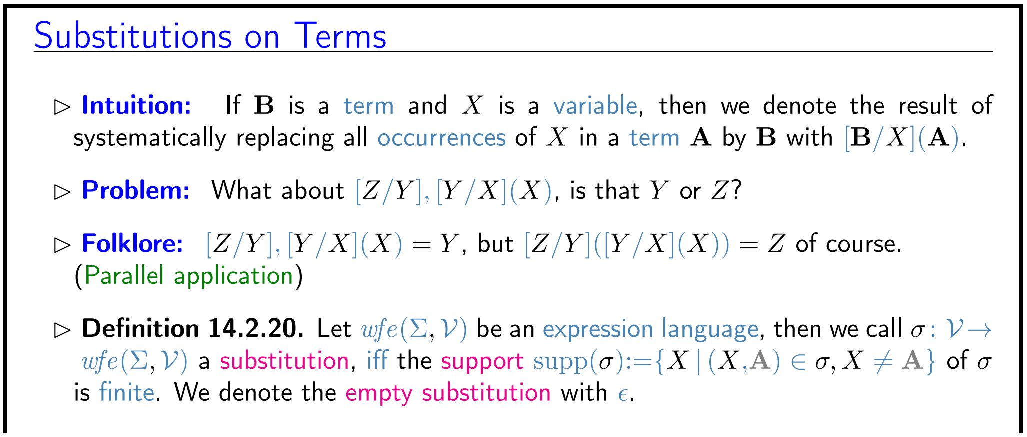
$$\begin{array}{lll}
\mathcal{I}_\varphi(\forall X.o(f(X), X)) = \mathsf{T} & \text{iff} & \mathcal{I}_\psi(o(f(X), X)) = \mathsf{T} \text{ for all } \mathsf{a} \in \mathcal{D}_\iota \\
 & \text{iff} & (\mathcal{I}_\psi(f(X)),\mathcal{I}_\psi(X)) \in \mathcal{I}(o) \text{ for all } \mathsf{a} \in \{J, M\} \\
 & \text{iff} & (\mathcal{I}(f)(\mathcal{I}_\psi(X)),\psi(X)) \in \{(M,J)\} \text{ for all } \mathsf{a} \in \{J, M\} \\
 & \text{iff} & (\mathcal{I}(f)(\psi(X)),a) = (M,J) \text{ for all } \mathsf{a} \in \{J, M\} \\
 & \text{iff} & \mathcal{I}(f)(a) = M \text{ and } a = J \text{ for all } \mathsf{a} \in \{J, M\}
\end{array}$$

  But $\mathsf{a} \neq J$ for $\mathsf{a} = M$, so $\mathcal{I}_\varphi(\forall X.o(f(X), X)) = \mathsf{F}$ in the model $\langle\mathcal{D}_\iota,\mathcal{I}\rangle$.

14.2.2 First-Order Substitutions

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/25156`.

We will now turn our attention to substitutions, special formula-to-formula mappings that operationalize the intuition that (individual) variables stand for arbitrary terms.



Substitutions on Terms

- **Intuition:** If $\mathbf{B}$ is a term and X is a variable, then we denote the result of systematically replacing all occurrences of X in a term $\mathbf{A}$ by $\mathbf{B}$ with $[\mathbf{B}/X](\mathbf{A})$.
- **Problem:** What about $[Z/Y], [Y/X](X)$, is that Y or Z?
- **Folklore:** $[Z/Y], [Y/X](X) = Y$, but $[Z/Y]([Y/X](X)) = Z$ of course. (Parallel application)
- **Definition 14.2.20.** Let $wfe(\Sigma, \mathcal{V})$ be an expression language, then we call $\sigma\colon \mathcal{V} \to wfe(\Sigma, \mathcal{V})$ a substitution, iff the support $\operatorname{supp}(\sigma){:=}\{X \mid (X,\mathbf{A}) \in \sigma, X \neq \mathbf{A}\}$ of σ is finite. We denote the empty substitution with ϵ.

- **Definition 14.2.21 (Substitution Application).** We define substitution application by
  - $\sigma(c) = c$ for $c \in \Sigma$
  - $\sigma(X) = \mathbf{A}$, iff $X \in \mathcal{V}$ and $(X,\mathbf{A}) \in \sigma$.
  - $\sigma(f(\mathbf{A}_1,\ldots,\mathbf{A}_n)) = f(\sigma(\mathbf{A}_1),\ldots,\sigma(\mathbf{A}_n))$,
  - $\sigma(\forall X.\mathbf{A}) = \forall X.\sigma_{-X}(\mathbf{A})$. (∃ analogous)
- **Example 14.2.22.** $[a/x], [f(b)/y], [a/z]$ instantiates $g(x,y,h(z))$ to $g(a,f(b),h(a))$.

The extension of a substitution is an important operation, which you will run into from time to time. Given a substitution σ, a variable x, and an expression $\mathbf{A}$, $\sigma,[\mathbf{A}/x]$ extends σ with a new value for x. The intuition is that the values right of the comma overwrite the pairs in the substitution on the left, which already has a value for x, even though the representation of σ may not show it.

Substitution Extension

- **Definition 14.2.23 (Substitution Extension).** Let σ be a substitution, then we denote the extension of σ with $[\mathbf{A}/X]$ by $\sigma,[\mathbf{A}/X]$ and define it as $\{(Y,\mathbf{B}) \in \sigma \,|\, Y \neq X\} \cup \{(X,\mathbf{A})\}$: $\sigma,[\mathbf{A}/X]$ coincides with σ off X, and gives the result $\mathbf{A}$ there.
- **Note:** If σ is a substitution, then $\sigma,[\mathbf{A}/X]$ is also a substitution.
- We also need the dual operation: removing a variable from the support:
- **Definition 14.2.24.** We can discharge a variable X from a substitution σ by setting $\sigma_{-X}{:=}\sigma,[X/X]$.

Note that the use of the comma notation for substitutions defined in **??** is consistent with substitution extension. We can view a substitution $[a/x], [f(b)/y]$ as the extension of the empty substitution (the identity function on variables) by $[f(b)/y]$ and then by $[a/x]$. Note furthermore, that substitution extension is not commutative in general.
For first-order substitutions we need to extend the substitutions defined on terms to act on propositions. This is technically more involved, since we have to take care of bound variables.

Substitutions on Propositions

- **Problem:** We want to extend substitutions to propositions, in particular to quantified formulae: What is $\sigma(\forall X.\mathbf{A})$?
- **Idea:** σ should not instantiate bound variables. ($[\mathbf{A}/X](\forall X.\mathbf{B}) = \forall \mathbf{A}.\mathbf{B}'$ ill-formed)
- **Definition 14.2.25.** $\sigma(\forall X.\mathbf{A}) := (\forall X.\sigma_{-X}(\mathbf{A}))$.
- **Problem:** This can lead to variable capture: $[f(X)/Y](\forall X.p(X,Y))$ would evaluate to $\forall X.p(X,f(X))$, where the second occurrence of X is bound after instanti-

ation, whereas it was free before. **Solution:** Rename away the bound variable X in $\forall X.p(X,Y)$ before applying the substitution.

▷ **Definition 14.2.26 (Capture-Avoiding Substitution Application).** Let σ be a substitution, $\mathbf{A}$ a formula, and $\mathbf{A}'$ an alphabetic variant of $\mathbf{A}$, such that $\mathrm{intro}(\sigma) \cap \mathrm{BVar}(\mathbf{A}) = \emptyset$. Then we define capture-avoiding substitution application via $\sigma(\mathbf{A}) := \sigma(\mathbf{A}')$.

We now introduce a central tool for reasoning about the semantics of substitutions: the “substitution value Lemma”, which relates the process of instantiation to (semantic) evaluation. This result will be the motor of all soundness proofs on axioms and inference rules acting on variables via substitutions. In fact, any logic with variables and substitutions will have (to have) some form of a substitution value Lemma to get the meta-theory going, so it is usually the first target in any development of such a logic. We establish the substitution-value Lemma for first-order logic in two steps, first on terms, where it is very simple, and then on propositions.

Substitution Value Lemma for Terms

▷ **Lemma 14.2.27.** *Let $\mathbf{A}$ and $\mathbf{B}$ be terms, then $\mathcal{I}_\varphi([\mathbf{B}/X]\mathbf{A}) = \mathcal{I}_\psi(\mathbf{A})$, where $\psi = \varphi,[\mathcal{I}_\varphi(\mathbf{B})/X]$.*

▷ *Proof:* by induction on the depth of $\mathbf{A}$:

1. depth=0 *Then $\mathbf{A}$ is a variable (say Y), or constant, so we have three cases*
    1.1. $\mathbf{A} = Y = X$
        1.1.1. then $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_\varphi([\mathbf{B}/X](X)) = \mathcal{I}_\varphi(\mathbf{B}) = \psi(X) = \mathcal{I}_\psi(X) = \mathcal{I}_\psi(\mathbf{A})$.
    1.2. $\mathbf{A} = Y \neq X$
        1.2.1. then $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_\varphi([\mathbf{B}/X](Y)) = \mathcal{I}_\varphi(Y) = \varphi(Y) = \psi(Y) = \mathcal{I}_\psi(Y) = \mathcal{I}_\psi(\mathbf{A})$.
    1.3. $\mathbf{A}$ is a constant
        1.3.1. Analogous to the preceding case ($Y \neq X$).
    1.4. This completes the base case (depth = 0).
2. depth> 0
    2.1. then $\mathbf{A} = f(\mathbf{A}_1,\ldots,\mathbf{A}_n)$ and we have

$$\begin{aligned}\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) &= \mathcal{I}(f)(\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A}_1)),\ldots,\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A}_n)))\\ &= \mathcal{I}(f)(\mathcal{I}_\psi(\mathbf{A}_1),\ldots,\mathcal{I}_\psi(\mathbf{A}_n))\\ &= \mathcal{I}_\psi(\mathbf{A}).\end{aligned}$$

    by induction hypothesis
    2.2. This completes the induction step, and we have proven the assertion.

Substitution Value Lemma for Propositions

▷ **Lemma 14.2.28.** $\mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A})) = \mathcal{I}_\psi(\mathbf{A})$*, where $\psi = \varphi,[\mathcal{I}_\varphi(\mathbf{B})/X]$.*

▷ *Proof:* by induction on the number n of connectives and quantifiers in $\mathbf{A}$:

1. $n = 0$
   1.1. then $\mathbf{A}$ is an atomic proposition, and we can argue like in the induction step of the substitution value lemma for terms.
2. $n > 0$ and $\mathbf{A} = \neg\mathbf{B}$ or $\mathbf{A} = \mathbf{C} \circ \mathbf{D}$
   2.1. Here we argue like in the induction step of the term lemma as well.
3. $n > 0$ and $\mathbf{A} = \forall Y.\mathbf{C}$ where (WLOG) $X \neq Y$ *(otherwise rename)*
   3.1. then $\mathcal{I}_\psi(\mathbf{A}) = \mathcal{I}_\psi(\forall Y.\mathbf{C}) = \mathsf{T}$, iff $\mathcal{I}_{\psi,[a/Y]}(\mathbf{C}) = \mathsf{T}$ for all $a \in \mathcal{D}_\iota$.
   3.2. But $\mathcal{I}_{\psi,[a/Y]}(\mathbf{C}) = \mathcal{I}_{\varphi,[a/Y]}([\mathbf{B}/X](\mathbf{C})) = \mathsf{T}$, by induction hypothesis.
   3.3. So $\mathcal{I}_\psi(\mathbf{A}) = \mathcal{I}_\varphi(\forall Y.[\mathbf{B}/X](\mathbf{C})) = \mathcal{I}_\varphi([\mathbf{B}/X](\forall Y.\mathbf{C})) = \mathcal{I}_\varphi([\mathbf{B}/X](\mathbf{A}))$

To understand the proof fully, you should think about where the *WLOG* – it stands for without loss of generality comes from.

14.3 First-Order Natural Deduction

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/25157`.

In this section, we will introduce the first-order natural deduction calculus. Recall from **??** that natural deduction calculus have introduction and elimination for every logical constant (the connectives in PL^0). Recall furthermore that we had two styles/notations for the calculus, the classical ND calculus and the sequent-style notation. These principles will be carried over to natural deduction in PL^1.

This allows us to introduce the calculi in two stages, first for the (propositional) connectives and then extend this to a calculus for first-order logic by adding rules for the quantifiers. In particular, we can define the first-order calculi simply by adding (introduction and elimination) rules for the (universal and existential) quantifiers to the calculus $\mathcal{ND}_0$ defined in **??**.

To obtain a first-order calculus, we have to extend $\mathcal{ND}_0$ with (introduction and elimination) rules for the quantifiers.

First-Order Natural Deduction ($\mathcal{ND}^1$; Gentzen [Gen34])

▷ Rules for connectives just as always

▷ **Definition 14.3.1 (New Quantifier Rules).** The first-order natural deduction calculus $\mathcal{ND}^1$ extends $\mathcal{ND}_0$ by the following four rules:

$$\frac{\mathbf{A}}{\forall X.\mathbf{A}}\;\mathcal{ND}^1\forall I^* \qquad\qquad \frac{\forall X.\mathbf{A}}{[\mathbf{B}/X](\mathbf{A})}\;\mathcal{ND}^1\forall E$$

$$\frac{[\mathbf{B}/X](\mathbf{A})}{\exists X.\mathbf{A}}\;\mathcal{ND}^1\exists I \qquad\qquad \frac{\exists X.\mathbf{A} \qquad \begin{array}{c}[[c/X](\mathbf{A})]^1\\ \vdots\\ \mathbf{C}\end{array} \qquad c \in \Sigma_0^{sk}\ \text{new}}{\mathbf{C}}\;\mathcal{ND}^1\exists E^1$$

* means that $\mathbf{A}$ does not depend on any hypothesis in which X is free.

The intuition behind the rule $\mathcal{ND}^1\forall I$ is that a formula $\mathbf{A}$ with a (free) variable X can be generalized to $\forall X.\mathbf{A}$, if X stands for an arbitrary object, i.e. there are no restricting assumptions about X. The $\mathcal{ND}^1\forall E$ rule is just a substitution rule that allows to instantiate arbitrary terms $\mathbf{B}$ for X

in $\mathbf{A}$. The $\mathcal{ND}^1\exists I$ rule says if we have a witness $\mathbf{B}$ for X in $\mathbf{A}$ (i.e. a concrete term $\mathbf{B}$ that makes $\mathbf{A}$ true), then we can existentially close $\mathbf{A}$. The $\mathcal{ND}^1\exists E$ rule corresponds to the common mathematical practice, where we give objects we know exist a new name c and continue the proof by reasoning about this concrete object c. Anything we can prove from the assumption $[c/X](\mathbf{A})$ we can prove outright if $\exists X.\mathbf{A}$ is known.

Now we reformulate the classical formulation of the calculus of natural deduction as a sequent calculus by lifting it to the "judgments level" as we did for propositional logic. We only need provide new quantifier rules.

First-Order Natural Deduction in Sequent Formulation

- ▷ Rules for connectives from $\mathcal{ND}^0_\vdash$
- ▷ **Definition 14.3.2 (New Quantifier Rules).** The inference rules of the first-order sequent calculus $\mathcal{ND}^1_\vdash$ consist of those from $\mathcal{ND}^0_\vdash$ plus the following quantifier rules:

$$\frac{\Gamma\vdash\mathbf{A} \quad X\notin\mathrm{free}(\Gamma)}{\Gamma\vdash\forall X.\mathbf{A}}\,\mathcal{ND}^1_\vdash\forall I \qquad \frac{\Gamma\vdash\forall X.\mathbf{A}}{\Gamma\vdash[\mathbf{B}/X](\mathbf{A})}\,\mathcal{ND}^1_\vdash\forall E$$

$$\frac{\Gamma\vdash[\mathbf{B}/X](\mathbf{A})}{\Gamma\vdash\exists X.\mathbf{A}}\,\mathcal{ND}^1_\vdash\exists I \qquad \frac{\Gamma\vdash\exists X.\mathbf{A} \quad \Gamma,[c/X](\mathbf{A})\vdash\mathbf{C} \quad c\in\Sigma_0^{sk}\text{ new}}{\Gamma\vdash\mathbf{C}}\,\mathcal{ND}^1_\vdash\exists E$$

FAU Michael Kohlhase: Artificial Intelligence 1 427 2025-02-06

Natural Deduction with Equality

- ▷ **Definition 14.3.3 (First-Order Logic with Equality).** We extend PL^1 with a new logical constant for equality $= \in \Sigma^p{}_2$ and fix its interpretation to $\mathcal{I}(=) := \{(x,x)\,|\,x\in\mathcal{D}_\iota\}$. We call the extended logic first-order logic with equality ($\mathrm{PL}^1_=$)
- ▷ We now extend natural deduction as well.
- ▷ **Definition 14.3.4.** For the calculus of natural deduction with equality ($\mathcal{ND}^1_=$) we add the following two rules to $\mathcal{ND}^1$ to deal with equality:

$$\frac{}{\mathbf{A}=\mathbf{A}}\,{=}I \qquad \frac{\mathbf{A}=\mathbf{B} \quad \mathbf{C}\,[\mathbf{A}]_p}{[\mathbf{B}/p]\mathbf{C}}\,{=}E$$

where $\mathbf{C}\,[\mathbf{A}]_p$ if the formula $\mathbf{C}$ has a subterm $\mathbf{A}$ at position p and $[\mathbf{B}/p]\mathbf{C}$ is the result of replacing that subterm with $\mathbf{B}$.

- ▷ In many ways equivalence behaves like equality, we will use the following rules in $\mathcal{ND}^1$
- ▷ **Definition 14.3.5.** $\Leftrightarrow I$ is derivable and $\Leftrightarrow E$ is admissible in $\mathcal{ND}^1$:

$$\frac{}{\mathbf{A}\Leftrightarrow\mathbf{A}}\,\Leftrightarrow I \qquad \frac{\mathbf{A}\Leftrightarrow\mathbf{B} \quad \mathbf{C}\,[\mathbf{A}]_p}{[\mathbf{B}/p]\mathbf{C}}\,\Leftrightarrow E$$

FAU Michael Kohlhase: Artificial Intelligence 1 428 2025-02-06

Again, we have two rules that follow the introduction/elimination pattern of natural deduction calculi.

Definition 14.3.6. We have the canonical sequent rules that correspond to them: $=I$, $=E$, $\Leftrightarrow I$, and $\Leftrightarrow E$

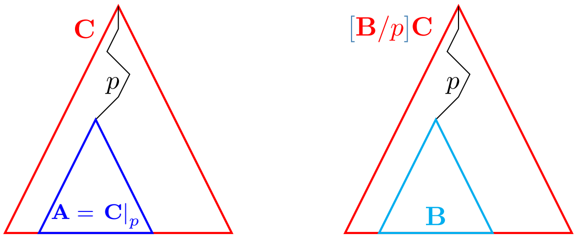
To make sure that we understand the constructions here, let us get back to the "replacement at position" operation used in the equality rules.

Positions in Formulae

- ▷ **Idea:** Formulae are (naturally) trees, so we can use tree positions to talk about subformulae
- ▷ **Definition 14.3.7.** A position p is a tuple of natural numbers that in each node of an expression (tree) specifies into which child to descend. For an expression $\mathbf{A}$ we denote the subexpression at p with $\mathbf{A}|_p$.

  We will sometimes write an expression $\mathbf{C}$ as $\mathbf{C}[\mathbf{A}]_p$ to indicate that $\mathbf{C}$ the subexpression $\mathbf{A}$ at position p.

  If $\mathbf{C}[\mathbf{A}]_p$ and $\mathbf{A}$ is atomic, then we speak of an occurrence of $\mathbf{A}$ in $\mathbf{C}$.
- ▷ **Definition 14.3.8.** Let p be a position, then $[\mathbf{A}/p]\mathbf{C}$ is the expression obtained from $\mathbf{C}$ by replacing the subexpression at p by $\mathbf{A}$.
- ▷ **Example 14.3.9 (Schematically).**



The operation of replacing a subformula at position p is quite different from e.g. (first-order) substitutions:

- We are replacing subformulae with subformulae instead of instantiating variables with terms.
- Substitutions replace all occurrences of a variable in a formula, whereas formula replacement only affects the (one) subformula at position p.

We conclude this section with an extended example: the proof of a classical mathematical result in the natural deduction calculus with equality. This shows us that we can derive strong properties about complex situations (here the real numbers; an uncountably infinite set of numbers).

$\mathcal{ND}^1_=$ Example: $\sqrt{2}$ is Irrational

- ▷ We can do real mathematics with $\mathcal{ND}^1_=$:
- ▷ **Theorem 14.3.10.** *$\sqrt{2}$ is irrational*

  Proof: We prove the assertion by contradiction

  1. Assume that $\sqrt{2}$ is rational.
  2. Then there are numbers p and q such that $\sqrt{2} = p/q$.

3. So we know $2q^2 = p^2$.
4. But $2q^2$ has an odd number of prime factors while p^2 an even number.
5. This is a contradiction (since they are equal), so we have proven the assertion

If we want to formalize this into $\mathcal{ND}^1$, we have to write down all the assertions in the proof steps in PL¹ syntax and come up with justifications for them in terms of $\mathcal{ND}^1$ inference rules. The next two slides show such a proof, where we write $\prime n$ to denote that n is prime, use $\#(n)$ for the number of prime factors of a number n, and write $\mathrm{irr}(r)$ if r is irrational.

$\mathcal{ND}^1_=$ Example: $\sqrt{2}$ is Irrational (the Proof)

#	hyp	formula	NDjust
1		$\forall n, m.\neg(2n+1) = (2m)$	lemma
2		$\forall n, m.\#(n^m) = m\#(n)$	lemma
3		$\forall n, p.\mathrm{prime}(p) \Rightarrow \#(pn) = (\#(n)+1)$	lemma
4		$\forall x.\mathrm{irr}(x) \Leftrightarrow \neg(\exists p, q.x = p/q)$	definition
5		$\mathrm{irr}(\sqrt{2}) \Leftrightarrow \neg(\exists p, q.\sqrt{2} = p/q)$	$\mathcal{ND}^1_\vdash \forall E(4)$
6	6	$\neg\mathrm{irr}(\sqrt{2})$	$\mathcal{ND}^0_\vdash \mathrm{Ax}$
7	6	$\neg\neg(\exists p, q.\sqrt{2} = p/q)$	$\Leftrightarrow E(6,5)$
8	6	$\exists p, q.\sqrt{2} = p/q$	$\mathcal{ND}^0_\vdash \neg E(7)$
9	6,9	$\sqrt{2} = p/q$	$\mathcal{ND}^0_\vdash \mathrm{Ax}$
10	6,9	$2q^2 = p^2$	arith(9)
11	6,9	$\#(p^2) = 2\#(p)$	$\mathcal{ND}^1_\vdash \forall E^2(2)$
12	6,9	$\mathrm{prime}(2) \Rightarrow \#(2q^2) = (\#(q^2)+1)$	$\mathcal{ND}^1_\vdash \forall E^2(1)$

Lines 6 and 9 are local hypotheses for the proof (they only have an implicit counterpart in the inference rules as defined above). Finally we have abbreviated the arithmetic simplification of line 9 with the justification "arith" to avoid having to formalize elementary arithmetic.

$\mathcal{ND}^1_=$ Example: $\sqrt{2}$ is Irrational (the Proof continued)

13		$\mathrm{prime}(2)$	lemma
14	6,9	$\#(2q^2) = \#(q^2)+1$	$\mathcal{ND}_0 \Rightarrow E(13,12)$
15	6,9	$\#(q^2) = 2\#(q)$	$\mathcal{ND}^1 \forall E^2(2)$
16	6,9	$\#(2q^2) = 2\#(q)+1$	$=E(14,15)$
17		$\#(p^2) = \#(p^2)$	$=I$
18	6,9	$\#(2q^2) = \#(q^2)$	$=E(17,10)$
19	6.9	$2\#(q)+1 = \#(p^2)$	$=E(18,16)$
20	6.9	$2\#(q)+1 = 2\#(p)$	$=E(19,11)$
21	6.9	$\neg(2\#(q)+1) = (2\#(p))$	$\mathcal{ND}^1 \forall E^2(1)$
22	6,9	F	$\mathcal{ND}_0 FI(20,21)$
23	6	F	$\mathcal{ND}^1 \exists E^6(22)$
24		$\neg\neg\mathrm{irr}(\sqrt{2})$	$\mathcal{ND}_0 \neg I^6(23)$
25		$\mathrm{irr}(\sqrt{2})$	$\mathcal{ND}_0 \neg E^2(23)$

FAU Michael Kohlhase: Artificial Intelligence 1 432 2025-02-06

We observe that the $\mathcal{ND}^1$ proof is much more detailed, and needs quite a few Lemmata about # to go through. Furthermore, we have added a definition of irrationality (and treat definitional equality via the equality rules). Apart from these artefacts of formalization, the two representations of proofs correspond to each other very directly.

14.4 Conclusion

Summary (Predicate Logic)

- ▷ First-order logic allows to explicitly speak about objects and their properties. It is thus a more natural and compact representation language than propositional logic; it also enables us to speak about infinite sets of objects.
- ▷ Logic has thousands of years of history. A major current application in AI is *semantic technology*. (up soon)
- ▷ First-order logic (PL^1) allows universal and existential quantifier quantification over individuals.
- ▷ A PL^1 model consists of a universe $\mathcal{D}_\iota$ and a function $\mathcal{I}$ mapping individual constants/predicate constants/function constants to elements/relations/functions on $\mathcal{D}_\iota$.
- ▷ First-order natural deduction is a sound and complete calculus for PL^1 intended and optimized for human understanding.

FAU Michael Kohlhase: Artificial Intelligence 1 433 2025-02-06

Applications for $\mathcal{ND}^1$ (and extensions)

- ▷ **Recap:** We can express mathematical theorems in PL^1 and prove them in $\mathcal{ND}^1$.
- ▷ **Problem:** These proofs can be huge (giga-steps), how can we trust them?
- ▷ **Definition 14.4.1.** A proof checker for a calculus $\mathcal{C}$ is a program that reads (a formal representation) of a $\mathcal{C}$-proof $\mathcal{P}$ and performs proof-checking, i.e. it checks whether all rule applications in $\mathcal{P}$ are (syntactically) correct.
- ▷ **Remark:** Proof-checking goes step-by-step $\leadsto$ proof checkers run in linear time.
- ▷ **Idea:** If the logic can express (safety)-properties of programs, we can use proof checkers for formal program verification. (there are extensions of PL^1 that can)
- ▷ **Problem:** These proofs can be humongous, how can humans write them?
- ▷ **Idea:** Automate proof construction via
  - ▷ lemma/theorem libraries that collect useful intermediate results
  - ▷ tactics $\widehat{=}$ subroutines that construct recurring sub-proofs
  - ▷ calls to automated theorem prover (ATP) (next chapter)

Proof checkers that do any/all of these are called proof assistants.

- ▷ **Definition 14.4.2.** Formal methods are logic-based techniques for the specification, development, analysis, and verification of software and hardware.
- ▷ Formal methods is a major (industrial) application of AI/logic technology.

Suggested Reading:

- *Chapter 8: First-Order Logic*, Sections 8.1 and 8.2 in [RN09]
  - A less formal account of what I cover in “Syntax” and “Semantics”. Contains different examples, and complementary explanations. Nice as additional background reading.
- Sections 8.3 and 8.4 provide additional material on using PL1, and on modeling in PL1, that I don’t cover in this lecture. Nice reading, not required for exam.
- *Chapter 9: Inference in First-Order Logic*, Section 9.5.1 in [RN09]
  - A very brief (2 pages) description of what I cover in “Normal Forms”. Much less formal; I couldn’t find where (if at all) RN cover transformation into prenex normal form. Can serve as additional reading, can’t replace the lecture.
- **Excursion:** A full analysis of any calculus needs a completeness proof. We will not cover this in AI-1, but provide one for the calculi introduced so far in**??**.

Chapter 15

Automated Theorem Proving in First-Order Logic

In this chapter, we take up the machine-oriented calculi for propositional logic from **??** and extend them to the first-order case. While this has been relatively easy for the natural deduction calculus – we only had to introduce the notion of substitutions for the elimination rule for the universal quantifier we have to work much more here to make the calculi effective for implementation.

15.1 First-Order Inference with Tableaux

15.1.1 First-Order Tableau Calculi

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/25156`.

Test Calculi: Tableaux and Model Generation

- ▷ **Idea:** A tableau calculus is a test calculus that
  - ▷ analyzes a labeled formulae in a tree to determine satisfiability,
  - ▷ its branches correspond to valuations (⇝ models).
- ▷ **Example 15.1.1.** Tableau calculi try to construct models for labeled formulae:

Tableau refutation (Validity)	Model generation (Satisfiability)
$\vDash P \wedge Q \Rightarrow Q \wedge P$	$\vDash P \wedge (Q \vee \neg R) \wedge \neg Q$
$\begin{array}{c} (P \wedge Q \Rightarrow Q \wedge P)^{\mathsf{F}} \\ (P \wedge Q)^{\mathsf{T}} \\ (Q \wedge P)^{\mathsf{F}} \\ P^{\mathsf{T}} \\ Q^{\mathsf{T}} \\ \begin{array}{c\|c} P^{\mathsf{F}} & Q^{\mathsf{F}} \\ \bot & \bot \end{array} \end{array}$	$\begin{array}{c} (P \wedge (Q \vee \neg R) \wedge \neg Q)^{\mathsf{T}} \\ (P \wedge (Q \vee \neg R))^{\mathsf{T}} \\ \neg Q^{\mathsf{T}} \\ Q^{\mathsf{F}} \\ P^{\mathsf{T}} \\ (Q \vee \neg R)^{\mathsf{T}} \\ \begin{array}{c\|c} Q^{\mathsf{T}} & \neg R^{\mathsf{T}} \\ \bot & R^{\mathsf{F}} \end{array} \end{array}$
No Model	Herbrand model $\{P^{\mathsf{T}}, Q^{\mathsf{F}}, R^{\mathsf{F}}\}$ $\varphi := \{P \mapsto \mathsf{T}, Q \mapsto \mathsf{F}, R \mapsto \mathsf{F}\}$

- ▷ **Idea:** Open branches in saturated tableaux yield models.
- ▷ **Algorithm:** Fully expand all possible tableaux, (no rule can be applied)

- Satisfiable, iff there are open branches (correspond to models)

Tableau calculi develop a formula in a tree-shaped arrangement that represents a case analysis on when a formula can be made true (or false). Therefore the formulae are decorated with upper indices that hold the intended truth value.

On the left we have a refutation tableau that analyzes a negated formula (it is decorated with the intended truth value F). Both branches contain an elementary contradiction ⊥.

On the right we have a model generation tableau, which analyzes a positive formula (it is decorated with the intended truth value T). This tableau uses the same rules as the refutation tableau, but makes a case analysis of when this formula can be satisfied. In this case we have a closed branch and an open one. The latter corresponds a model.

Now that we have seen the examples, we can write down the tableau rules formally.

Analytical Tableaux (Formal Treatment of $\mathcal{T}_0$)

- **Idea:** A test calculus where
  - A labeled formula is analyzed in a tree to determine satisfiability,
  - branches correspond to valuations (models)
- **Definition 15.1.2.** The propositional tableau calculus $\mathcal{T}_0$ has two inference rules per connective (one for each possible label)

$$\frac{(\mathbf{A}\wedge\mathbf{B})^{\mathsf{T}}}{\begin{array}{c}\mathbf{A}^{\mathsf{T}}\\ \mathbf{B}^{\mathsf{T}}\end{array}}\,\mathcal{T}_0\wedge \qquad \frac{(\mathbf{A}\wedge\mathbf{B})^{\mathsf{F}}}{\mathbf{A}^{\mathsf{F}}\;\big|\;\mathbf{B}^{\mathsf{F}}}\,\mathcal{T}_0\vee \qquad \frac{\neg\mathbf{A}^{\mathsf{T}}}{\mathbf{A}^{\mathsf{F}}}\,\mathcal{T}_0\neg^{\mathsf{T}} \qquad \frac{\neg\mathbf{A}^{\mathsf{F}}}{\mathbf{A}^{\mathsf{T}}}\,\mathcal{T}_0\neg^{\mathsf{F}} \qquad \frac{\begin{array}{c}\mathbf{A}^{\alpha}\\ \mathbf{A}^{\beta}\end{array}\quad \alpha\neq\beta}{\perp}\,\mathcal{T}_0\perp$$

  Use rules exhaustively as long as they contribute new material (⇝ termination)
- **Definition 15.1.3.** We call any tree (| introduces branches) produced by the $\mathcal{T}_0$ inference rules from a set Φ of labeled formulae a tableau for Φ.
- **Definition 15.1.4.** Call a tableau saturated, iff no rule adds new material and a branch closed, iff it ends in ⊥, else open. A tableau is closed, iff all of its branches are.

  In analogy to the ⊥ at the end of closed branches, we sometimes decorate open branches with a □ symbol.

These inference rules act on tableaux have to be read as follows: if the formulae over the line appear in a tableau branch, then the branch can be extended by the formulae or branches below the line. There are two rules for each primary connective, and a branch closing rule that adds the special symbol ⊥ (for unsatisfiability) to a branch.

We use the tableau rules with the convention that they are only applied, if they contribute new material to the branch. This ensures termination of the tableau procedure for propositional logic (every rule eliminates one primary connective).

Definition 15.1.5. We will call a closed tableau with the labeled formula $\mathbf{A}^{\alpha}$ at the root a tableau refutation for $\mathcal{A}^{\alpha}$.

The saturated tableau represents a full case analysis of what is necessary to give $\mathbf{A}$ the truth value α; since all branches are closed (contain contradictions) this is impossible.

Analytical Tableaux ($\mathcal{T}_0$ continued)

▷ **Definition 15.1.6 ($\mathcal{T}_0$-Theorem/Derivability).** $\mathbf{A}$ is a $\mathcal{T}_0$-theorem ($\vdash_{\mathcal{T}_0}\mathbf{A}$), iff there is a closed tableau with $\mathbf{A}^{\mathsf{F}}$ at the root.

$\Phi \subseteq \mathit{wff}_0(\mathcal{V}_0)$ derives $\mathbf{A}$ in $\mathcal{T}_0$ ($\Phi\vdash_{\mathcal{T}_0}\mathbf{A}$), iff there is a closed tableau starting with $\mathbf{A}^{\mathsf{F}}$ and Φ^{T}. The tableau with only a branch of $\mathbf{A}^{\mathsf{F}}$ and Φ^{T} is called initial for $\Phi\vdash_{\mathcal{T}_0}\mathbf{A}$.

FAU Michael Kohlhase: Artificial Intelligence 1 437 2025-02-06

Definition 15.1.7. We will call a tableau refutation for $\mathbf{A}^{\mathsf{F}}$ a tableau proof for $\mathbf{A}$, since it refutes the possibility of finding a model where $\mathbf{A}$ evaluates to F. Thus $\mathbf{A}$ must evaluate to T in all models, which is just our definition of validity.

Thus the tableau procedure can be used as a calculus for propositional logic. In contrast to the propositional Hilbert calculus it does not prove a theorem $\mathbf{A}$ by deriving it from a set of axioms, but it proves it by refuting its negation. Such calculi are called negative or test calculi. Generally negative calculi have computational advantages over positive ones, since they have a built-in sense of direction.

We have rules for all the necessary connectives (we restrict ourselves to $\wedge$ and $\neg$, since the others can be expressed in terms of these two via the propositional identities above. For instance, we can write $\mathbf{A}\vee\mathbf{B}$ as $\neg(\neg\mathbf{A}\wedge\neg\mathbf{B})$, and $\mathbf{A}\Rightarrow\mathbf{B}$ as $\neg\mathbf{A}\vee\mathbf{B}$,...)

We will now extend the propositional tableau techniques to first-order logic. We only have to add two new rules for the universal quantifier (in positive and negative polarity).

First-Order Standard Tableaux ($\mathcal{T}_1$)

▷ **Definition 15.1.8.** The standard tableau calculus ($\mathcal{T}_1$) extends $\mathcal{T}_0$ (propositional tableau calculus) with the following quantifier rules:

$$\frac{(\forall X.\mathbf{A})^{\mathsf{T}} \quad \mathbf{C}\in \mathit{cwff}_{\iota}(\Sigma_{\iota})}{([\mathbf{C}/X](\mathbf{A}))^{\mathsf{T}}}\,\mathcal{T}_1\forall \qquad \frac{(\forall X.\mathbf{A})^{\mathsf{F}} \quad c\in\Sigma_0^{sk}\text{ new}}{([c/X](\mathbf{A}))^{\mathsf{F}}}\,\mathcal{T}_1\exists$$

▷ **Problem:** The rule $\mathcal{T}_1\forall$ displays a case of "don't know indeterminism": to find a refutation we have to guess a formula $\mathbf{C}$ from the (usually infinite) set $\mathit{cwff}_{\iota}(\Sigma_{\iota})$.

For proof search, this means that we have to systematically try all, so $\mathcal{T}_1\forall$ is infinitely branching in general.

FAU Michael Kohlhase: Artificial Intelligence 1 438 2025-02-06

The rule $\mathcal{T}_1\forall$ operationalizes the intuition that a universally quantified formula is true, iff all of the instances of the scope are. To understand the $\mathcal{T}_1\exists$ rule, we have to keep in mind that $\exists X.\mathbf{A}$ abbreviates $\neg(\forall X.\neg\mathbf{A})$, so that we have to read $(\forall X.\mathbf{A})^{\mathsf{F}}$ existentially — i.e. as $(\exists X.\neg\mathbf{A})^{\mathsf{T}}$, stating that there is an object with property $\neg\mathbf{A}$. In this situation, we can simply give this object a name: c, which we take from our (infinite) set of witness constants Σ_0^{sk}, which we have given ourselves expressly for this purpose when we defined first-order syntax. In other words $([c/X](\neg\mathbf{A}))^{\mathsf{T}} = ([c/X](\mathbf{A}))^{\mathsf{F}}$ holds, and this is just the conclusion of the $\mathcal{T}_1\exists$ rule.

Note that the $\mathcal{T}_1\forall$ rule is computationally extremely inefficient: we have to guess an (i.e. in a search setting to systematically consider all) instance $\mathbf{C}\in\mathit{wff}_{\iota}(\Sigma_{\iota},\mathcal{V}_{\iota})$ for X. This makes the rule infinitely branching.

In the next calculus we will try to remedy the computational inefficiency of the $\mathcal{T}_1\forall$ rule. We do this by delaying the choice in the universal rule.

Free variable Tableaux ($\mathcal{T}_1^f$)

- **Definition 15.1.9.** The free variable tableau calculus ($\mathcal{T}_1^f$) extends $\mathcal{T}_0$ (propositional tableau calculus) with the quantifier rules:

  $$\frac{(\forall X.\mathbf{A})^{\mathsf{T}} \quad Y \text{ new}}{([Y/X](\mathbf{A}))^{\mathsf{T}}}\,\mathcal{T}_1^f\forall \qquad \frac{(\forall X.\mathbf{A})^{\mathsf{F}} \quad \mathrm{free}(\forall X.\mathbf{A}) = \{X^1,\ldots,X^k\} \quad f \in \Sigma_k^{sk} \text{ new}}{([f(X^1,\ldots,X^k)/X](\mathbf{A}))^{\mathsf{F}}}\,\mathcal{T}_1^f\exists$$

  and generalizes its cut rule $\mathcal{T}_0\bot$ to:

  $$\frac{\begin{matrix}\mathbf{A}^\alpha \\ \mathbf{B}^\beta\end{matrix} \quad \alpha \neq \beta \quad \sigma(\mathbf{A}) = \sigma(\mathbf{B})}{\bot : \sigma}\,\mathcal{T}_1^f\bot$$

  $\mathcal{T}_1^f\bot$ instantiates the whole tableau by σ.

- **Advantage:** No guessing necessary in $\mathcal{T}_1^f\forall$-rule!

- **New Problem:** find suitable substitution (most general unifier) (later)

Metavariables: Instead of guessing a concrete instance for the universally quantified variable as in the $\mathcal{T}_1\forall$ rule, $\mathcal{T}_1^f\forall$ instantiates it with a new metavariable Y, which will be instantiated by need in the course of the derivation.
Skolem terms as witnesses: The introduction of metavariables makes is necessary to extend the treatment of witnesses in the existential rule. Intuitively, we cannot simply invent a new name, since the meaning of the body $\mathbf{A}$ may contain metavariables introduced by the $\mathcal{T}_1^f\forall$ rule. As we do not know their values yet, the witness for the existential statement in the antecedent of the $\mathcal{T}_1^f\exists$ rule needs to depend on that. So witness it using a witness term, concretely by applying a Skolem function to the metavariables in $\mathbf{A}$.
Instantiating Metavariables: Finally, the $\mathcal{T}_1^f\bot$ rule completes the treatment of metavariables, it allows to instantiate the whole tableau in a way that the current branch closes. This leaves us with the problem of finding substitutions that make two terms equal.

Free variable Tableaux ($\mathcal{T}_1^f$): Derivable Rules

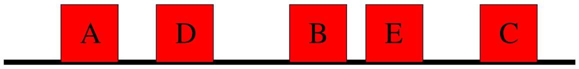
- **Definition 15.1.10.** Derivable quantifier rules in $\mathcal{T}_1^f$:

  $$\frac{(\exists X.\mathbf{A})^{\mathsf{T}} \quad \mathrm{free}(\forall X.\mathbf{A}) = \{X^1,\ldots,X^k\} \quad f \in \Sigma_k^{sk} \text{ new}}{([f(X^1,\ldots,X^k)/X](\mathbf{A}))^{\mathsf{T}}}$$

  $$\frac{(\exists X.\mathbf{A})^{\mathsf{F}} \quad Y \text{ new}}{([Y/X](\mathbf{A}))^{\mathsf{F}}}$$

Tableau Reasons about Blocks

▷ **Example 15.1.11 (Reasoning about Blocks).** Returing to slide 405



Can we prove $\mathrm{red}(\mathbf{A})$ from $\forall x.\mathrm{block}(x) \Rightarrow \mathrm{red}(x)$ and $\mathrm{block}(\mathbf{A})$?

$$\begin{array}{c|c}
\multicolumn{2}{c}{(\forall X.\mathrm{block}(X) \Rightarrow \mathrm{red}(X))^{\mathsf{T}}} \\
\multicolumn{2}{c}{\mathrm{block}(\mathbf{A})^{\mathsf{T}}} \\
\multicolumn{2}{c}{\mathrm{red}(\mathbf{A})^{\mathsf{F}}} \\
\multicolumn{2}{c}{(\mathrm{block}(Y) \Rightarrow \mathrm{red}(Y))^{\mathsf{T}}} \\
\mathrm{block}(Y)^{\mathsf{F}} & \mathrm{red}(\mathbf{A})^{\mathsf{T}} \\
\bot : [\mathbf{A}/Y] & \bot
\end{array}$$

Michael Kohlhase: Artificial Intelligence 1 441 2025-02-06

15.1.2 First-Order Unification

Video Nuggets covering this subsection can be found at `https://fau.tv/clip/id/26810` and `https://fau.tv/clip/id/26811`.

We will now look into the problem of finding a substitution σ that make two terms equal (we say it unifies them) in more detail. The presentation of the unification algorithm we give here "transformation-based" this has been a very influential way to treat certain algorithms in theoretical computer science.

A transformation-based view of algorithms: The "transformation-based" view of algorithms divides two concerns in presenting and reasoning about algorithms according to Kowalski's slogan [Kow97]

$$\text{algorithm} = \text{logic} + \text{control}$$

The computational paradigm highlighted by this quote is that (many) algorithms can be thought of as manipulating representations of the problem at hand and transforming them into a form that makes it simple to read off solutions. Given this, we can simplify thinking and reasoning about such algorithms by separating out their "logical" part, which deals with is concerned with how the problem representations can be manipulated in principle from the "control" part, which is concerned with questions about when to apply which transformations.

It turns out that many questions about the algorithms can already be answered on the "logic" level, and that the "logical" analysis of the algorithm can already give strong hints as to how to optimize control.

In fact we will only concern ourselves with the "logical" analysis of unification here.

The first step towards a theory of unification is to take a closer look at the problem itself. A first set of examples show that we have multiple solutions to the problem of finding substitutions that make two terms equal. But we also see that these are related in a systematic way.

Unification (Definitions)

▷ **Definition 15.1.12.** For given terms $\mathbf{A}$ and $\mathbf{B}$, unification is the problem of finding a substitution σ, such that $\sigma(\mathbf{A}) = \sigma(\mathbf{B})$.

▷ **Notation:** We write term pairs as $\mathbf{A}=^{?}\mathbf{B}$ e.g. $f(X)=^{?}f(g(Y))$.

- ▷ **Definition 15.1.13.** Solutions (e.g. $[g(a)/X],[a/Y]$, $[g(g(a))/X],[g(a)/Y]$, or $[g(Z)/X],[Z/Y]$) are called unifiers, $\mathbf{U}(\mathbf{A}=^?\mathbf{B}) := \{\sigma \,|\, \sigma(\mathbf{A}) = \sigma(\mathbf{B})\}$.
- ▷ **Idea:** Find representatives in $\mathbf{U}(\mathbf{A}=^?\mathbf{B})$, that generate the set of solutions.
- ▷ **Definition 15.1.14.** Let σ and θ be substitutions and $W \subseteq \mathcal{V}_\iota$, we say that a substitution σ is more general than θ (on W; write $\sigma \leq \theta[W]$), iff there is a substitution ρ, such that $\theta = \rho \circ \sigma[W]$, where $\sigma = \rho[W]$, iff $\sigma(X) = \rho(X)$ for all $X \in W$.
- ▷ **Definition 15.1.15.** σ is called a most general unifier (mgu) of $\mathbf{A}$ and $\mathbf{B}$, iff it is minimal in $\mathbf{U}(\mathbf{A}=^?\mathbf{B})$ wrt. $\leq[(\mathrm{free}(\mathbf{A}) \cup \mathrm{free}(\mathbf{B}))]$.

Michael Kohlhase: Artificial Intelligence 1 442 2025-02-06

The idea behind a most general unifier is that all other unifiers can be obtained from it by (further) instantiation. In an automated theorem proving setting, this means that using most general unifiers is the least committed choice — any other choice of unifiers (that would be necessary for completeness) can later be obtained by other substitutions.

Note that there is a subtlety in the definition of the ordering on substitutions: we only compare on a subset of the variables. The reason for this is that we have defined substitutions to be total on (the infinite set of) variables for flexibility, but in the applications (see the definition of most general unifiers), we are only interested in a subset of variables: the ones that occur in the initial problem formulation. Intuitively, we do not care what the unifiers do off that set. If we did not have the restriction to the set W of variables, the ordering relation on substitutions would become much too fine-grained to be useful (i.e. to guarantee unique most general unifiers in our case).

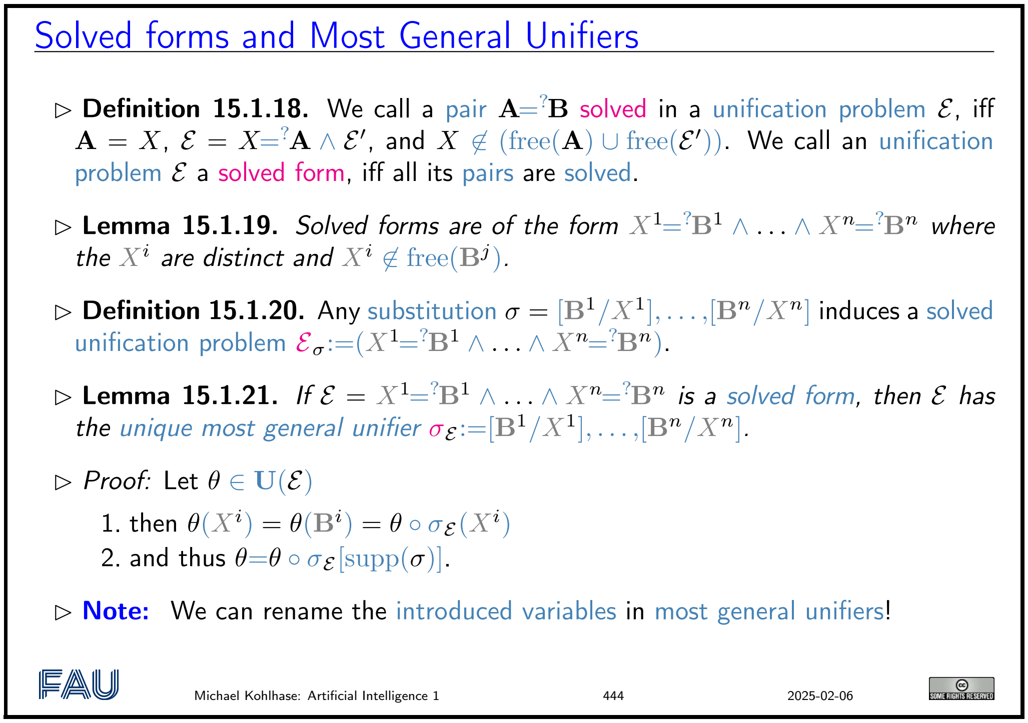
Now that we have defined the problem, we can turn to the unification algorithm itself. We will define it in a way that is very similar to logic programming: we first define a calculus that generates "solved forms" (formulae from which we can read off the solution) and reason about control later. In this case we will reason that control does not matter.

Unification Problems ($\hat{=}$ Equational Systems)

- ▷ **Idea:** Unification is equation solving.
- ▷ **Definition 15.1.16.** We call a formula $\mathbf{A}^1=^?\mathbf{B}^1 \wedge \ldots \wedge \mathbf{A}^n=^?\mathbf{B}^n$ an unification problem iff $\mathbf{A}^i, \mathbf{B}^i \in \mathit{wff}_\iota(\Sigma_\iota, \mathcal{V}_\iota)$.
- ▷ **Note:** We consider unification problems as sets of equations ($\wedge$ is ACI), and equations as two-element multisets ($=^?$ is C).
- ▷ **Definition 15.1.17.** A substitution is called a unifier for a unification problem $\mathcal{E}$ (and thus $\mathcal{D}$ unifiable), iff it is a (simultaneous) unifier for all pairs in $\mathcal{E}$.

Michael Kohlhase: Artificial Intelligence 1 443 2025-02-06

In principle, unification problems are sets of equations, which we write as conjunctions, since all of them have to be solved for finding a unifier. Note that it is not a problem for the "logical view" that the representation as conjunctions induces an order, since we know that conjunction is associative, commutative and idempotent, i.e. that conjuncts do not have an intrinsic order or multiplicity, if we consider two equational problems as equal, if they are equivalent as propositional formulae. In the same way, we will abstract from the order in equations, since we know that the equality relation is symmetric. Of course we would have to deal with this somehow in the implementation (typically, we would implement equational problems as lists of pairs), but that belongs into the "control" aspect of the algorithm, which we are abstracting from at the moment.

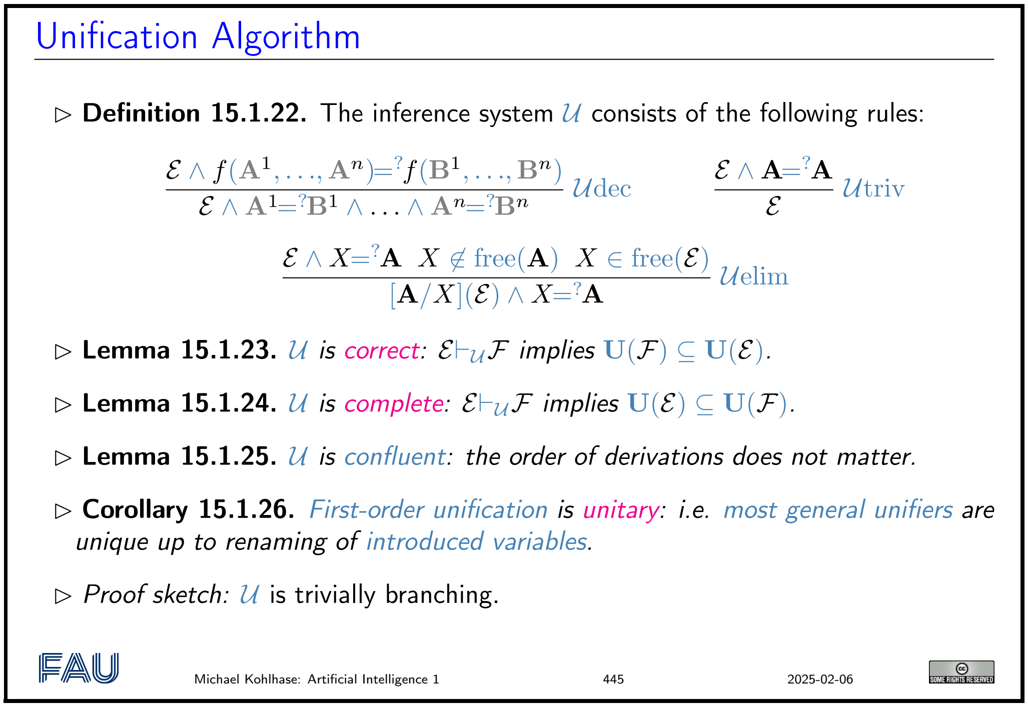


Solved forms and Most General Unifiers

- ▷ **Definition 15.1.18.** We call a pair $\mathbf{A}=^{?}\mathbf{B}$ solved in a unification problem $\mathcal{E}$, iff $\mathbf{A}=X$, $\mathcal{E}=X=^{?}\mathbf{A}\wedge\mathcal{E}'$, and $X\notin(\mathrm{free}(\mathbf{A})\cup\mathrm{free}(\mathcal{E}'))$. We call an unification problem $\mathcal{E}$ a solved form, iff all its pairs are solved.
- ▷ **Lemma 15.1.19.** *Solved forms are of the form $X^1=^{?}\mathbf{B}^1\wedge\ldots\wedge X^n=^{?}\mathbf{B}^n$ where the X^i are distinct and $X^i\notin\mathrm{free}(\mathbf{B}^j)$.*
- ▷ **Definition 15.1.20.** Any substitution $\sigma=[\mathbf{B}^1/X^1],\ldots,[\mathbf{B}^n/X^n]$ induces a solved unification problem $\mathcal{E}_\sigma{:=}(X^1=^{?}\mathbf{B}^1\wedge\ldots\wedge X^n=^{?}\mathbf{B}^n)$.
- ▷ **Lemma 15.1.21.** *If $\mathcal{E}=X^1=^{?}\mathbf{B}^1\wedge\ldots\wedge X^n=^{?}\mathbf{B}^n$ is a solved form, then $\mathcal{E}$ has the unique most general unifier $\sigma_\mathcal{E}{:=}[\mathbf{B}^1/X^1],\ldots,[\mathbf{B}^n/X^n]$.*
- ▷ *Proof:* Let $\theta\in\mathbf{U}(\mathcal{E})$
  1. then $\theta(X^i)=\theta(\mathbf{B}^i)=\theta\circ\sigma_\mathcal{E}(X^i)$
  2. and thus $\theta=\theta\circ\sigma_\mathcal{E}[\mathrm{supp}(\sigma)]$.
- ▷ **Note:** We can rename the introduced variables in most general unifiers!

It is essential to our “logical” analysis of the unification algorithm that we arrive at unification problems whose unifiers we can read off easily. Solved forms serve that need perfectly as **??** shows.

Given the idea that unification problems can be expressed as formulae, we can express the algorithm in three simple rules that transform unification problems into solved forms (or unsolvable ones).



Unification Algorithm

- ▷ **Definition 15.1.22.** The inference system $\mathcal{U}$ consists of the following rules:

$$\frac{\mathcal{E}\wedge f(\mathbf{A}^1,\ldots,\mathbf{A}^n)=^{?}f(\mathbf{B}^1,\ldots,\mathbf{B}^n)}{\mathcal{E}\wedge\mathbf{A}^1=^{?}\mathbf{B}^1\wedge\ldots\wedge\mathbf{A}^n=^{?}\mathbf{B}^n}\,\mathcal{U}\mathrm{dec}\qquad\frac{\mathcal{E}\wedge\mathbf{A}=^{?}\mathbf{A}}{\mathcal{E}}\,\mathcal{U}\mathrm{triv}$$

$$\frac{\mathcal{E}\wedge X=^{?}\mathbf{A}\quad X\notin\mathrm{free}(\mathbf{A})\quad X\in\mathrm{free}(\mathcal{E})}{[\mathbf{A}/X](\mathcal{E})\wedge X=^{?}\mathbf{A}}\,\mathcal{U}\mathrm{elim}$$

- ▷ **Lemma 15.1.23.** *$\mathcal{U}$ is correct: $\mathcal{E}\vdash_\mathcal{U}\mathcal{F}$ implies $\mathbf{U}(\mathcal{F})\subseteq\mathbf{U}(\mathcal{E})$.*
- ▷ **Lemma 15.1.24.** *$\mathcal{U}$ is complete: $\mathcal{E}\vdash_\mathcal{U}\mathcal{F}$ implies $\mathbf{U}(\mathcal{E})\subseteq\mathbf{U}(\mathcal{F})$.*
- ▷ **Lemma 15.1.25.** *$\mathcal{U}$ is confluent: the order of derivations does not matter.*
- ▷ **Corollary 15.1.26.** *First-order unification is unitary: i.e. most general unifiers are unique up to renaming of introduced variables.*
- ▷ *Proof sketch:* $\mathcal{U}$ is trivially branching.

The decomposition rule $\mathcal{U}$dec is completely straightforward, but note that it transforms one unification pair into multiple argument pairs; this is the reason, why we have to directly use unification problems with multiple pairs in $\mathcal{U}$.

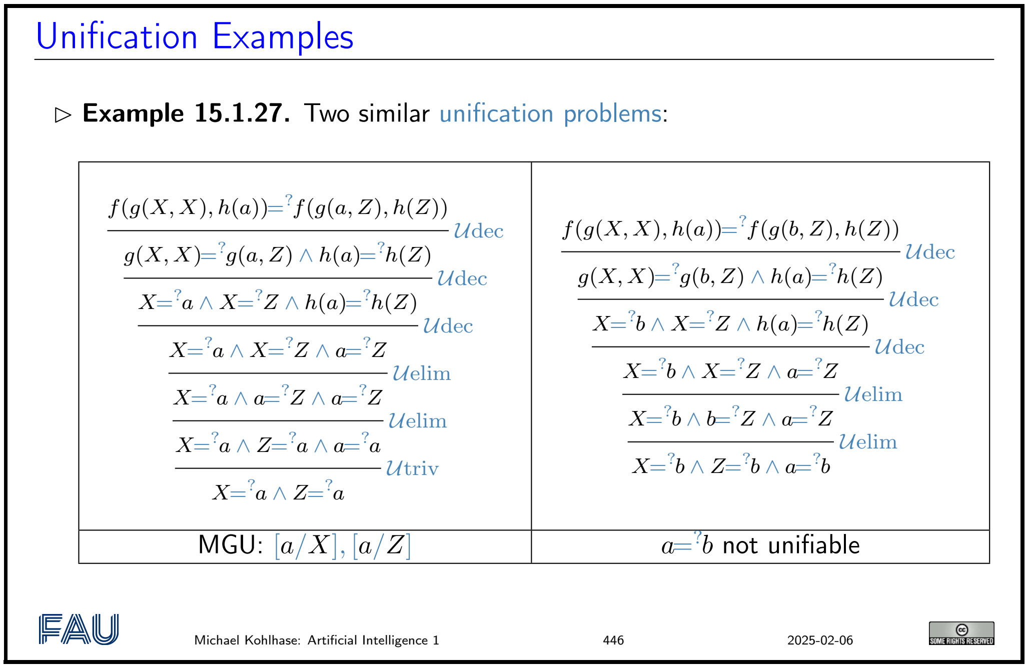
Note furthermore, that we could have restricted the $\mathcal{U}$triv rule to variable-variable pairs, since for any other pair, we can decompose until only variables are left. Here we observe, that constant-constant pairs can be decomposed with the $\mathcal{U}$dec rule in the somewhat degenerate case without arguments.

Finally, we observe that the first of the two variable conditions in $\mathcal{U}$elim (the "occurs-in-check") makes sure that we only apply the transformation to unifiable unification problems, whereas the second one is a termination condition that prevents the rule to be applied twice.

The notion of completeness and correctness is a bit different than that for calculi that we compare to the entailment relation. We can think of the "logical system of unifiability" with the model class of sets of substitutions, where a set satisfies an equational problem $\mathcal{E}$, iff all of its members are unifiers. This view induces the soundness and completeness notions presented above.

The three meta-properties above are relatively trivial, but somewhat tedious to prove, so we leave the proofs as an exercise to the reader.

We now fortify our intuition about the unification calculus by two examples. Note that we only need to pursue one possible $\mathcal{U}$ derivation since we have confluence.



Unification Examples

▷ **Example 15.1.27.** Two similar unification problems:

$f(g(X,X),h(a))=^{?}f(g(a,Z),h(Z))$ ——— $\mathcal{U}$dec $g(X,X)=^{?}g(a,Z)\wedge h(a)=^{?}h(Z)$ ——— $\mathcal{U}$dec $X=^{?}a\wedge X=^{?}Z\wedge h(a)=^{?}h(Z)$ ——— $\mathcal{U}$dec $X=^{?}a\wedge X=^{?}Z\wedge a=^{?}Z$ ——— $\mathcal{U}$elim $X=^{?}a\wedge a=^{?}Z\wedge a=^{?}Z$ ——— $\mathcal{U}$elim $X=^{?}a\wedge Z=^{?}a\wedge a=^{?}a$ ——— $\mathcal{U}$triv $X=^{?}a\wedge Z=^{?}a$	$f(g(X,X),h(a))=^{?}f(g(b,Z),h(Z))$ ——— $\mathcal{U}$dec $g(X,X)=^{?}g(b,Z)\wedge h(a)=^{?}h(Z)$ ——— $\mathcal{U}$dec $X=^{?}b\wedge X=^{?}Z\wedge h(a)=^{?}h(Z)$ ——— $\mathcal{U}$dec $X=^{?}b\wedge X=^{?}Z\wedge a=^{?}Z$ ——— $\mathcal{U}$elim $X=^{?}b\wedge b=^{?}Z\wedge a=^{?}Z$ ——— $\mathcal{U}$elim $X=^{?}b\wedge Z=^{?}b\wedge a=^{?}b$
MGU: $[a/X]$, $[a/Z]$	$a=^{?}b$ not unifiable

Michael Kohlhase: Artificial Intelligence 1 446 2025-02-06

We will now convince ourselves that there cannot be any infinite sequences of transformations in $\mathcal{U}$. Termination is an important property for an algorithm.

The proof we present here is very typical for termination proofs. We map unification problems into a partially ordered set $\langle S,\prec\rangle$ where we know that there cannot be any infinitely descending sequences (we think of this as measuring the unification problems). Then we show that all transformations in $\mathcal{U}$ strictly decrease the measure of the unification problems and argue that if there were an infinite transformation in $\mathcal{U}$, then there would be an infinite descending chain in S, which contradicts our choice of $\langle S,\prec\rangle$.

The crucial step in coming up with such proofs is finding the right partially ordered set. Fortunately, there are some tools we can make use of. We know that $\langle\mathbb{N},<\rangle$ is terminating, and there are some ways of lifting component orderings to complex structures. For instance it is well-known that the lexicographic ordering lifts a terminating ordering to a terminating ordering on finite dimensional Cartesian spaces. We show a similar, but less known construction with multisets for our proof.

Unification (Termination)

- ▷ **Definition 15.1.28.** Let S and T be multisets and $\leq$ a partial ordering on $S \cup T$. Then we define $S \prec^m S$, iff $S = C \uplus T'$ and $T = C \uplus \{t\}$, where $s \leq t$ for all $s \in S'$. We call $\leq^m$ the multiset ordering induced by $\leq$.
- ▷ **Definition 15.1.29.** We call a variable X solved in an unification problem $\mathcal{E}$, iff $\mathcal{E}$ contains a solved pair $X =^? \mathbf{A}$.
- ▷ **Lemma 15.1.30.** *If $\prec$ is linear/terminating On S, then $\prec^m$ is linear/terminating on $\mathcal{P}(S)$.*
- ▷ **Lemma 15.1.31.** *$\mathcal{U}$ is terminating.* *(any $\mathcal{U}$-derivation is finite)*
- ▷ *Proof:* We prove termination by mapping $\mathcal{U}$ transformation into a Noetherian space.
  1. Let $\mu(\mathcal{E}) := \langle n, \mathcal{N} \rangle$, where
     - ▷ n is the number of unsolved variables in $\mathcal{E}$
     - ▷ $\mathcal{N}$ is the multiset of term depths in $\mathcal{E}$
  2. The lexicographic order $\prec$ on pairs $\mu(\mathcal{E})$ is decreased by all inference rules.

     2.1. $\mathcal{U}\mathrm{dec}$ and $\mathcal{U}\mathrm{triv}$ decrease the multiset of term depths without increasing the unsolved variables.

     2.2. $\mathcal{U}\mathrm{elim}$ decreases the number of unsolved variables (by one), but may increase term depths.

But it is very simple to create terminating calculi, e.g. by having no inference rules. So there is one more step to go to turn the termination result into a decidability result: we must make sure that we have enough inference rules so that any unification problem is transformed into solved form if it is unifiable.

First-Order Unification is Decidable

- ▷ **Definition 15.1.32.** We call an equational problem $\mathcal{E}$ $\mathcal{U}$-reducible, iff there is a $\mathcal{U}$-step $\mathcal{E} \vdash_{\mathcal{U}} \mathcal{F}$ from $\mathcal{E}$.
- ▷ **Lemma 15.1.33.** *If $\mathcal{E}$ is unifiable but not solved, then it is $\mathcal{U}$-reducible.*
- ▷ *Proof:* We assume that $\mathcal{E}$ is unifiable but unsolved and show the $\mathcal{U}$ rule that applies.
  1. There is an unsolved pair $\mathbf{A} =^? \mathbf{B}$ in $\mathcal{E} = \mathcal{E} \wedge \mathbf{A} =^? \mathbf{B}'$.

  we have two cases

  2. $\mathbf{A}, \mathbf{B} \notin \mathcal{V}_\iota$

     2.1. then $\mathbf{A} = f(\mathbf{A}^1 \ldots \mathbf{A}^n)$ and $\mathbf{B} = f(\mathbf{B}^1 \ldots \mathbf{B}^n)$, and thus $\mathcal{U}\mathrm{dec}$ is applicable

  3. $\mathbf{A} = X \in \mathrm{free}(\mathcal{E})$

     3.1. then $\mathcal{U}\mathrm{elim}$ (if $\mathbf{B} \neq X$) or $\mathcal{U}\mathrm{triv}$ (if $\mathbf{B} = X$) is applicable.
- ▷ **Corollary 15.1.34.** *First-order unification is decidable in PL^1.*

  Proof:
- ▷ 1. $\mathcal{U}$-irreducible unification problems can be reached in finite time by ??.

  2. They are either solved or unsolvable by ??, so they provide the answer.

15.1.3 Efficient Unification

Now that we have seen the basic ingredients of an unification algorithm, let us as always consider complexity and efficiency issues.
We start with a look at the complexity of unification and – somewhat surprisingly – find exponential time/space complexity based simply on the fact that the results – the unifiers – can be exponentially large.

Complexity of Unification

- ▷ **Observation:** Naive implementations of unification are exponential in time and space.
- ▷ **Example 15.1.35.** Consider the terms

$$\begin{aligned} s_n &= f(f(x_0,x_0),f(f(x_1,x_1),f(\ldots,f(x_{n-1},x_{n-1}))\ldots)) \\ t_n &= f(x_1,f(x_2,f(x_3,f(\cdots,x_n)\cdots))) \end{aligned}$$

  - ▹ The most general unifier of s_n and t_n is

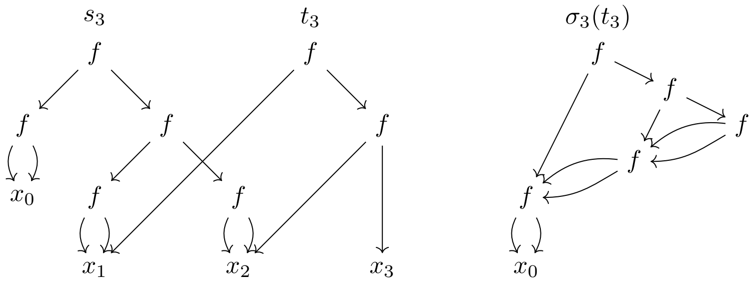
    $\sigma_n := [f(x_0,x_0)/x_1],[f(f(x_0,x_0),f(x_0,x_0))/x_2],[f(f(f(x_0,x_0),f(x_0,x_0)),f(f(x_0,x_0),f(x_0,x_0)))/x_3],\ldots$

  - ▹ It contains $\sum_{i=1}^{n} 2^i = 2^{n+1}-2$ occurrences of the variable x_0. (exponential)
- ▷ **Problem:** The variable x_0 has been copied too often.
- ▷ **Idea:** Find a term representation that re-uses subterms.

Indeed, the only way to escape this combinatorial explosion is to find representations of substitutions that are more space efficient.

Directed Acyclic Graphs (DAGs) for Terms

- ▷ **Recall:** Terms in first-order logic are essentially trees.
- ▷ **Concrete Idea:** Use directed acyclic graphs for representing terms:
  - ▹ variables my only occur once in the DAG.
  - ▹ subterms can be referenced multiply. (subterm sharing)
  - ▹ we can even represent multiple terms in a common DAG
- ▷ **Observation 15.1.36.** *Terms can be transformed into DAGs in linear time.*
- ▷ **Example 15.1.37.** Continuing from ?? … s_3, t_3, and $\sigma_3(s_3)$ as DAGs:



In general: s_n, t_n, and $\sigma_n(s_n)$ only need space in $\mathcal{O}(n)$. (just count)

If we look at the unification algorithm from **??** and the considerations in the termination proof (**??**) with a particular focus on the role of copying, we easily find the culprit for the exponential blowup: $\mathcal{U}\mathrm{elim}$, which applies solved pairs as substitutions.

DAG Unification Algorithm

- **Observation:** In $\mathcal{U}$, the $\mathcal{U}\mathrm{elim}$ rule applies solved pairs $\rightsquigarrow$ subterm duplication.
- **Idea:** Replace $\mathcal{U}\mathrm{elim}$ the notion of solved forms by something better.
- **Definition 15.1.38.** We say that $X^1=^?\mathbf{B}^1 \wedge \ldots \wedge X^n=^?\mathbf{B}^n$ is a DAG solved form, iff the X^i are distinct and $X^i \notin \mathrm{free}(\mathbf{B}^j)$ for $i \leq j$.
- **Definition 15.1.39.** The inference system $\mathcal{DU}$ contains rules $\mathcal{U}\mathrm{dec}$ and $\mathcal{U}\mathrm{triv}$ from $\mathcal{U}$ plus the following:

$$\frac{\mathcal{E} \wedge X=^?\mathbf{A} \wedge X=^?\mathbf{B} \quad \mathbf{A},\mathbf{B} \notin \mathcal{V}_\iota \quad |\mathbf{A}| \leq |\mathbf{B}|}{\mathcal{E} \wedge X=^?\mathbf{A} \wedge \mathbf{A}=^?\mathbf{B}}\ \mathcal{DU}\mathrm{merge}$$

$$\frac{\mathcal{E} \wedge X=^?Y \quad X \neq Y \quad X,Y \in \mathrm{free}(\mathcal{E})}{[Y/X](\mathcal{E}) \wedge X=^?Y}\ \mathcal{DU}\mathrm{evar}$$

  where $|\mathbf{A}|$ is the number of symbols in $\mathbf{A}$.
- The analysis for $\mathcal{U}$ applies mutatis mutandis.

We will now turn the ideas we have developed in the last couple of slides into a usable functional algorithm. The starting point is treating terms as DAGs. Then we try to conduct the transformation into solved form without adding new nodes.

Unification by DAG-chase

- **Idea:** Extend the Input-DAGs by edges that represent unifiers.
- **Definition 15.1.40.** Write $n.a$, if a is the symbol of node n.
- (standard) auxiliary procedures: (all constant or linear time in DAGs)
  - find(n) follows the path from n and returns the end node.

- ▷ union(n,m) adds an edge between n and m.
- ▷ occur(n,m) determines whether $n.x$ occurs in the DAG with root m.

Algorithm dag—unify

▷ Input: symmetric pairs of nodes in DAGs

```
fun dag—unify($n$,$n$) = true
  | dag—unify($n.x$,$m$) = if occur($n$,$m$) then true else union($n$,$m$)
  | dag—unify($n.f$,$m.g$) =
      if $g$!=$f$ then false
      else
         forall ($i$,$j$) => dag—unify(find($i$),find($j$)) (chld $m$,chld $n$)
      end
```

▷ **Observation 15.1.41.** dag—unify *uses linear space, since no new nodes are created, and at most one link per variable.*

▷ **Problem:** dag—unify still uses exponential time.

▷ **Example 15.1.42.** Consider terms $f(s_n,f(t'_n,x_n)),f(t_n,f(s'_n,y_n)))$, where $s'_n = [y_i/x_i](s_n)$ und $t'_n = [y_i/x_i](t_n)$.

dag—unify needs exponentially many recursive calls to unify the nodes x_n and y_n. (they are unified after n calls, but checking needs the time)

Algorithm uf—unify

▷ **Recall:** dag—unify still uses exponential time.

▷ **Idea:** Also bind the function nodes, if the arguments are unified.

```
uf—unify($n.f$,$m.g$) =
  if $g$!=$f$ then false
  else union($n$,$m$);
    forall ($i$,$j$) => uf—unify(find($i$),find($j$)) (chld $m$,chld $n$)
  end
```

▷ This only needs linearly many recursive calls as it directly returns with true or makes a node inaccessible for find.

▷ Linearly many calls to linear procedures give quadratic running time.

▷ **Remark:** There are versions of uf—unify that are linear in time and space, but for most purposes, our algorithm suffices.

15.1.4 Implementing First-Order Tableaux

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/26797`.
We now come to some issues (and clarifications) pertaining to implementing proof search for free variable tableaux. They all have to do with the – often overlooked – fact that $\mathcal{T}_1^f\bot$ instantiates the whole tableau.
The first question one may ask for implementation is whether we expect a terminating proof search; after all, $\mathcal{T}_0$ terminated. We will see that the situation for $\mathcal{T}_1^f$ is different.

Termination and Multiplicity in Tableaux

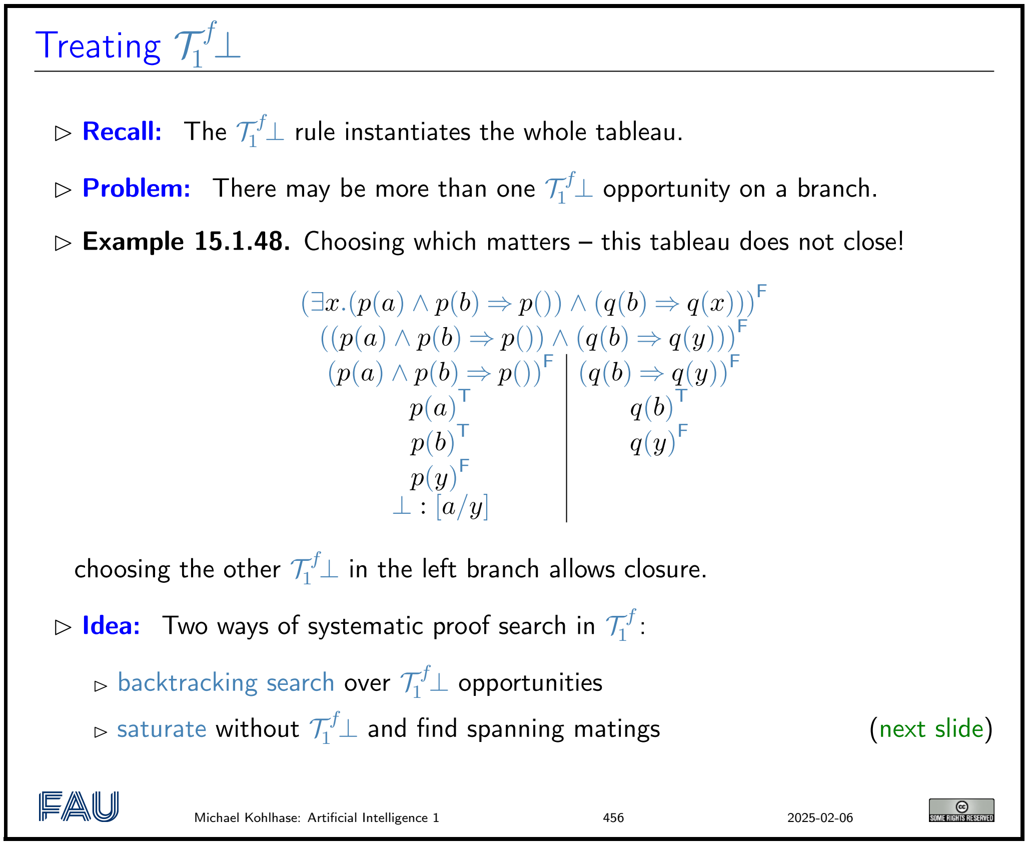
- ▷ **Recall:** In $\mathcal{T}_0$, all rules only needed to be applied once.
  ⇝ $\mathcal{T}_0$ terminates and thus induces a decision procedure for PL^0.
- ▷ **Observation 15.1.43.** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f\forall$ only need to be applied once.*
- ▷ **Example 15.1.44.** A tableau proof for $(p(a)\vee p(b))\Rightarrow(\exists.p())$.

<table>
<tr><th>Start, close left branch</th><th>use $\mathcal{T}_1^f\forall$ again (right branch)</th></tr>
<tr><td>$\begin{array}{c} ((p(a)\vee p(b))\Rightarrow(\exists.p()))^{\mathsf{F}} \\ (p(a)\vee p(b))^{\mathsf{T}} \\ (\exists x.p(x))^{\mathsf{F}} \\ (\forall x.\neg p(x))^{\mathsf{T}} \\ \neg p(y)^{\mathsf{T}} \\ p(y)^{\mathsf{F}} \\ \begin{array}{c|c} p(a)^{\mathsf{T}} & p(b)^{\mathsf{T}} \\ \bot:[a/y] & \end{array} \end{array}$</td><td>$\begin{array}{c} ((p(a)\vee p(b))\Rightarrow(\exists.p()))^{\mathsf{F}} \\ (p(a)\vee p(b))^{\mathsf{T}} \\ (\exists x.p(x))^{\mathsf{F}} \\ (\forall x.\neg p(x))^{\mathsf{T}} \\ \neg p(a)^{\mathsf{T}} \\ p(a)^{\mathsf{F}} \\ \begin{array}{c|c} p(a)^{\mathsf{T}} & p(b)^{\mathsf{T}} \\ \bot:[a/y] & \neg p(z)^{\mathsf{T}} \\ & p(z)^{\mathsf{F}} \\ & \bot:[b/z] \end{array} \end{array}$</td></tr>
</table>

  After we have used up $p(y)^{\mathsf{F}}$ by applying $[a/y]$ in $\mathcal{T}_1^f\bot$, we have to get a new instance $p(z)^{\mathsf{F}}$ via $\mathcal{T}_1^f\forall$.
- ▷ **Definition 15.1.45.** Let $\mathcal{T}$ be a tableau for $\mathbf{A}$, and a positive occurrence of $\forall x.\mathbf{B}$ in $\mathbf{A}$, then we call the number of applications of $\mathcal{T}_1^f\forall$ to $\forall x.\mathbf{B}$ its multiplicity.
- ▷ **Observation 15.1.46.** *Given a prescribed multiplicity for each positive $\forall$, saturation with $\mathcal{T}_1^f$ terminates.*
- ▷ *Proof sketch:* All $\mathcal{T}_1^f$ rules reduce the number of connectives and negative $\forall$ or the multiplicity of positive $\forall$.
- ▷ **Theorem 15.1.47.** *$\mathcal{T}_1^f$ is only complete with unbounded multiplicities.*
- ▷ *Proof sketch:* Replace $p(a)\vee p(b)$ with $p(a_1)\vee\ldots\vee p(a_n)$ in ??.
- ▷ **Remark:** Otherwise validity in PL^1 would be decidable.
- ▷ **Implementation:** We need an iterative multiplicity deepening process.

The other thing we need to realize is that there may be multiple ways we can use $\mathcal{T}_1^f\bot$ to close a branch in a tableau, and – as $\mathcal{T}_1^f\bot$ instantiates the whole tableau and not just the branch itself –

this choice matters.



Treating $\mathcal{T}_1^f\bot$

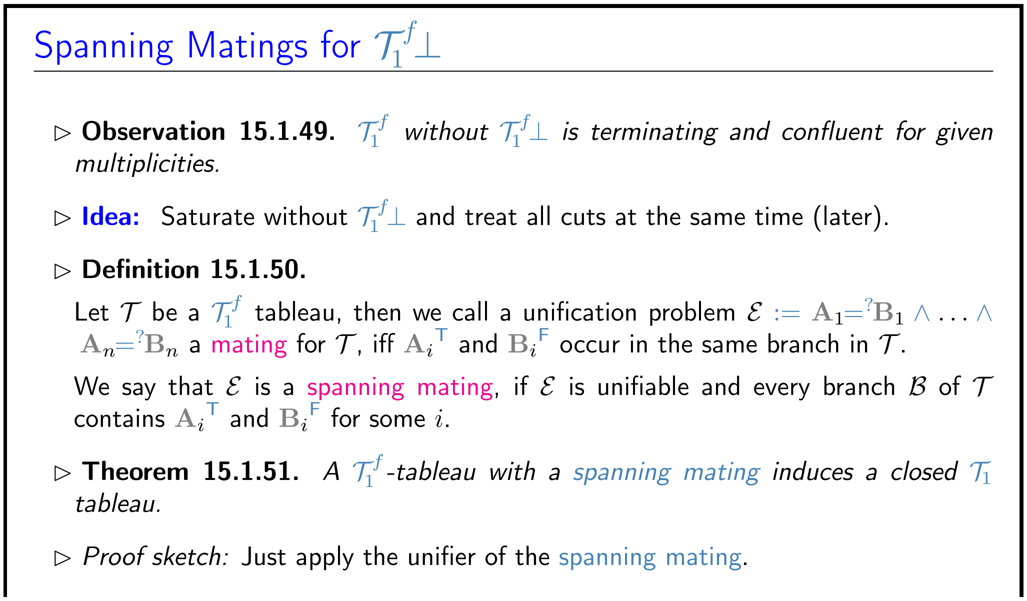
- **Recall:** The $\mathcal{T}_1^f\bot$ rule instantiates the whole tableau.
- **Problem:** There may be more than one $\mathcal{T}_1^f\bot$ opportunity on a branch.
- **Example 15.1.48.** Choosing which matters – this tableau does not close!

$$\begin{array}{c|c}
\multicolumn{2}{c}{(\exists x.(p(a) \land p(b) \Rightarrow p()) \land (q(b) \Rightarrow q(x)))^{\mathsf{F}}} \\
\multicolumn{2}{c}{((p(a) \land p(b) \Rightarrow p()) \land (q(b) \Rightarrow q(y)))^{\mathsf{F}}} \\
(p(a) \land p(b) \Rightarrow p())^{\mathsf{F}} & (q(b) \Rightarrow q(y))^{\mathsf{F}} \\
p(a)^{\mathsf{T}} & q(b)^{\mathsf{T}} \\
p(b)^{\mathsf{T}} & q(y)^{\mathsf{F}} \\
p(y)^{\mathsf{F}} & \\
\bot : [a/y] &
\end{array}$$

choosing the other $\mathcal{T}_1^f\bot$ in the left branch allows closure.

- **Idea:** Two ways of systematic proof search in $\mathcal{T}_1^f$:
  - backtracking search over $\mathcal{T}_1^f\bot$ opportunities
  - saturate without $\mathcal{T}_1^f\bot$ and find spanning matings (next slide)

The method of spanning matings follows the intuition that if we do not have good information on how to decide for a pair of opposite literals on a branch to use in $\mathcal{T}_1^f\bot$, we delay the choice by initially disregarding the rule altogether during saturation and then – in a later phase– looking for a configuration of cuts that have a joint overall unifier. The big advantage of this is that we only need to know that one exists, we do not need to compute or apply it, which would lead to exponential blow-up as we have seen above.



Spanning Matings for $\mathcal{T}_1^f\bot$

- **Observation 15.1.49.** *$\mathcal{T}_1^f$ without $\mathcal{T}_1^f\bot$ is terminating and confluent for given multiplicities.*
- **Idea:** Saturate without $\mathcal{T}_1^f\bot$ and treat all cuts at the same time (later).
- **Definition 15.1.50.**

  Let $\mathcal{T}$ be a $\mathcal{T}_1^f$ tableau, then we call a unification problem $\mathcal{E} := \mathbf{A}_1 =^? \mathbf{B}_1 \land \ldots \land \mathbf{A}_n =^? \mathbf{B}_n$ a mating for $\mathcal{T}$, iff $\mathbf{A}_i^{\mathsf{T}}$ and $\mathbf{B}_i^{\mathsf{F}}$ occur in the same branch in $\mathcal{T}$.

  We say that $\mathcal{E}$ is a spanning mating, if $\mathcal{E}$ is unifiable and every branch $\mathcal{B}$ of $\mathcal{T}$ contains $\mathbf{A}_i^{\mathsf{T}}$ and $\mathbf{B}_i^{\mathsf{F}}$ for some i.
- **Theorem 15.1.51.** *A $\mathcal{T}_1^f$-tableau with a spanning mating induces a closed $\mathcal{T}_1$ tableau.*
- *Proof sketch:* Just apply the unifier of the spanning mating.

- ▷ **Idea:** Existence is sufficient, we do not need to compute the unifier.
- ▷ **Implementation:** Saturate without $\mathcal{T}_1^f\bot$, backtracking search for spanning matings with $\mathcal{DU}$, adding pairs incrementally.

Excursion: Now that we understand basic unification theory, we can come to the meta-theoretical properties of the tableau calculus. We delegate this discussion to??.

15.2 First-Order Resolution

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/26817`.

First-Order Resolution (and CNF)

- ▷ **Definition 15.2.1.** The first-order CNF calculus CNF_1 is given by the inference rules of CNF_0 extended by the following quantifier rules:

$$\frac{(\forall X.\mathbf{A})^{\mathsf{T}} \vee \mathbf{C} \quad Z \notin (\mathrm{free}(\mathbf{A}) \cup \mathrm{free}(\mathbf{C}))}{([Z/X](\mathbf{A}))^{\mathsf{T}} \vee \mathbf{C}}$$

$$\frac{(\forall X.\mathbf{A})^{\mathsf{F}} \vee \mathbf{C} \quad \{X_1, \ldots, X_k\} = \mathrm{free}(\forall X.\mathbf{A}) \quad f \in \Sigma_k^{sk} \text{ new}}{([f(X_1, \ldots, X_k)/X](\mathbf{A}))^{\mathsf{F}} \vee \mathbf{C}}$$

the first-order CNF $\mathit{CNF}_1(\Phi)$ of Φ is the set of all clauses that can be derived from Φ.

- ▷ **Definition 15.2.2 (First-Order Resolution Calculus).** The First-order resolution calculus ($\mathcal{R}_1$) is a test calculus that manipulates formulae in conjunctive normal form. $\mathcal{R}_1$ has two inference rules:

$$\frac{\mathbf{A}^{\mathsf{T}} \vee \mathbf{C} \quad \mathbf{B}^{\mathsf{F}} \vee \mathbf{D} \quad \sigma = \mathrm{mgu}(\mathbf{A}, \mathbf{B})}{(\sigma(\mathbf{C})) \vee (\sigma(\mathbf{D}))} \qquad \frac{\mathbf{A}^{\alpha} \vee \mathbf{B}^{\alpha} \vee \mathbf{C} \quad \sigma = \mathrm{mgu}(\mathbf{A}, \mathbf{B})}{(\sigma(\mathbf{A})) \vee (\sigma(\mathbf{C}))}$$

First-Order CNF – Derived Rules

- ▷ **Definition 15.2.3.** The following inference rules are derivable from the ones above via $(\exists X.\mathbf{A}) = \neg(\forall X.\neg\mathbf{A})$:

$$\frac{(\exists X.\mathbf{A})^{\mathsf{T}} \vee \mathbf{C} \quad \{X_1, \ldots, X_k\} = \mathrm{free}(\forall X.\mathbf{A}) \quad f \in \Sigma_k^{sk} \text{ new}}{([f(X_1, \ldots, X_k)/X](\mathbf{A}))^{\mathsf{T}} \vee \mathbf{C}}$$

$$\frac{(\exists X.\mathbf{A})^{\mathsf{F}} \vee \mathbf{C} \quad Z \notin (\mathrm{free}(\mathbf{A}) \cup \mathrm{free}(\mathbf{C}))}{([Z/X](\mathbf{A}))^{\mathsf{F}} \vee \mathbf{C}}$$

Excursion: Again, we relegate the meta-theoretical properties of the first-order resolution calculus to??.

15.2.1 Resolution Examples

Col. West, a *Criminal?*

▷ **Example 15.2.4.** From [RN09]

> The law says it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

Prove that Col. West is a criminal.

▷ **Remark:** Modern resolution theorem provers prove this in less than $50\mathrm{ms}$.

▷ **Problem:** That is only true, if we only give the theorem prover exactly the right laws and background knowledge. If we give it all of them, it drowns in the combinatorial explosion.

▷ Let us build a resolution proof for the claim above.

▷ **But first** we must translate the situation into first-order logic clauses.

▷ **Convention:** In what follows, for better readability we will sometimes write implications $P \wedge Q \wedge R \Rightarrow S$ instead of clauses $P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} \vee S^{\mathsf{T}}$.

Col. West, a *Criminal?*

▷ *It is a crime for an American to sell weapons to hostile nations*:
Clause: $\mathrm{ami}(X_1) \wedge \mathrm{weap}(Y_1) \wedge \mathrm{sell}(X_1, Y_1, Z_1) \wedge \mathrm{host}(Z_1) \Rightarrow \mathrm{crook}(X_1)$

▷ *Nono has some missiles*: $\exists X.\mathrm{own}(\mathrm{NN}, X) \wedge \mathrm{mle}(X)$
Clauses: $\mathrm{own}(\mathrm{NN}, c)^{\mathsf{T}}$ and $\mathrm{mle}(c)$ (c is Skolem constant)

▷ *All of Nono's missiles were sold to it by Colonel West.*
Clause: $\mathrm{mle}(X_2) \wedge \mathrm{own}(\mathrm{NN}, X_2) \Rightarrow \mathrm{sell}(\mathrm{West}, X_2, \mathrm{NN})$

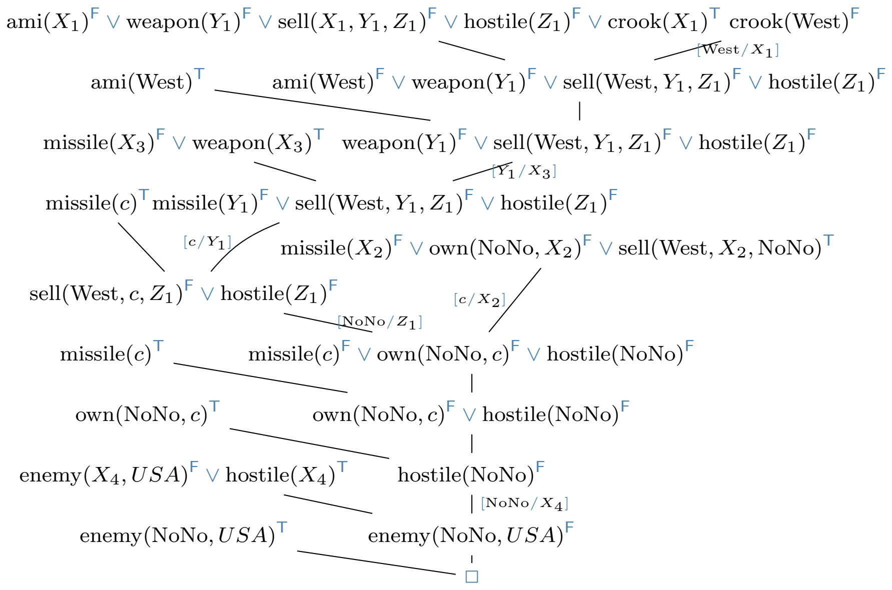
▷ *Missiles are weapons*:
Clause: $\mathrm{mle}(X_3) \Rightarrow \mathrm{weap}(X_3)$

▷ *An enemy of America counts as "hostile"*:
Clause: $\mathrm{enmy}(X_4, \mathrm{USA}) \Rightarrow \mathrm{host}(X_4)$

▷ *West is an American:*
Clause: $\mathrm{ami}(\mathrm{West})$

▷ *The country Nono is an enemy of America*:
$\mathrm{enmy}(\mathrm{NN}, \mathrm{USA})$

Col. West, *a Criminal!* PL1 Resolution Proof



$\mathrm{ami}(X_1)^{\mathsf{F}} \vee \mathrm{weapon}(Y_1)^{\mathsf{F}} \vee \mathrm{sell}(X_1,Y_1,Z_1)^{\mathsf{F}} \vee \mathrm{hostile}(Z_1)^{\mathsf{F}} \vee \mathrm{crook}(X_1)^{\mathsf{T}}$ $\quad \mathrm{crook}(\mathrm{West})^{\mathsf{F}}$

$[\mathrm{West}/X_1]$

$\mathrm{ami}(\mathrm{West})^{\mathsf{T}}$ $\quad \mathrm{ami}(\mathrm{West})^{\mathsf{F}} \vee \mathrm{weapon}(Y_1)^{\mathsf{F}} \vee \mathrm{sell}(\mathrm{West},Y_1,Z_1)^{\mathsf{F}} \vee \mathrm{hostile}(Z_1)^{\mathsf{F}}$

$\mathrm{missile}(X_3)^{\mathsf{F}} \vee \mathrm{weapon}(X_3)^{\mathsf{T}}$ $\quad \mathrm{weapon}(Y_1)^{\mathsf{F}} \vee \mathrm{sell}(\mathrm{West},Y_1,Z_1)^{\mathsf{F}} \vee \mathrm{hostile}(Z_1)^{\mathsf{F}}$

$[Y_1/X_3]$

$\mathrm{missile}(c)^{\mathsf{T}}$ $\quad \mathrm{missile}(Y_1)^{\mathsf{F}} \vee \mathrm{sell}(\mathrm{West},Y_1,Z_1)^{\mathsf{F}} \vee \mathrm{hostile}(Z_1)^{\mathsf{F}}$

$[c/Y_1]$ $\quad \mathrm{missile}(X_2)^{\mathsf{F}} \vee \mathrm{own}(\mathrm{NoNo},X_2)^{\mathsf{F}} \vee \mathrm{sell}(\mathrm{West},X_2,\mathrm{NoNo})^{\mathsf{T}}$

$\mathrm{sell}(\mathrm{West},c,Z_1)^{\mathsf{F}} \vee \mathrm{hostile}(Z_1)^{\mathsf{F}}$ $\quad [c/X_2]$

$[\mathrm{NoNo}/Z_1]$

$\mathrm{missile}(c)^{\mathsf{T}}$ $\quad \mathrm{missile}(c)^{\mathsf{F}} \vee \mathrm{own}(\mathrm{NoNo},c)^{\mathsf{F}} \vee \mathrm{hostile}(\mathrm{NoNo})^{\mathsf{F}}$

$\mathrm{own}(\mathrm{NoNo},c)^{\mathsf{T}}$ $\quad \mathrm{own}(\mathrm{NoNo},c)^{\mathsf{F}} \vee \mathrm{hostile}(\mathrm{NoNo})^{\mathsf{F}}$

$\mathrm{enemy}(X_4,USA)^{\mathsf{F}} \vee \mathrm{hostile}(X_4)^{\mathsf{T}}$ $\quad \mathrm{hostile}(\mathrm{NoNo})^{\mathsf{F}}$

$[\mathrm{NoNo}/X_4]$

$\mathrm{enemy}(\mathrm{NoNo},USA)^{\mathsf{T}}$ $\quad \mathrm{enemy}(\mathrm{NoNo},USA)^{\mathsf{F}}$

$\Box$

Curiosity Killed the Cat?

▷ **Example 15.2.5.** From [RN09]

Everyone who loves all animals is loved by someone.
Anyone who kills an animal is loved by noone.
Jack loves all animals.
Cats are animals.
Either Jack or curiosity killed the cat (whose name is "Garfield").

Prove that curiosity killed the cat.

Curiosity Killed the Cat? Clauses

▷ *Everyone who loves all animals is loved by someone*:
$\forall X.(\forall Y.\mathrm{animal}(Y) \Rightarrow \mathrm{love}(X,Y)) \Rightarrow (\exists.\mathrm{love}(Z,X))$
Clauses: $\mathrm{animal}(g(X_1))^{\mathsf{T}} \vee \mathrm{love}(g(X_1),X_1)^{\mathsf{T}}$ and $\mathrm{love}(X_2,f(X_2))^{\mathsf{F}} \vee \mathrm{love}(g(X_2),X_2)^{\mathsf{T}}$

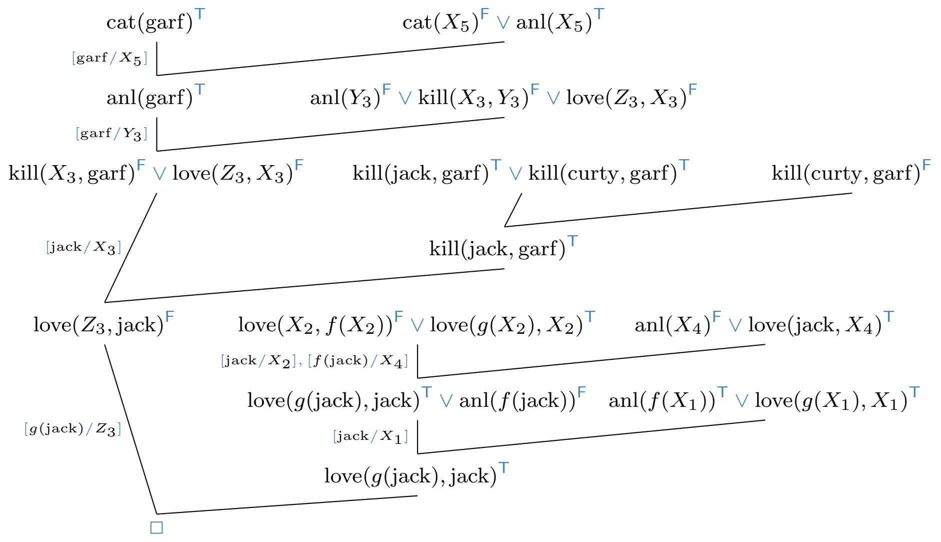
▷ *Anyone who kills an animal is loved by noone*:
$\forall X.\exists Y.\mathrm{animal}(Y) \wedge \mathrm{kill}(X,Y) \Rightarrow (\forall.\neg\mathrm{love}(Z,X))$
Clause: $\mathrm{animal}(Y_3)^{\mathsf{F}} \vee \mathrm{kill}(X_3,Y_3)^{\mathsf{F}} \vee \mathrm{love}(Z_3,X_3)^{\mathsf{F}}$

▷ *Jack loves all animals*:

Clause: $\text{animal}(X_4)^{\mathsf{F}} \vee \text{love}(\text{jack}, X_4)^{\mathsf{T}}$

- ▷ *Cats are animals*:
  Clause: $\text{cat}(X_5)^{\mathsf{F}} \vee \text{animal}(X_5)^{\mathsf{T}}$
- ▷ *Either Jack or curiosity killed the cat (whose name is "Garfield")*:
  Clauses: $\text{kill}(\text{jack}, \text{garf})^{\mathsf{T}} \vee \text{kill}(\text{curiosity}, \text{garf})^{\mathsf{T}}$ and $\text{cat}(\text{garf})^{\mathsf{T}}$

Curiosity Killed the Cat! PL1 Resolution Proof


$\text{cat}(\text{garf})^{\mathsf{T}}$
$\text{cat}(X_5)^{\mathsf{F}} \vee \text{anl}(X_5)^{\mathsf{T}}$
$[\text{garf}/X_5]$
$\text{anl}(\text{garf})^{\mathsf{T}}$
$\text{anl}(Y_3)^{\mathsf{F}} \vee \text{kill}(X_3, Y_3)^{\mathsf{F}} \vee \text{love}(Z_3, X_3)^{\mathsf{F}}$
$[\text{garf}/Y_3]$
$\text{kill}(X_3, \text{garf})^{\mathsf{F}} \vee \text{love}(Z_3, X_3)^{\mathsf{F}}$
$\text{kill}(\text{jack}, \text{garf})^{\mathsf{T}} \vee \text{kill}(\text{curty}, \text{garf})^{\mathsf{T}}$
$\text{kill}(\text{curty}, \text{garf})^{\mathsf{F}}$
$[\text{jack}/X_3]$
$\text{kill}(\text{jack}, \text{garf})^{\mathsf{T}}$
$\text{love}(Z_3, \text{jack})^{\mathsf{F}}$
$\text{love}(X_2, f(X_2))^{\mathsf{F}} \vee \text{love}(g(X_2), X_2)^{\mathsf{T}}$
$\text{anl}(X_4)^{\mathsf{F}} \vee \text{love}(\text{jack}, X_4)^{\mathsf{T}}$
$[\text{jack}/X_2], [f(\text{jack})/X_4]$
$\text{love}(g(\text{jack}), \text{jack})^{\mathsf{T}} \vee \text{anl}(f(\text{jack}))^{\mathsf{F}}$
$\text{anl}(f(X_1))^{\mathsf{T}} \vee \text{love}(g(X_1), X_1)^{\mathsf{T}}$
$[g(\text{jack})/Z_3]$
$[\text{jack}/X_1]$
$\text{love}(g(\text{jack}), \text{jack})^{\mathsf{T}}$
$\square$

Excursion: A full analysis of any calculus needs a completeness proof. We will not cover this in the course, but provide one for the calculi introduced so far in**??**.

15.3 Logic Programming as Resolution Theorem Proving

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/26820`. To understand Prolog better, we can interpret the language of Prolog as resolution in PL¹.

We know all this already

- ▷ Goals, goal sets, rules, and facts are just clauses. (called Horn clauses)
- ▷ **Observation 15.3.1 (Rule).** *H:−B_1,...,B_n. corresponds to $H^{\mathsf{T}} \vee {B_1}^{\mathsf{F}} \vee \ldots \vee {B_n}^{\mathsf{F}}$ (head the only positive literal)*
- ▷ **Observation 15.3.2 (Goal set).** *?− G_1,...,G_n. corresponds to ${G_1}^{\mathsf{F}} \vee \ldots \vee {G_n}^{\mathsf{F}}$*
- ▷ **Observation 15.3.3 (Fact).** *F. corresponds to the unit clause F^{T}.*

- **Definition 15.3.4.** A Horn clause is a clause with at most one positive literal.
- **Recall:** Backchaining as search:
  - state = tuple of goals; goal state = empty list (of goals).
  - $next(\langle G, R_1, \ldots, R_l \rangle) := \langle \sigma(B_1), \ldots, \sigma(B_m), \sigma(R_1), \ldots, \sigma(R_l) \rangle$ if there is a rule H:−B_1,. . ., B_m. and a substitution σ with $\sigma(H) = \sigma(G)$.
- **Note:** Backchaining becomes resolution

$$\frac{P^{\mathsf{T}} \vee \mathbf{A} \quad P^{\mathsf{F}} \vee \mathbf{B}}{\mathbf{A} \vee \mathbf{B}}$$

positive, unit-resulting hyperresolution (PURR)

This observation helps us understand Prolog better, and use implementation techniques from automated theorem proving.

PROLOG (Horn Logic)

- **Definition 15.3.5.** A clause is called a Horn clause, iff contains at most one positive literal, i.e. if it is of the form $B_1{}^{\mathsf{F}} \vee \ldots \vee B_n{}^{\mathsf{F}} \vee A^{\mathsf{T}}$ – i.e. A:−B_1,. . .,B_n. in Prolog notation.
  - Rule clause: general case, e.g. fallible(X) : human(X).
  - Fact clause: no negative literals, e.g. human(sokrates).
  - Program: set of rule and fact clauses.
  - Query: no positive literals: e.g. ?− fallible(X),greek(X).
- **Definition 15.3.6.** Horn logic is the formal system whose language is the set of Horn clauses together with the calculus $\mathcal{H}$ given by MP, $\wedge I$, and Subst.
- **Definition 15.3.7.** A logic program P entails a query Q with answer substitution σ, iff there is a $\mathcal{H}$ derivation D of Q from P and σ is the combined substitution of the Subst instances in D.

PROLOG: Our Example

- **Program:**

```
human(leibniz).
human(sokrates).
greek(sokrates).
fallible(X):−human(X).
```

- **Example 15.3.8 (Query).** ?− fallible(X),greek(X).
- **Answer substitution:** $[\mathrm{sokrates}/X]$

To gain an intuition for this quite abstract definition let us consider a concrete knowledge base about cars. Instead of writing down everything we know about cars, we only write down that cars are motor vehicles with four wheels and that a particular object c has a motor and four wheels. We can see that the fact that c is a car can be derived from this. Given our definition of a knowledge base as the deductive closure of the facts and rule explicitly written down, the assertion that c is a car is in the induced knowledge base, which is what we are after.

Knowledge Base (Example)

▷ **Example 15.3.9.** car(c). is in the knowlege base generated by

```
has_motor(c).
has_wheels(c,4).
car(X):— has_motor(X),has_wheels(X,4).
```

$$\cfrac{\cfrac{m(c) \quad w(c,4)}{m(c) \wedge w(c,4)}\wedge I \qquad \cfrac{m(x) \wedge w(x,4) \Rightarrow car()}{m(c) \wedge w(c,4) \Rightarrow car()}\text{Subst}}{car(c)}\text{MP}$$

In this very simple example $car(c)$ is about the only fact we can derive, but in general, knowledge bases can be infinite (we will see examples below).

Why Only Horn Clauses?

▷ General clauses of the form A1,. . .,An : B1,. . .,Bn.

▷ e.g. greek(sokrates),greek(perikles)

- ▷ **Question**: Are there fallible greeks?
- ▷ **Indefinite answer**: Yes, Perikles or Sokrates
- ▷ Warning: how about Sokrates and Perikles?

▷ e.g. greek(sokrates),roman(sokrates):—.

- ▷ **Query**: Are there fallible greeks?
- ▷ **Answer**: Yes, Sokrates, if he is not a roman
- ▷ Is this abduction?????

Three Principal Modes of Inference

▷ **Definition 15.3.10.** Deduction $\widehat{=}$ knowledge extension

▷ **Example 15.3.11.** $\frac{rains \Rightarrow wet\_street \quad rains}{wet\_street}\,D$

▷ **Definition 15.3.12.** Abduction $\widehat{=}$ explanation

▷ **Example 15.3.13.** $\frac{rains \Rightarrow wet\_street \quad wet\_street}{rains}\,A$

▷ **Definition 15.3.14.** Induction $\widehat{=}$ learning general rules from examples

▷ **Example 15.3.15.** $\frac{wet\_street \quad rains}{rains \Rightarrow wet\_street}\,I$

15.4 Summary: ATP in First-Order Logic

Summary: ATP in First-Order Logic

▷ The propositional calculi for ATP can be extended to first-order logic by adding quantifier rules.

- ▹ The rule for the universal quantifier can be made efficient by introducing metavariables that postpone the decision for instances.
- ▹ We have to extend the witness constants in the rules for existential quantifiers to Skolem functions.
- ▹ The cut rules can used to instantiate the metavariables by unification.

These ideas are enough to build a tableau calculus for first-order logic.

▷ Unification is an efficient decision procdure for finding substitutions that make first-order terms (syntactically) equal.

▷ In prenex normal form, all quantifiers are up front. In Skolem normal form, additionally there are no existential quantifiers. In claus normal form, additionally the formula is in CNF.

▷ Any PL^1 formula can efficiently be brought into a satisfiability-equivalent clause normal form.

▷ This allows first-order resolution.

Chapter 16

Knowledge Representation and the Semantic Web

The field of "Knowledge Representation" is usually taken to be an area in Artificial Intelligence that studies the representation of knowledge in formal systems and how to leverage inference techniques to generate new knowledge items from existing ones. Note that this definition coincides with with what we know as logical systems in this course. This is the view taken by the subfield of "description logics", but restricted to the case, where the logical systems have an entailment relation to ensure applicability. This chapter is organized as follows. We will first give a general introduction to the concepts of knowledge representation using semantic networks – an early and very intuitive approach to knowledge representation – as an object-to-think-with. In **??** we introduce the principles and services of logic-based knowledge-representation using a non-standard interpretation of propositional logic as the basis, this gives us a formal account of the taxonomic part of semantic networks. In **??** we introduce the logic $\mathcal{ALC}$ that adds relations (called "roles") and restricted quantification and thus gives us the full expressive power of semantic networks. Thus $\mathcal{ALC}$ can be seen as a prototype description logic. In **??** we show how description logics are applied as the basis of the "semantic web".

16.1 Introduction to Knowledge Representation

A Video Nugget covering the introduction to knowledge representation can be found at `https://fau.tv/clip/id/27279`.

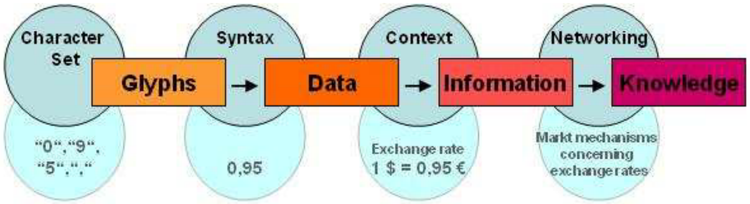
Before we start into the development of description logics, we set the stage by looking into alternatives for knowledge representation.

16.1.1 Knowledge & Representation

To approach the question of knowledge representation, we first have to ask ourselves, what knowledge might be. This is a difficult question that has kept philosophers occupied for millennia. We will not answer this question in this course, but only allude to and discuss some aspects that are relevant to our cause of knowledge representation.

What is knowledge? Why Representation?

- ▷ Lots/all of (academic) disciplines deal with knowledge!
- ▷ According to Probst/Raub/Romhardt [PRR97]



▷ **For the purposes of this course:** Knowledge is the information necessary to support intelligent reasoning!

representation	can be used to determine
set of words	whether a word is admissible
list of words	the rank of a word
a lexicon	translation and/or grammatical function
structure	function

According to an influential view of [PRR97], knowledge appears in layers. Staring with a character set that defines a set of glyphs, we can add syntax that turns mere strings into data. Adding context information gives information, and finally, by relating the information to other information allows to draw conclusions, turning information into knowledge.

Note that we already have aspects of representation and function in the diagram at the top of the slide. In this, the additional functionaltiy added in the successive layers gives the representations more and more functions, until we reach the knowledge level, where the function is given by inferencing. In the second example, we can see that representations determine possible functions.

Let us now strengthen our intuition about knowledge by contrasting knowledge representations from "regular" data structures in computation.

Knowledge Representation vs. Data Structures

▷ **Idea:** Representation as structure and function.

- ▷ the representation determines the content theory (what is the data?)
- ▷ the function determines the process model (what do we do with the data?)

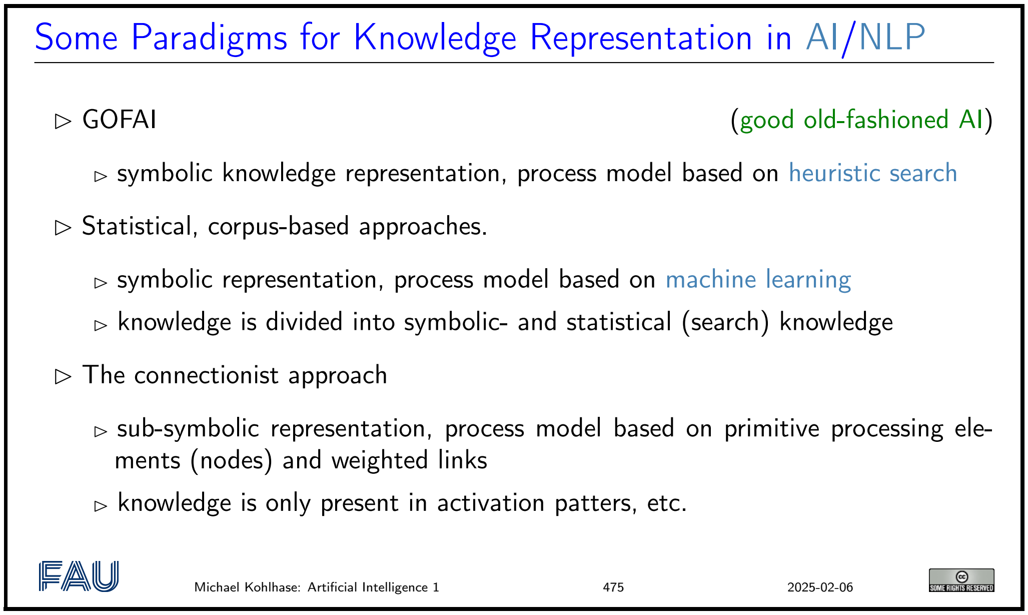
▷ **Question:** Why do we use the term "knowledge representation" rather than

- ▷ data structures? (sets, lists, ... above)
- ▷ information representation? (it is information)

▷ **Answer:** No good reason other than AI practice, with the intuition that

- ▷ data is simple and general (supports many algorithms)
- ▷ knowledge is complex (has distinguished process model)

As knowledge is such a central notion in artificial intelligence, it is not surprising that there are multiple approaches to dealing with it. We will only deal with the first one and leave the others to self-study.

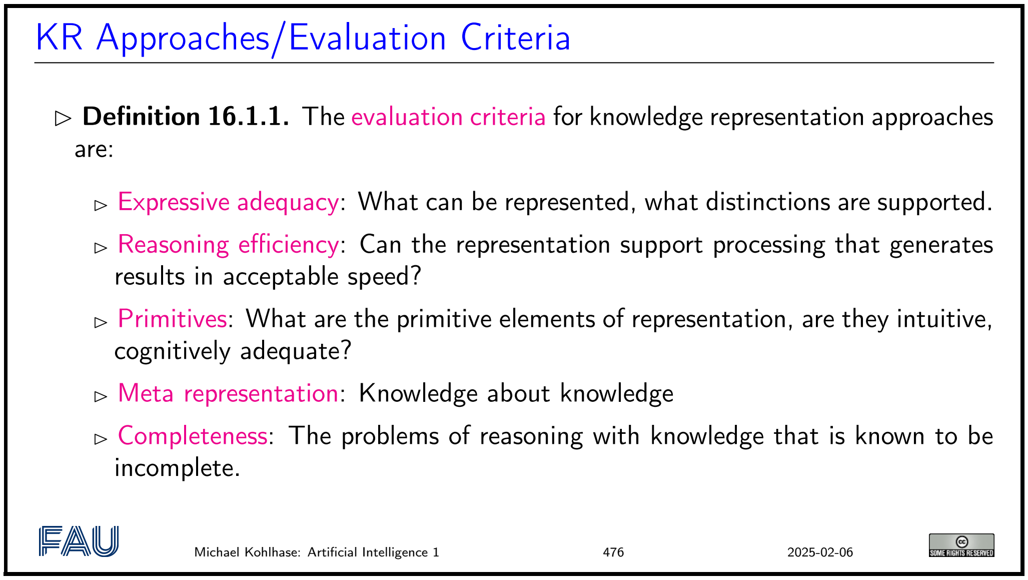


Some Paradigms for Knowledge Representation in AI/NLP

- ▷ GOFAI (good old-fashioned AI)
  - ▹ symbolic knowledge representation, process model based on heuristic search
- ▷ Statistical, corpus-based approaches.
  - ▹ symbolic representation, process model based on machine learning
  - ▹ knowledge is divided into symbolic- and statistical (search) knowledge
- ▷ The connectionist approach
  - ▹ sub-symbolic representation, process model based on primitive processing elements (nodes) and weighted links
  - ▹ knowledge is only present in activation patters, etc.

FAU Michael Kohlhase: Artificial Intelligence 1 475 2025-02-06

When assessing the relative strengths of the respective approaches, we should evaluate them with respect to a pre-determined set of criteria.



KR Approaches/Evaluation Criteria

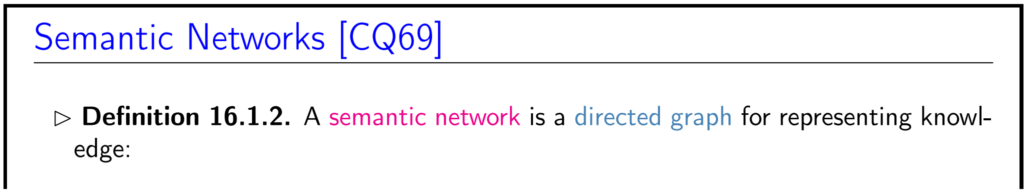
- ▷ **Definition 16.1.1.** The evaluation criteria for knowledge representation approaches are:
  - ▹ Expressive adequacy: What can be represented, what distinctions are supported.
  - ▹ Reasoning efficiency: Can the representation support processing that generates results in acceptable speed?
  - ▹ Primitives: What are the primitive elements of representation, are they intuitive, cognitively adequate?
  - ▹ Meta representation: Knowledge about knowledge
  - ▹ Completeness: The problems of reasoning with knowledge that is known to be incomplete.

FAU Michael Kohlhase: Artificial Intelligence 1 476 2025-02-06

16.1.2 Semantic Networks

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/27280`.

To get a feeling for early knowledge representation approaches from which description logics developed, we take a look at "semantic networks" and contrast them to logical approaches. Semantic networks are a very simple way of arranging knowledge about objects and concepts and their relationships in a graph.

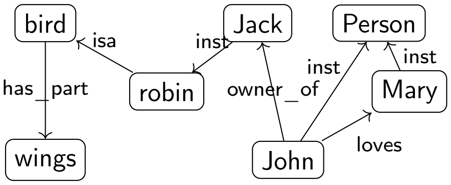


Semantic Networks [CQ69]

- ▷ **Definition 16.1.2.** A semantic network is a directed graph for representing knowledge:

- nodes represent objects and concepts (classes of objects) (e.g. John (object) and bird (concept))
- edges (called links) represent relations between these (isa, father_of, belongs_to)

▷ **Example 16.1.3.** A semantic network for birds and persons:



▷ **Problem:** How do we derive new information from such a network?

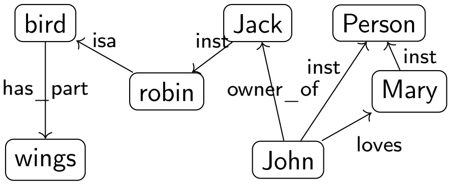
▷ **Idea:** Encode taxonomic information about objects and concepts in special links (“isa” and “inst”) and specify property inheritance along them in the process model.

Even though the network in **??** is very intuitive (we immediately understand the concepts depicted), it is unclear how we (and more importantly a machine that does not associate meaning with the labels of the nodes and edges) can draw inferences from the “knowledge” represented.

Deriving Knowledge Implicit in Semantic Networks

▷ **Observation 16.1.4.** *There is more knowledge in a semantic network than is explicitly written down.*

▷ **Example 16.1.5.** In the network below, we “know” that *robins have wings* and in particular, *Jack has wings*.



▷ **Idea:** Links labeled with “isa” and “inst” are special: they propagate properties encoded by other links.

▷ **Definition 16.1.6.** We call links labeled by

- “isa” an inclusion or isa link (inclusion of concepts)
- “inst” instance or inst link (concept membership)

We now make the idea of “propagating properties” rigorous by defining the notion of derived relations, i.e. the relations that are left implicit in the network, but can be added without changing its meaning.

Deriving Knowledge Semantic Networks

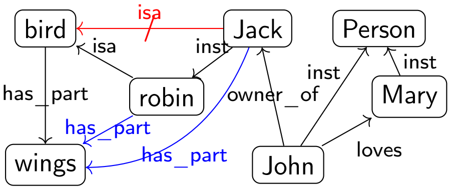
- ▷ **Definition 16.1.7 (Inference in Semantic Networks).** We call all link labels except "inst" and "isa" in a semantic network relations.

  Let N be a semantic network and R a relation in N such that $A \xrightarrow{\text{isa}} B \xrightarrow{R} C$ or $A \xrightarrow{\text{inst}} B \xrightarrow{R} C$, then we can derive a relation $A \xrightarrow{R} C$ in N.

  The process of deriving new concepts and relations from existing ones is called inference and concepts/relations that are only available via inference implicit (in a semantic network).

- ▷ **Intuition:** Derived relations represent knowledge that is implicit in the network; they could be added, but usually are not to avoid clutter.

- ▷ **Example 16.1.8.** Derived relations in ??



- ▷ **Slogan:** Get out more knowledge from a semantic networks than you put in.

Note that **??** does not quite allow to derive that *Jack is a bird* (did you spot that "isa" is not a relation that can be inferred?), even though we know it is true in the world. This shows us that inference in semantic networks has be to very carefully defined and may not be "complete", i.e. there are things that are true in the real world that our inference procedure does not capture.

Dually, if we are not careful, then the inference procedure might derive properties that are not true in the real world even if all the properties explicitly put into the network are. We call such an inference procedure unsound or incorrect.

These are two general phenomena we have to keep an eye on.

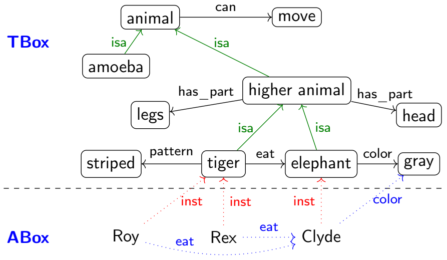
Another problem is that semantic networks (e.g. in **??**) confuse two kinds of concepts: individuals (represented by proper names like *John* and *Jack*) and concepts (nouns like *robin* and *bird*). Even though the isa and inst link already acknowledge this distinction, the "has_part" and "loves" relations are at different levels entirely, but not distinguished in the networks.

Terminologies and Assertions

- ▷ *Remark 16.1.9.* We should distinguish concepts from objects.

- ▷ **Definition 16.1.10.** We call the subgraph of a semantic network N spanned by the isa links and relations between concepts the terminology (or TBox, or the famous Isa Hierarchy) and the subgraph spanned by the inst links and relations between objects, the assertions (together the ABox) of N.

- ▷ **Example 16.1.11.** In this semantic network we keep objects concept apart notationally:



In particular we have objects "Rex", "Roy", and "Clyde", which have (derived) relations (e.g. *Clyde* is *gray*).

But there are severe shortcomings of semantic networks: the suggestive shape and node names give (humans) a false sense of meaning, and the inference rules are only given in the process model (the implementation of the semantic network processing system).

This makes it very difficult to assess the strength of the inference system and make assertions e.g. about completeness.

Limitations of Semantic Networks

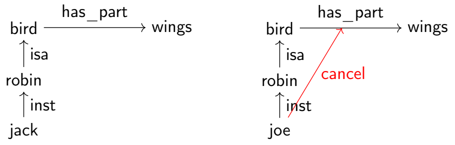
▷ What is the meaning of a link?

  ▹ link labels are very suggestive (misleading for humans)

  ▹ meaning of link types defined in the process model (no denotational semantics)

▷ **Problem:** No distinction of optional and defining traits!

▷ **Example 16.1.12.** Consider a robin that has lost its wings in an accident:



"Cancel-links" have been proposed, but their status and process model are debatable.

To alleviate the perceived drawbacks of semantic networks, we can contemplate another notation that is more linear and thus more easily implemented: function/argument notation.

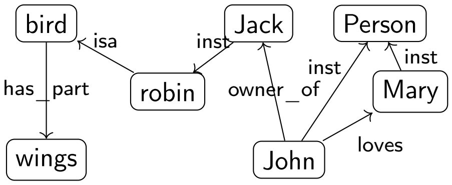
Another Notation for Semantic Networks

▷ **Definition 16.1.13.** Function/argument notation for semantic networks

  ▹ interprets nodes as arguments (reification to individuals)

  ▹ interprets links as functions (predicates actually)

▷ **Example 16.1.14.**



isa(robin,bird)
haspart(bird,wings)
inst(Jack,robin)
owner_of(John, robin)
loves(John,Mary)

- ▷ **Evaluation:**
  - + linear notation (equivalent, but better to implement on a computer)
  - + easy to give process model by deduction (e.g. in Prolog)
  - – worse locality properties (networks are associative)

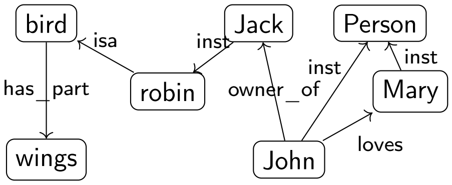
Indeed the function/argument notation is the immediate idea how one would naturally represent semantic networks for implementation.

This notation has been also characterized as subject/predicate/object triples, alluding to simple (English) sentences. This will play a role in the “semantic web” later.

Building on the function/argument notation from above, we can now give a formal semantics for semantic network: we translate them into first-order logic and use the semantics of that.

A Denotational Semantics for Semantic Networks

▷ **Observation:** If we handle isa and inst links specially in function/argument notation



robin $\subseteq$ bird
haspart(bird,wings)
Jack $\in$ robin
owner_of(John, Jack)
loves(John,Mary)

it looks like first-order logic, if we take

- ▷ $a \in S$ to mean $S(a)$ for an object a and a concept S.
- ▷ $A \subseteq B$ to mean $\forall X.A(X) \Rightarrow B(X)$ and concepts A and B
- ▷ $R(A, B)$ to mean $\forall X.A(X) \Rightarrow (\exists Y.B(Y) \wedge R(X, Y))$ for a relation R.

▷ **Idea:** Take first-order deduction as process model (gives inheritance for free)

Indeed, the semantics induced by the translation to first-order logic, gives the intuitive meaning to the semantic networks. Note that this only holds only for the features of semantic networks that are representable in this way, e.g. the “cancel links” shown above are not (and that is a feature, not a bug).

But even more importantly, the translation to first-order logic gives a first process model: we can use first-order inference to compute the set of inferences that can be drawn from a semantic network.

Before we go on, let us have a look at an important application of knowledge representation technologies: the semantic web.

16.1.3 The Semantic Web

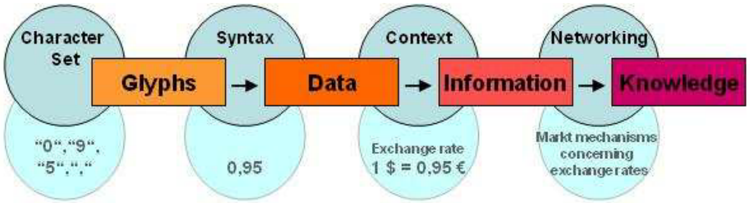
A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/27281`. We will now define the term semantic web and discuss the pertinent ideas involved. There are two central ones, we will cover here:

- Information and data come in different levels of explicitness; this is usually visualized by a "ladder" of information.
- if information is sufficiently machine-understandable, then we can automate drawing conclusions.

The Semantic Web

▷ **Definition 16.1.15.** The semantic web is the result including of semantic content in web pages with the aim of converting the WWW into a machine-understandable "web of data", where inference based services can add value to the ecosystem.

▷ **Idea:** Move web content up the ladder, use inference to make connections.



▷ **Example 16.1.16.** Information not explicitly represented (in one place)

Query: *Who was US president when Barak Obama was born?*

Google: *...BIRTH DATE: August 04, 1961...*

Query: *Who was US president in 1961?*

Google: *President: Dwight D. Eisenhower [...] John F. Kennedy (starting Jan. 20.)*

Humans understand the text and combine the information to get the answer. Machines need more than just text ⇝ semantic web technology.

The term "semantic web" was coined by Tim Berners Lee in analogy to semantic networks, only applied to the world wide web. And as for semantic networks, where we have inference processes that allow us the recover information that is not explicitly represented from the network (here the world-wide-web).

To see that problems have to be solved, to arrive at the semantic web, we will now look at a concrete example about the "semantics" in web pages. Here is one that looks typical enough.

What is the Information a User sees?

▷ **Example 16.1.17.** Take the following web-site with a conference announcement

WWW2002
The eleventh International World Wide Web Conference
Sheraton Waikiki Hotel
Honolulu, Hawaii, USA

7-11 May 2002

Registered participants coming from
Australia, Canada, Chile Denmark, France, Germany, Ghana, Hong Kong, India,
Ireland, Italy, Japan, Malta, New Zealand, The Netherlands, Norway,
Singapore, Switzerland, the United Kingdom, the United States, Vietnam, Zaire

On the 7th May Honolulu will provide the backdrop of the eleventh International World Wide Web Conference.

Speakers confirmed
Tim Berners-Lee: Tim is the well known inventor of the Web,
Ian Foster: Ian is the pioneer of the Grid, the next generation internet.

But as for semantic networks, what you as a human can see (“understand” really) is deceptive, so let us obfuscate the document to confuse your “semantic processor”. This gives an impression of what the computer “sees”.

What the machine sees

▷ **Example 16.1.18.** Here is what the machine “sees” from the conference announcement:

𝒲𝒲𝒲∈′′∈
𝒯⟨⌉⌉↕⌉⊑⌉\⊔⟨ℐ\⊔⌉∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈⌉𝒲⌉⌊𝒞≀\{⌉∇⌉\⌋⌉
𝒮⟨⌉∇⊣⊔≀\𝒲⊣⟩‖⟩‖⟩ℋ≀⊔⌉↕
ℋ≀\≀↕⊓↕⊓⇔ℋ⊣⊒⊣⟩⟩⇔𝒰𝒮𝒜
↖∞∞ℳ⊣†∈′′∈

ℛ⌉}⟩∫⊔⌉∇⌉⌈√⊣∇⊔⟩⌋⟩√⊣\⊔∫⌋≀⇕⟩\}{∇≀⇕
𝒜⊓∫⊔∇⊣↕⟩⊣⇔𝒞⊣\⊣⌈⊣⇔𝒞⟨⟩↕⌉𝒟⌉\⇕⊣∇‖⇔ℱ∇⊣\⌋⌉⇔𝒢⌉∇⇕⊣\†⇔𝒢⟨⊣\⊣⇔ℋ≀\}𝒦≀\}⇔ℐ\⌈⟩⊣⇔
ℐ∇⌉↕⊣\⌈⇔ℐ⊔⊣↕†⇔𝒥⊣√⊣\⇔ℳ⊣↕⊔⊣⇔𝒩⌉⊒𝒵⌉⊣↕⊣\⌈⇔𝒯⟨⌉𝒩⌉⊔⟨⌉∇↕⊣\⌈∫⇔𝒩≀∇⊒⊣†⇔
𝒮⟩\}⊣√≀∇⌉⇔𝒮⊒⟩⊔‡⌉∇↕⊣\⌈⇔⊔⟨⌉𝒰\⟩⊔⌉⌈𝒦⟩\}⌈≀⇕⇔⊔⟨⌉𝒰\⟩⊔⌉⌈𝒮⊔⊣⊔⌉∫⇔𝒱⟩⌉⊔\⊣⇕⇔𝒵⊣⟩∇⌉

𝒪\⊔⟨⌉⊔⟨ℳ⊣†ℋ≀\≀↕⊓↕⊓⊒⟩↕↕√∇≀⊑⟩⌈⌉⊔⟨⌉⌊⊣⌋‖⌈∇≀√≀{⊔⟨⌉⌉↕⌉⊑⌉\⊔⟨
ℐ\⊔⌉∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈⌉𝒲⌉⌊𝒞≀\{⌉∇⌉\⌋⌉↙

𝒮√⌉⊣‖⌉∇∫⌋≀\{⟩∇⇕⌉⌈
𝒯⟩⇕ℬ⌉∇\⌉∇∫↖ℒ⌉⌉¬𝒯⟩⇕⟩∫⊔⟨⌉⊒⌉↕↕‖\≀⊒\⟩\⊑⌉\⊔≀∇≀{⊔⟨⌉𝒲⌉⌊⇔
ℐ⊣\ℱ≀∫⊔⌉∇¬ℐ⊣\⟩∫⊔⟨⌉√⟩≀\⌉⌉∇≀{⊔⟨⌉𝒢∇⟩⌈⇔⊔⟨⌉\⌉§⊔}⌉\⌉∇⊣⊔⟩≀\⟩\⊔⌉∇\⌉⊔↙

Obviously, there is not much the computer understands, and as a consequence, there is not a lot the computer can support the reader with. So we have to “help” the computer by providing some meaning. Conventional wisdom is that we add some semantic/functional markup. Here we pick XML without loss of generality, and characterize some fragments of text e.g. as dates.

Solution: XML markup with “meaningful” Tags

▷ **Example 16.1.19.** Let’s annotate (parts of) the meaning via XML markup

```
<title>𝒲𝒲𝒲∈′′∈
𝒯⟨⌉⌉↕⌉⊑⌉\⊔⟨ℐ\⊔⌉∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈⌉𝒲⌉⌊𝒞≀\{⌉∇⌉\⌋⌉</title>
<place>𝒮⟨⌉∇⊣⊔≀\𝒲⊣⟩‖⟩‖⟩ℋ≀⊔⌉↕ℋ≀\≀↕⊓↕⊓⇔ℋ⊣⊒⊣⟩⟩⇔𝒰𝒮𝒜</place>
<date>↖∞∞ℳ⊣†∈′′∈</date>
<participants>ℛ⌉}⟩∫⊔⌉∇⌉⌈√⊣∇⊔⟩⌋⟩√⊣\⊔∫⌋≀⇕⟩\}{∇≀⇕
𝒜⊓∫⊔∇⊣↕⟩⊣⇔𝒞⊣\⊣⌈⊣⇔𝒞⟨⟩↕⌉𝒟⌉\⇕⊣∇‖⇔ℱ∇⊣\⌋⌉⇔𝒢⌉∇⇕⊣\†⇔𝒢⟨⊣\⊣⇔ℋ≀\}𝒦≀\}⇔ℐ\⌈⟩⊣⇔
ℐ∇⌉↕⊣\⌈⇔ℐ⊔⊣↕†⇔𝒥⊣√⊣\⇔ℳ⊣↕⊔⊣⇔𝒩⌉⊒𝒵⌉⊣↕⊣\⌈⇔𝒯⟨⌉𝒩⌉⊔⟨⌉∇↕⊣\⌈∫⇔𝒩≀∇⊒⊣†⇔
𝒮⟩\}⊣√≀∇⌉⇔𝒮⊒⟩⊔‡⌉∇↕⊣\⌈⇔⊔⟨⌉𝒰\⟩⊔⌉⌈𝒦⟩\}⌈≀⇕⇔⊔⟨⌉𝒰\⟩⊔⌉⌈𝒮⊔⊣⊔⌉∫⇔𝒱⟩⌉⊔\⊣⇕⇔𝒵⊣⟩∇⌉
</participants>
<introduction>𝒪\⊔⟨⌉⊔⟨ℳ⊣†ℋ≀\≀↕⊓↕⊓⊒⟩↕↕√∇≀⊑⟩⌈⌉⊔⟨⌉⌊⊣⌋‖⌈∇≀√≀{⊔⟨⌉⌉↕⌉⊑⌉\⊔⟨ℐ\⊔⌉∇↖
\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈⌉𝒲⌉⌊𝒞≀\{⌉∇⌉\⌋⌉↙</introduction>
<program>𝒮√⌉⊣‖⌉∇∫⌋≀\{⟩∇⇕⌉⌈
<speaker>𝒯⟩⇕ℬ⌉∇\⌉∇∫↖ℒ⌉⌉¬𝒯⟩⇕⟩∫⊔⟨⌉⊒⌉↕↕‖\≀⊒\⟩\⊑⌉\⊔≀∇≀{⊔⟨⌉𝒲⌉⌊</speaker>
<speaker>ℐ⊣\ℱ≀∫⊔⌉∇¬ℐ⊣\⟩∫⊔⟨⌉√⟩≀\⌉⌉∇≀{⊔⟨⌉𝒢∇⟩⌈⇔⊔⟨⌉\⌉§⊔}⌉\⌉∇⊣⊔⟩≀\⟩\⊔⌉∇\⌉⊔<speaker>
</program>
```

FAU Michael Kohlhase: Artificial Intelligence 1 487 2025-02-06

But does this really help? Is conventional wisdom correct?

What can we do with this?

▷ **Example 16.1.20.** Consider the following fragments:

```
ℜ⊔⟩⊔↕⌉⊤𝒲𝒲𝒲∈′′∈
𝒯⟨⌉⌉↕⌉⊑⌉\⊔⟨ℐ\⊔⌉∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈⌉𝒲⌉⌊𝒞≀\{⌉∇⌉\⌋⌉ℜ∝⊔⟩⊔↕⌉⊤
ℜ√↕⊣⌋⌉⊤𝒮⟨⌉∇⊣⊔≀\𝒲⊣⟩‖⟩‖⟩ℋ≀⊔⌉↕ℋ≀\≀↕⊓↕⊓⇔ℋ⊣⊒⊣⟩⟩⇔𝒰𝒮𝒜ℜ∝√↕⊣⌋⌉⊤
ℜ⌈⊣⊔⌉⊤↖∞∞ℳ⊣†∈′′∈ℜ∝⌈⊣⊔⌉⊤
```

Given the markup above, a machine agent can

- ▷ parse ∞∞ℳ⊣†∈′′∈ as the date May 7 11 2002 and add this to the user’s calendar,
- ▷ parse 𝒮⟨⌉∇⊣⊔≀\𝒲⊣⟩‖⟩‖⟩ℋ≀⊔⌉↕ℋ≀\≀↕⊓↕⊓⇔ℋ⊣⊒⊣⟩⟩⇔𝒰𝒮𝒜 as a destination and find flights.

▷ **But:** do not be deceived by your ability to understand English!

FAU Michael Kohlhase: Artificial Intelligence 1 488 2025-02-06

To understand what a machine can understand we have to obfuscate the markup as well, since it does not carry any intrinsic meaning to the machine either.

What the machine sees of the XML

▷ **Example 16.1.21.** Here is what the machine sees of the XML

```
<title>𝒲𝒲𝒲∈′′∈
𝒯⟨⌉⌉↕⌉⊑⌉\⊔⟨ℐ\⊔⌉∇\⊣⊔⟩≀\⊣↕𝒲≀∇↕⌈𝒲⟩⌈⌉𝒲⌉⌊𝒞≀\{⌉∇⌉\⌋⌉</⊔⟩⊔↕⌉>
<√‡⊣|]>𝒮⟨]∇⊣⊔≀\𝒲⊣⟩∥⟩∥⟩ℋ≀⊔]‡ℋ≀\≀‡⊓‡⊓⇔ℋ⊣⊒⊣⟩⟩⇔𝒰𝒮𝒜</√‡⊣|]>
<⌈⊣⊔]>↖∞∞ℳ⊣†∈′′∈</⌈⊣⊔]>
<√⊣∇⊔⟩⌋⟩√⊣\⊔∫>ℛ]}⟩∫⊔]∇]⌈√⊣∇⊔⟩⌋⟩√⊣\⊔∫]≀‡⟩\}{∇≀‡
𝒜⊓∫⊔∇⊣‡⟩⊣⇔𝒞⊣\⊣⌈⊣⇔𝒞⟨⟩‡]𝒟]\‡⊣∇∥⇔ℱ∇⊣\⌋]⇔𝒢]∇‡⊣\†⇔𝒢⟨⊣\⊣⇔ℋ≀\}𝒦≀\}⇔ℐ\⌈⟩⊣⇔
ℐ∇]‡⊣\⌈⇔ℐ⊔⊣‡†⇔𝒥⊣√⊣\⇔ℳ⊣‡⊔⊣⇔𝒩]⊒𝒵]⊣‡⊣\⌈⇔𝒯⟨]𝒩]⊔⟨]∇‡⊣\⌈∫⇔𝒩≀∇⊒⊣†⇔
𝒮⟩\}⊣√≀∇]⇔𝒮⊒⟩⊔‡]∇‡⊣\⌈⇔⊔⟨]𝒰\⟩⊔]⌈𝒦⟩\}⌈≀‡⇔⊔⟨]𝒰\⟩⊔]⌈𝒮⊔⊣⊔]∫⇔𝒱⟩]⊔\⊣‡⇔𝒵⊣⟩∇]
</√⊣∇⊔⟩⌋⟩√⊣\⊔∫>
<⟩\⊔∇≀⌈⊓⌋⊔⟩≀\>𝒪\⊔⟨]⊔⟨ℳ⊣†ℋ≀\≀‡⊓‡⊓⊒⟩‡‡√∇≀⊑⟩⌈]⊔⟨]⌊⊣⌋∥⌈∇≀√≀{⊔⟨]]‡]⊑]\⊔⟨ℐ\⊔]∇\⊣↖
⊔⟩≀\⊣‡𝒲≀∇‡⌈𝒲⟩⌈]𝒲]⌊𝒞≀\{]∇]\⌋]↙</⟩\⊔∇≀⌈⊓⌋⊔⟩≀\>
<√∇≀}∇⊣‡>𝒮√]⊣∥]∇∫⌋≀\{⟩∇‡]⌈
<∫√]⊣∥]∇>𝒯⟩‡ℬ]∇\]∇∫↖ℒ]]¬𝒯⟩‡⟩∫⊔⟨]⊒]‡‡∥\≀⊒\⟩\⊑]\⊔≀∇≀{⊔⟨]𝒲]⌊</∫√]⊣∥]∇>
<∫√]⊣∥]∇>ℐ⊣\ℱ≀∫⊔]∇¬ℐ⊣\⟩∫⊔⟨]√⟩≀\]]∇≀{⊔⟨]𝒢∇⟩⌈⇔⊔⟨]\§⊔}]\]∇⊣⊔⟩≀\⟩\⊔]∇\]⊔<∫√]⊣∥]∇>
</√∇≀}∇⊣‡>
```

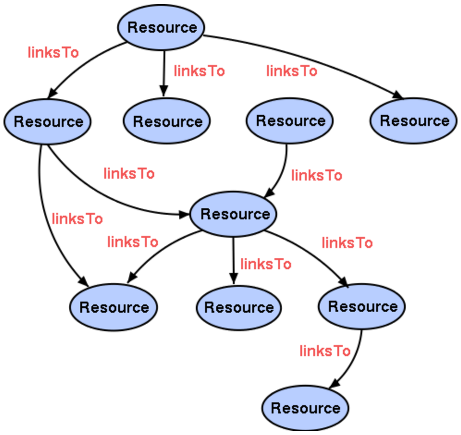
Michael Kohlhase: Artificial Intelligence 1 489 2025-02-06

So we have not really gained much either with the markup, we really have to give meaning to the markup as well, this is where techniques from semenatic web come into play.

To understand how we can make the web more semantic, let us first take stock of the current status of (markup on) the web. It is well-known that world-wide-web is a hypertext, where multimedia documents (text, images, videos, etc. and their fragments) are connected by hyperlinks. As we have seen, all of these are largely opaque (non-understandable), so we end up with the following situation (from the viewpoint of a machine).

The Current Web

- ▷ **Resources:** identified by URIs, untyped
- ▷ **Links:** href, src, ...limited, non-descriptive
- ▷ **User:** Exciting world - semantics of the resource, however, gleaned from content
- ▷ **Machine:** Very little information available - significance of the links only evident from the context around the anchor.

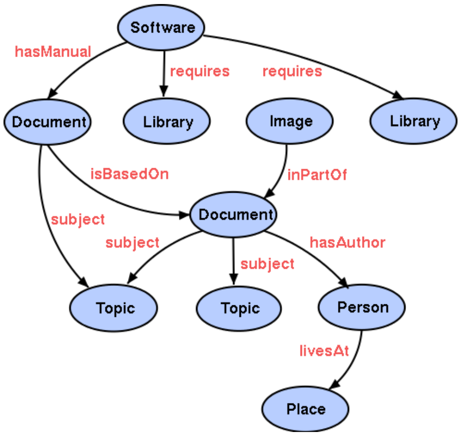


Michael Kohlhase: Artificial Intelligence 1 490 2025-02-06

Let us now contrast this with the envisioned semantic web.

The Semantic Web

- ▷ **Resources:** Globally identified by URIs or Locally scoped (Blank), Extensible, Relational.
- ▷ **Links:** Identified by URIs, Extensible, Relational.
- ▷ **User:** Even more exciting world, richer user experience.
- ▷ **Machine:** More processable information is available (Data Web).
- ▷ **Computers and people:** Work, learn and exchange knowledge effectively.



Essentially, to make the web more machine-processable, we need to classify the resources by the concepts they represent and give the links a meaning in a way, that we can do inference with that. The ideas presented here gave rise to a set of technologies jointly called the "semantic web", which we will now summarize before we return to our logical investigations of knowledge representation techniques.

Towards a "Machine-Actionable Web"

- ▷ **Recall:** We need external agreement on meaning of annotation tags.
- ▷ **Idea:** standardize them in a community process (e.g. DIN or ISO)
- ▷ **Problem:** Inflexible, Limited number of things can be expressed
- ▷ **Better:** Use ontologies to specify meaning of annotations
  - ▷ Ontologies provide a vocabulary of terms
  - ▷ New terms can be formed by combining existing ones
  - ▷ Meaning (semantics) of such terms is formally specified
  - ▷ Can also specify relationships between terms in multiple ontologies
- ▷ Inference with annotations and ontologies (get out more than you put in!)
  - ▷ Standardize annotations in RDF [KC04] or RDFa [Her+13b] and ontologies on OWL [OWL09]
  - ▷ Harvest RDF and RDFa in to a triplestore or OWL reasoner.
  - ▷ Query that for implied knowledge (e.g. chaining multiple facts from Wikipedia)

    SPARQL: Who was US President when Barack Obama was Born?

    DBPedia: John F. Kennedy (was president in August 1961)

16.1.4 Other Knowledge Representation Approaches

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/27282`.

Now that we know what semantic networks mean, let us look at a couple of other approaches that were influential for the development of knowledge representation. We will just mention them for reference here, but not cover them in any depth.

Frame Notation as Logic with Locality

▷ Predicate Logic: (where is the locality?)

$catch\_22 \in catch\_object$	There is an instance of catching
$catcher(catch\_22, jack\_2)$	Jack did the catching
$caught(catch\_22, ball\_5)$	He caught a certain ball

▷ **Definition 16.1.22.** Frames (group everything around the object)

```
(catch_object catch_22
              (catcher jack_2)
              (caught ball_5))
```

+ Once you have decided on a frame, all the information is local

+ easy to define schemes for concept (aka. types in feature structures)

– how to determine frame, when to choose frame (log/chair)

KR involving Time (Scripts [Shank '77])

▷ **Idea:** Organize typical event sequences, actors and props into representation.

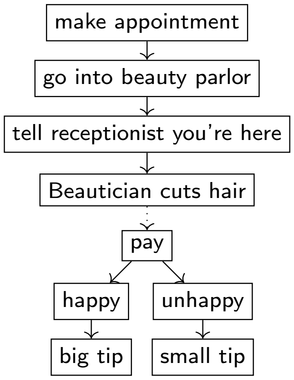
▷ **Definition 16.1.23.** A script is a structured representation describing a stereotyped sequence of events in a particular context. Structurally, scripts are very much like frames, except the values that fill the slots must be ordered.

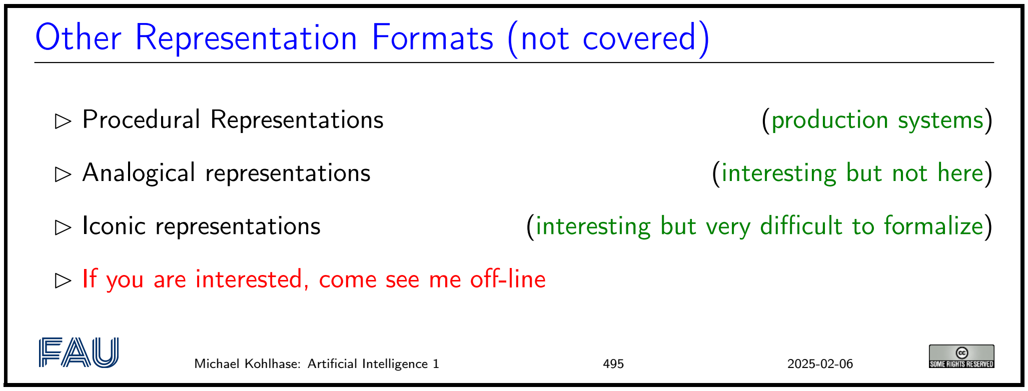
▷ **Example 16.1.24.** getting your hair cut (at a beauty parlor)

- ▷ props, actors as "script variables"
- ▷ events in a (generalized) sequence

▷ use script material for

- ▷ anaphora, bridging references
- ▷ default common ground
- ▷ to fill in missing material into situations





Other Representation Formats (not covered)

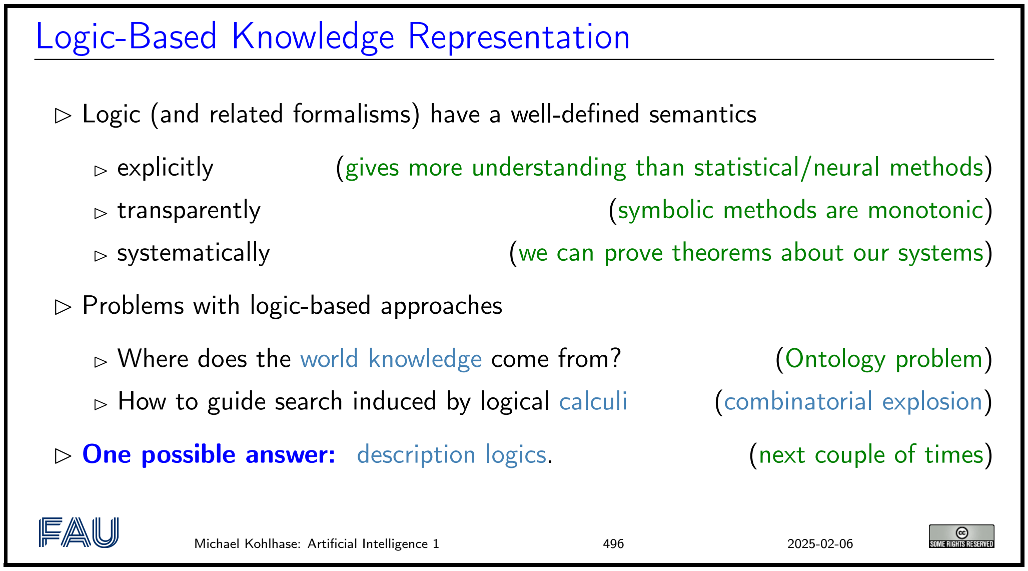
- ▷ Procedural Representations (production systems)
- ▷ Analogical representations (interesting but not here)
- ▷ Iconic representations (interesting but very difficult to formalize)
- ▷ If you are interested, come see me off-line

Michael Kohlhase: Artificial Intelligence 1 495 2025-02-06

16.2 Logic-Based Knowledge Representation

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/27297`.

We now turn to knowledge representation approaches that are based on some kind of logical system. These have the advantage that we know exactly what we are doing: as they are based on symbolic representations and declaratively given inference calculi as process models, we can inspect them thoroughly and even prove facts about them.



Logic-Based Knowledge Representation

- ▷ Logic (and related formalisms) have a well-defined semantics
  - ▷ explicitly (gives more understanding than statistical/neural methods)
  - ▷ transparently (symbolic methods are monotonic)
  - ▷ systematically (we can prove theorems about our systems)
- ▷ Problems with logic-based approaches
  - ▷ Where does the world knowledge come from? (Ontology problem)
  - ▷ How to guide search induced by logical calculi (combinatorial explosion)
- ▷ **One possible answer:** description logics. (next couple of times)

Michael Kohlhase: Artificial Intelligence 1 496 2025-02-06

But of course logic-based approaches have big drawbacks as well. The first is that we have to obtain the symbolic representations of knowledge to do anything – a non-trivial challenge, since most knowledge does not exist in this form in the wild, to obtain it, some agent has to experience the word, pass it through its cognitive apparatus, conceptualize the phenomena involved, systematize them sufficiently to form symbols, and then represent those in the respective formalism at hand.

The second drawback is that the process models induced by logic-based approaches (inference with calculi) are quite intractable. We will see that all inferences can be played back to satisfiability tests in the underlying logical system, which are exponential at best, and undecidable or even incomplete at worst.

Therefore a major thrust in logic-based knowledge representation is to investigate logical systems that are expressive enough to be able to represent most knowledge, but still have a decidable – and maybe even tractable in practice – satisfiability problem. Such logics are called “description logics”. We will study the basics of such logical systems and their inference procedures in the following.

16.2.1 Propositional Logic as a Set Description Language

Before we look at "real" description logics in ??, we will make a "dry run" with a logic we already understand: propositional logic, which we will re-interpret the system as a set description language by giving a new, non-standard semantics. This allows us to already preview most of the inference procedures and knowledge services of knowledge representation systems in the next subsection.
To establish propositional logic as a set description language, we use a different interpretation than usual. We interpret propositional variables as names of sets and the connectives as set operations, which is why we give them a different – more suggestive – syntax.

Propositional Logic as Set Description Language

▷ **Idea:** Use propositional logic as a set description language: (variant syntax/semantics)

▷ **Definition 16.2.1.** Let $\mathrm{PL}^0_{\mathrm{DL}}$ be given by the following grammar for the $\mathrm{PL}^0_{\mathrm{DL}}$ concepts. (formulae)

$$\mathcal{L}::=C \mid \top \mid \bot \mid \overline{\mathcal{L}} \mid \mathcal{L}\sqcap\mathcal{L} \mid \mathcal{L}\sqcup\mathcal{L} \mid \mathcal{L}\sqsubseteq\mathcal{L} \mid \mathcal{L}\equiv\mathcal{L}$$

i.e. $\mathrm{PL}^0_{\mathrm{DL}}$ formed from

- ▷ atomic formulae ($\widehat{=}$ propositional variables)
- ▷ concept intersection ($\sqcap$) ($\widehat{=}$ conjunction $\wedge$)
- ▷ concept complement ($\overline{\cdot}$) ($\widehat{=}$ negation $\neg$)
- ▷ concept union ($\sqcup$), subsumption ($\sqsubseteq$), and equivalence ($\equiv$) defined from these. ($\widehat{=}$ $\vee$, $\Rightarrow$, and $\Leftrightarrow$)

▷ **Definition 16.2.2 (Formal Semantics).** Let $\mathcal{D}$ be a given set (called the domain of discourse) and $\varphi\colon \mathcal{V}_0 \to \mathcal{P}(\mathcal{D})$, then we define

- ▷ $[\![P]\!] := \varphi(P)$, (remember $\varphi(P) \subseteq \mathcal{D}$).
- ▷ $[\![\mathbf{A}\sqcap\mathbf{B}]\!] := [\![\mathbf{A}]\!] \cap [\![\mathbf{B}]\!]$ and $[\![\overline{\mathbf{A}}]\!] := \mathcal{D}\backslash[\![\mathbf{A}]\!]$ ...

We call this construction the set description semantics of PL^0.

▷ **Note:** $\langle \mathrm{PL}^0_{\mathrm{DL}}, \mathcal{S}, [\![\cdot]\!] \rangle$, where $\mathcal{S}$ is the class of possible domains forms a logical system.

The main use of the set-theoretic semantics for PL^0 is that we can use it to give meaning to concept axioms, which we use to describe the "world".

Concept Axioms

▷ **Observation:** Set-theoretic semantics of 'true' and 'false' ($\top := \varphi\sqcup\overline{\varphi}$ $\bot := \varphi\sqcap\overline{\varphi}$)

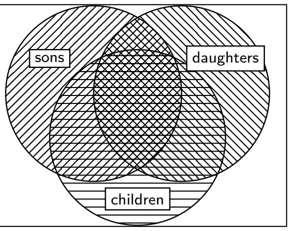
$$[\![\top]\!] = [\![p]\!] \cup [\![\overline{p}]\!] = [\![p]\!] \cup \mathcal{D}\backslash[\![p]\!] = \mathcal{D} \qquad \text{Analogously: } [\![\bot]\!] = \emptyset$$

▷ **Idea:** Use logical axioms to describe the world (Axioms restrict the class of admissible domain structures)

▷ **Definition 16.2.3.** A concept axiom is a $\mathrm{PL}^0_{\mathrm{DL}}$ formula $\mathbf{A}$ that is assumed to be true in the world.

▷ **Definition 16.2.4 (Set-Theoretic Semantics of Axioms).** $\mathbf{A}$ is true in domain of discourse $\mathcal{D}$ iff $[\![\mathbf{A}]\!] = \mathcal{D}$.

▷ **Example 16.2.5.** A world with three concepts and no concept axioms

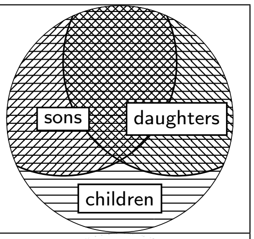
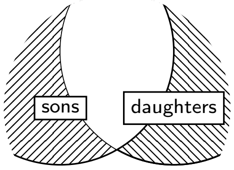
concepts	Set Semantics
child daughter son	

Concept axioms are used to restrict the set of admissible domains to the intended ones. In our situation, we require them to be true – as usual – which here means that they denote the whole domain $\mathcal{D}$.

Let us fortify our intuition about concept axioms with a simple example about the sibling relation. We give four concept axioms and study their effect on the admissible models by looking at the respective Venn diagrams. In the end we see that in all admissible models, the denotations of the concepts son and daughter are disjointq, and child is the union of the two – just as intended.

Effects of Axioms to Siblings

▷ **Example 16.2.6.** We can use concept axioms to describe the world from ??.

Axioms	Semantics
son $\sqsubseteq$ child iff $\overline{[\![\text{son}]\!]} \cup [\![\text{child}]\!] = \mathcal{D}$ iff $[\![\text{son}]\!] \subseteq [\![\text{child}]\!]$ daughter $\sqsubseteq$ child iff $\overline{[\![\text{daughter}]\!]} \cup [\![\text{child}]\!] = \mathcal{D}$ iff $[\![\text{daughter}]\!] \subseteq [\![\text{child}]\!]$	
$\overline{\text{son} \sqcap \text{daughter}}$ child $\sqsubseteq$ son $\sqcup$ daughter	

The set-theoretic semantics introduced above is compatible with the regular semantics of propositional logic, therefore we have the same propositional identities. Their validity can be established directly from the settings in **??**.

Propositional Identities

Name	for $\sqcap$	for $\sqcup$
Idempot.	$\varphi \sqcap \varphi = \varphi$	$\varphi \sqcup \varphi = \varphi$
Identity	$\varphi \sqcap \top = \varphi$	$\varphi \sqcup \bot = \varphi$
Absorpt.	$\varphi \sqcup \top = \top$	$\varphi \sqcap \bot = \bot$
Commut.	$\varphi \sqcap \psi = \psi \sqcap \varphi$	$\varphi \sqcup \psi = \psi \sqcup \varphi$
Assoc.	$\varphi \sqcap (\psi \sqcap \theta) = (\varphi \sqcap \psi) \sqcap \theta$	$\varphi \sqcup (\psi \sqcup \theta) = (\varphi \sqcup \psi) \sqcup \theta$
Distrib.	$\varphi \sqcap (\psi \sqcup \theta) = (\varphi \sqcap \psi) \sqcup (\varphi \sqcap \theta)$	$\varphi \sqcup (\psi \sqcap \theta) = (\varphi \sqcup \psi) \sqcap (\varphi \sqcup \theta)$
Absorpt.	$\varphi \sqcap (\varphi \sqcup \theta) = \varphi$	$\varphi \sqcup \varphi \sqcap \theta = \varphi \sqcap \theta$
Morgan	$\overline{\varphi \sqcap \psi} = \overline{\varphi} \sqcup \overline{\psi}$	$\overline{\varphi \sqcup \psi} = \overline{\varphi} \sqcap \overline{\psi}$
dneg	$\overline{\overline{\varphi}} = \varphi$	

Michael Kohlhase: Artificial Intelligence 1 500 2025-02-06

There is another way we can approach the set description interpretation of propositional logic: by translation into a logic that can express knowledge about sets – first-order logic.

Set-Theoretic Semantics and Predicate Logic

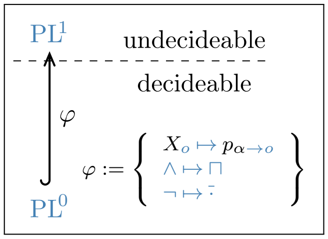
▷ **Definition 16.2.7.** Translation into PL^1 (borrow semantics from that)

- ▷ recursively add argument variable x
- ▷ change back $\sqcap, \sqcup, \sqsubseteq, \equiv$ to $\wedge, \vee, \Rightarrow, \Leftrightarrow$
- ▷ universal closure for x at formula level.

Definition	Comment
$\overline{p}^{fo(x)} := p(x)$	
$\overline{\overline{\mathbf{A}}}^{fo(x)} := \neg\overline{\mathbf{A}}^{fo(x)}$	
$\overline{\mathbf{A} \sqcap \mathbf{B}}^{fo(x)} := \overline{\mathbf{A}}^{fo(x)} \wedge \overline{\mathbf{B}}^{fo(x)}$	$\wedge$ vs. $\sqcap$
$\overline{\mathbf{A} \sqcup \mathbf{B}}^{fo(x)} := \overline{\mathbf{A}}^{fo(x)} \vee \overline{\mathbf{B}}^{fo(x)}$	$\vee$ vs. $\sqcup$
$\overline{\mathbf{A} \sqsubseteq \mathbf{B}}^{fo(x)} := \overline{\mathbf{A}}^{fo(x)} \Rightarrow \overline{\mathbf{B}}^{fo(x)}$	$\Rightarrow$ vs. $\sqsubseteq$
$\overline{\mathbf{A} = \mathbf{B}}^{fo(x)} := \overline{\mathbf{A}}^{fo(x)} \Leftrightarrow \overline{\mathbf{B}}^{fo(x)}$	$\Leftrightarrow$ vs. $=$
$\overline{\mathbf{A}}^{fo} := (\forall x.\overline{\mathbf{A}}^{fo(x)})$	for formulae

Michael Kohlhase: Artificial Intelligence 1 501 2025-02-06

Normally, we embed PL^0 into PL^1 by mapping propositional variables to atomic first-order propositions and the connectives to themselves. The purpose of this embedding is to "talk about truth/falsity of assertions". For "talking about sets" we use a non-standard embedding: propositional variables in PL^0 are mapped to first-order predicates, and the connectives to corresponding set operations. This uses the convention that a set S is represented by a unary predicate p_S (its characteristic predicate), and set membership $a \in S$ as $p_S(a)$.


PL^1
undecideable
decideable
φ
$\varphi := \left\{ \begin{array}{l} X_o \mapsto p_{\alpha\to o} \\ \wedge \mapsto \sqcap \\ \neg \mapsto \overline{\cdot} \end{array} \right\}$
PL^0

Translation Examples

- **Example 16.2.8.** We translate the concept axioms from ?? to fortify our intuition:

$$\begin{aligned}
\overline{\mathsf{son} \sqsubseteq \mathsf{child}}^{fo} &= \forall x.\mathsf{son}(x) \Rightarrow \mathsf{child}(x)\\
\overline{\mathsf{daughter} \sqsubseteq \mathsf{child}}^{fo} &= \forall x.\mathsf{daughter}(x) \Rightarrow \mathsf{child}(x)\\
\overline{\overline{\mathsf{son} \sqcap \mathsf{daughter}}}^{fo} &= \forall x.\overline{\mathsf{son}(x) \wedge \mathsf{daughter}(x)}\\
\overline{\mathsf{child} \sqsubseteq \mathsf{son} \sqcup \mathsf{daughter}}^{fo} &= \forall x.\mathsf{child}(x) \Rightarrow (\mathsf{son}(x) \vee \mathsf{daughter}(x))
\end{aligned}$$

- What are the advantages of translation to PL^1?
  - **theoretically**: A better understanding of the semantics
  - **computationally**: Description Logic Framework, but NOTHING for PL^0
    - we can follow this pattern for richer description logics.
    - many tests are decidable for PL^0, but not for PL^1. (Description Logics?)

Michael Kohlhase: Artificial Intelligence 1 502 2025-02-06

16.2.2 Ontologies and Description Logics

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/27298`. We have seen how sets of concept axioms can be used to describe the "world" by restricting the set of admissible models. We want to call such concept descriptions "ontologies" – formal descriptions of (classes of) objects and their relations.

Ontologies aka. "World Descriptions"

- **Definition 16.2.9 (Classical).** An ontology is a representation of the types, properties, and interrelationships of the entities that really or fundamentally exist for a particular domain of discourse.

- **Remark:** ?? is very general, and depends on what we mean by "representation", "entities", "types", and "interrelationships".

  This may be a feature, and not a bug, since we can use the same intuitions across a variety of representations.

- **Definition 16.2.10.** An ontology consists of a formal system $\langle\mathcal{L},\mathcal{C},\mathcal{K},\models\rangle$ with concept axiom (expressed in $\mathcal{L}$) about
  - individuals: concrete entities in a domain of discourse,
  - concepts: particular collections of individuals that share properties and aspects – the instances of the concept, and
  - relations: ways in which individuals can be related to one another.

- **Example 16.2.11.** Semantic networks are ontologies. (relatively informal)

- **Example 16.2.12.** $\mathrm{PL}^0_{\mathrm{DL}}$ is an ontology format. (formal, but relatively weak)

- **Example 16.2.13.** PL^1 is an ontology format as well. (formal, expressive)

As we will see, the situation for $\mathrm{PL}^0_{\mathrm{DL}}$ is typical for formal ontologies (even though it only offers concepts), so we state the general description logic paradigm for ontologies. The important idea is that having a formal system as an ontology format allows us to capture, study, and implement ontological inference.

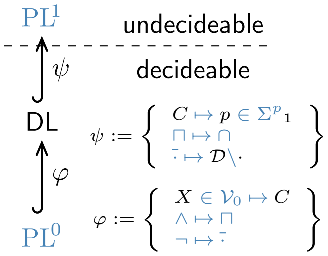
The Description Logic Paradigm

▷ **Idea:** Build a whole family of logics for describing sets and their relations. (tailor their expressivity and computational properties)

▷ **Definition 16.2.14.** A description logic is a formal system for talking about collections of objects and their relations that is at least as expressive as PL^0 with set-theoretic semantics and offers individuals and relations.

A description logic has the following four components:

- a formal language $\mathcal{L}$ with logical constants $\sqcap$, $\overline{\cdot}$, $\sqcup$, $\sqsubseteq$, and $\equiv$,
- a set-theoretic semantics $\langle\mathcal{D}, [\![\cdot]\!]\rangle$,
- a translation into first-order logic that is compatible with $\langle\mathcal{D}, [\![\cdot]\!]\rangle$, and
- a calculus for $\mathcal{L}$ that induces a decision procedure for $\mathcal{L}$-satisfiability.



$$\psi := \left\{\begin{array}{l} C \mapsto p \in {\Sigma^p}_1 \\ \sqcap \mapsto \cap \\ \overline{\cdot} \mapsto \mathcal{D}\backslash\cdot \end{array}\right\}$$

$$\varphi := \left\{\begin{array}{l} X \in \mathcal{V}_0 \mapsto C \\ \wedge \mapsto \sqcap \\ \neg \mapsto \overline{\cdot} \end{array}\right\}$$

▷ **Definition 16.2.15.** Given a description logic $\mathcal{D}$, a $\mathcal{D}$ ontology consists of

- a terminology (or TBox): concepts and roles and a set of concept axioms that describe them, and
- assertions (or ABox): a set of individuals and statements about concept membership and role relationships for them.

For convenience we add concept definitions as a mechanism for defining new concepts from old ones. The so-defined concepts inherit the properties from the concepts they are defined from.

TBoxes in Description Logics

▷ Let $\mathcal{D}$ be a description logic with concepts $\mathcal{C}$.

▷ **Definition 16.2.16.** A concept definition is a pair $c{=}\mathbf{C}$, where c is a new concept name and $\mathbf{C} \in \mathcal{C}$ is a $\mathcal{D}$-formula.

▷ **Example 16.2.17.** We can define mother=woman ⊓ has_child.

▷ **Definition 16.2.18.** A concept definition $c{=}C$ is called recursive, iff c occurs in C.

▷ **Definition 16.2.19.** An TBox is a finite set of concept definitions and concept axioms. It is called acyclic, iff it does not contain recursive definitions.

- ▷ **Definition 16.2.20.** A formula $\mathbf{A}$ is called normalized wrt. an TBox $\mathcal{T}$, iff it does not contain concepts defined in $\mathcal{T}$. (convenient)
- ▷ **Definition 16.2.21 (Algorithm).** (for arbitrary DLs)
  Input: A formula $\mathbf{A}$ and a TBox $\mathcal{T}$.
  - ▹ While [$\mathbf{A}$ contains concept c and $\mathcal{T}$ a concept definition $c{=}\mathbf{C}$]
    - ▹ substitute c by $\mathbf{C}$ in $\mathbf{A}$.
- ▷ **Lemma 16.2.22.** *This algorithm terminates for acyclic TBoxes, but results can be exponentially large.*

FAU Michael Kohlhase: Artificial Intelligence 1 505 2025-02-06

As $\mathrm{PL}^0_{\mathrm{DL}}$ does not offer any guidance on this, we will leave the discussion of ABoxes to **??** when we have introduced our first proper description logic $\mathcal{ALC}$.

16.2.3 Description Logics and Inference

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/27299`.

Now that we have established the description logic paradigm, we will have a look at the inference services that can be offered on this basis.

Before we go into details of particular description logics, we must ask ourselves what kind of inference support we would want for building systems that support knowledge workers in building, maintaining and using ontologies. An example of such a system is the Protégé system [Pro], which can serve for guiding our intuition.

Kinds of Inference in Description Logics

- ▷ **Definition 16.2.23.** Ontology systems employ three main reasoning services:
  - ▹ Consistency test: is a concept definition satisfiable?
  - ▹ Subsumption test: does a concept subsume another?
  - ▹ Instance test: is an individual an example of a concept?
- ▷ **Problem:** decidability, complexity, algorithm

FAU Michael Kohlhase: Artificial Intelligence 1 506 2025-02-06

We will now through these inference-based tests separately.

The consistency test checks for concepts that do not/cannot have instances. We want to avoid such concepts in our ontologies, since they clutter the namespace and do not contribute any meaningful contribution.

Consistency Test

- ▷ **Definition 16.2.24.** We call a concept C consistent, iff there is no concept A, with both $C \sqsubseteq A$ and $C \sqsubseteq \overline{A}$.
- ▷ Or equivalently:
- ▷ **Definition 16.2.25.** A concept C is called inconsistent, iff $[\![C]\!] = \emptyset$ for all $\mathcal{D}$.
- ▷ **Example 16.2.26 (T-Box in $\mathrm{PL}^0_{\mathrm{DL}}$).**

man	=	person ⊓ has_Y	person with y-chromosome
woman	=	person ⊓ $\overline{\text{has\_Y}}$	person without y-chromosome
hermaphrodite	=	man ⊓ woman	man and woman

This specification is inconsistent, i.e. $[\![\text{hermaphrodite}]\!] = \emptyset$ for all $\mathcal{D}$.

- ▷ **Algorithm:** Satisfiability test (usually NP complete)
  we know how to do this, e.g. tableaux, resolution, DPLL in $\mathrm{PL}^0_{\mathrm{DL}}$.

Even though consistency in our example seems trivial, large ontologies can make machine support necessary. This is even more true for ontologies that change over time. Say that an ontology initially has the concept definitions woman=person⊓long_hair and man=person⊓bearded, and then is modernized to a more biologically correct state. In the initial version the concept hermaphrodite is consistent, but becomes inconsistent after the renovation; the authors of the renovation should be made aware of this by the system.
The subsumption test determines whether the sets denoted by two concepts are in a subset relation. The main justification for this is that humans tend to be aware of concept subsumption, and tend to think in taxonomic hierarchies. To cater to this, the subsumption test is useful.

Subsumption Test

- ▷ **Example 16.2.27.** In this case trivial

axiom	entailed subsumption relation
man = person ⊓ has_Y	man ⊑ person
woman = person ⊓ $\overline{\text{has\_Y}}$	woman ⊑ person

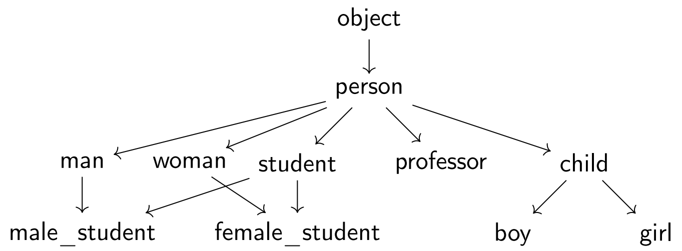
- ▷ **Definition 16.2.28.** $\mathbf{A}$ subsumes $\mathbf{B}$ (modulo a set $\mathcal{A}$ of concept axioms), iff $[\![\mathbf{B}]\!] \subseteq [\![\mathbf{A}]\!]$ for all interpretations $\mathcal{D}$ that satisfy $\mathcal{A}$.
- ▷ **Observation:** Or equivalently, iff $\mathcal{A} \sqsubseteq \mathbf{B} \sqsubseteq \mathbf{A} = \top$
- ▷ **Reduction to consistency test:** (need to implement only one)
  In PL^0, $\mathcal{A} \Rightarrow (\mathbf{A} \Rightarrow \mathbf{B})$ is valid iff $\mathcal{A} \wedge \mathbf{A} \wedge \neg\mathbf{B}$ is inconsistent.
- ▷ **In our example:** The concept person subsumes woman and man.

The good news is that we can reduce the subsumption test to the consistency test, so we can re-use our existing implementation.
The main user-visible service of the subsumption test is to compute the actual taxonomy induced by an ontology.

Classification

- ▷ The subsumption relation among all concepts (subsumption graph)
- ▷ Visualization of the subsumption graph for inspection (plausibility)
- ▷ **Definition 16.2.29.** Classification is the computation of the subsumption graph.

▷ **Example 16.2.30.** (not always so trivial)



Instance Test: Inferring Concept Membership

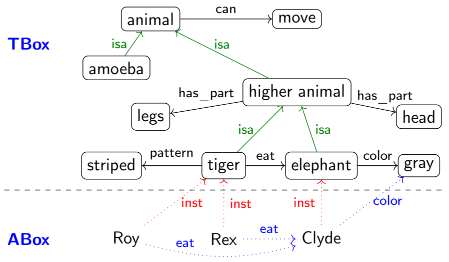
▷ **Definition 16.2.31.** An instance test computes whether given an ontology an individual is a member of a given concept.

▷ **Remark:** This is not something we can do in $\mathrm{PL}^0_{\mathrm{DL}}$, which is a TBox-only system. PL^1 (where concepts are predicate constants an assertions are atoms) suffices.

▷ **Example 16.2.32.** If we define a concept "mother" as "woman who has a child", and have the assertions "Mary is a woman" and "Jesus is a child of Mary", then we can infer that "Mary" is a "Mother", e.g. in the $\mathcal{ND}^1$:

$$\forall x.m(x) \Leftrightarrow w(x) \wedge (\exists y.hc(x,y)), w(M), hc(M,J) \vdash_{\mathcal{ND}^1} m(M)$$

▷ **Remark:** This only works in the presence of concept definitions, not in a purely descriptive framework like semantic networks:



If we take stock of what we have developed so far, then we can see $\mathrm{PL}^0_{\mathrm{DL}}$ as a rational reconstruction of semantic networks restricted to the "isa" relation. We relegate the "instance" relation to **??**.

This reconstruction can now be used as a basis on which we can extend the expressivity and inference procedures without running into problems.

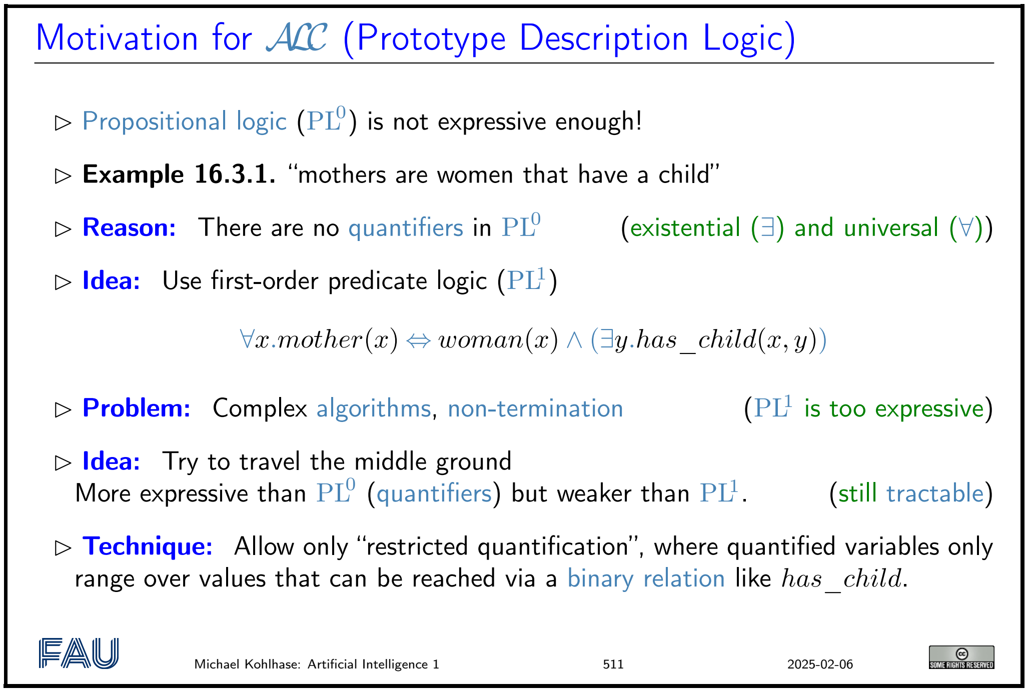
16.3 A simple Description Logic: ALC

In this section, we instantiate the description-logic paradigm further with the prototypical logic $\mathcal{ALC}$, which we will introduce now.

16.3.1 Basic ALC: Concepts, Roles, and Quantification

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/27300`.

In this subsection, we instantiate the description-logic paradigm with the prototypical logic $\mathcal{ALC}$, which we will develop now.



Motivation for $\mathcal{ALC}$ (Prototype Description Logic)

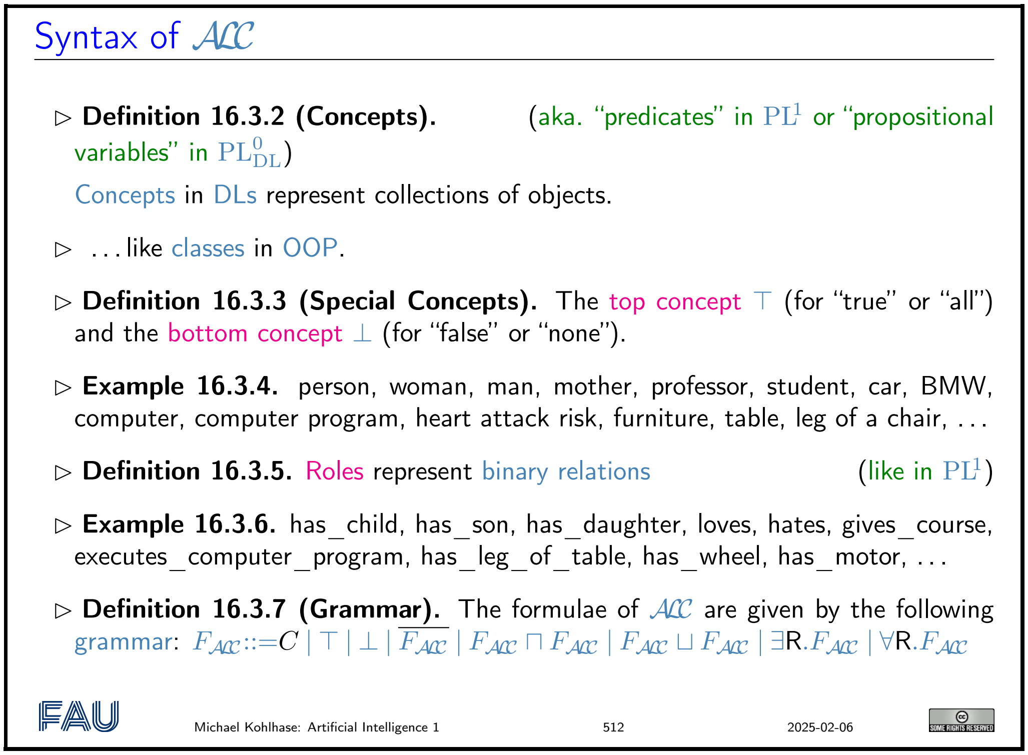
- ▷ Propositional logic (PL^0) is not expressive enough!
- ▷ **Example 16.3.1.** "mothers are women that have a child"
- ▷ **Reason:** There are no quantifiers in PL^0 (existential ($\exists$) and universal ($\forall$))
- ▷ **Idea:** Use first-order predicate logic (PL^1)

$$\forall x.mother(x) \Leftrightarrow woman(x) \wedge (\exists y.has\_child(x,y))$$

- ▷ **Problem:** Complex algorithms, non-termination (PL^1 is too expressive)
- ▷ **Idea:** Try to travel the middle ground
  More expressive than PL^0 (quantifiers) but weaker than PL^1. (still tractable)
- ▷ **Technique:** Allow only "restricted quantification", where quantified variables only range over values that can be reached via a binary relation like has_child.

FAU Michael Kohlhase: Artificial Intelligence 1 511 2025-02-06

$\mathcal{ALC}$ extends the concept operators of $\mathrm{PL}^0_{\mathrm{DL}}$ with binary relations (called "roles" in $\mathcal{ALC}$). This gives $\mathcal{ALC}$ the expressive power we had for the basic semantic networks from ??.



Syntax of $\mathcal{ALC}$

- ▷ **Definition 16.3.2 (Concepts).** (aka. "predicates" in PL^1 or "propositional variables" in $\mathrm{PL}^0_{\mathrm{DL}}$)
  Concepts in DLs represent collections of objects.
- ▷ . . . like classes in OOP.
- ▷ **Definition 16.3.3 (Special Concepts).** The top concept $\top$ (for "true" or "all") and the bottom concept $\bot$ (for "false" or "none").
- ▷ **Example 16.3.4.** person, woman, man, mother, professor, student, car, BMW, computer, computer program, heart attack risk, furniture, table, leg of a chair, . . .
- ▷ **Definition 16.3.5.** Roles represent binary relations (like in PL^1)
- ▷ **Example 16.3.6.** has_child, has_son, has_daughter, loves, hates, gives_course, executes_computer_program, has_leg_of_table, has_wheel, has_motor, . . .
- ▷ **Definition 16.3.7 (Grammar).** The formulae of $\mathcal{ALC}$ are given by the following grammar: $F_{\mathcal{ALC}} ::= C \mid \top \mid \bot \mid \overline{F_{\mathcal{ALC}}} \mid F_{\mathcal{ALC}} \sqcap F_{\mathcal{ALC}} \mid F_{\mathcal{ALC}} \sqcup F_{\mathcal{ALC}} \mid \exists \mathsf{R}.F_{\mathcal{ALC}} \mid \forall \mathsf{R}.F_{\mathcal{ALC}}$

FAU Michael Kohlhase: Artificial Intelligence 1 512 2025-02-06

$\mathcal{ALC}$ restricts the quantification to range all individuals reachable as role successor. The distinction between universal and existential quantifiers clarifies an implicit ambiguity in semantic networks.

Syntax of $\mathcal{ALC}$: Examples

- ▷ **Example 16.3.8.** $\mathrm{person}\sqcap\exists\mathrm{has\_child}.\mathrm{student}$

  $\widehat{=}$ The set of persons that have a child which is a student

  $\widehat{=}$ parents of students

- ▷ **Example 16.3.9.** $\mathrm{person}\sqcap\exists\mathrm{has\_child}.\exists\mathrm{has\_child}.\mathrm{student}$

  $\widehat{=}$ grandparents of students

- ▷ **Example 16.3.10.** $\mathrm{person}\sqcap\exists\mathrm{has\_child}.\exists\mathrm{has\_child}.(\mathrm{student}\sqcup\mathrm{teacher})$

  $\widehat{=}$ grandparents of students or teachers

- ▷ **Example 16.3.11.** $\mathrm{person}\sqcap\forall\mathrm{has\_child}.\mathrm{student}$

  $\widehat{=}$ parents whose children are all students

- ▷ **Example 16.3.12.** $\mathrm{person}\sqcap\forall haschild.\exists\mathrm{has\_child}.\mathrm{student}$

  $\widehat{=}$ grandparents, whose children all have at least one child that is a student

More $\mathcal{ALC}$ Examples

- ▷ **Example 16.3.13.** $\mathrm{car}\sqcap\exists\mathrm{has\_part}.\exists\mathrm{made\_in}.\overline{\mathrm{EU}}$

  $\widehat{=}$ cars that have at least one part that has not been made in the EU

- ▷ **Example 16.3.14.** $\mathrm{student}\sqcap\forall\mathrm{audits\_course}.\mathrm{graduatelevelcourse}$

  $\widehat{=}$ students, that only audit graduate level courses

- ▷ **Example 16.3.15.** $\mathrm{house}\sqcap\forall\mathrm{has\_parking}.\mathrm{off\_street}$ $\widehat{=}$ houses with off-street parking

- ▷ **Note:** $p\sqsubseteq q$ can still be used as an abbreviation for $\overline{p}\sqcup q$.

- ▷ **Example 16.3.16.** $\mathrm{student}\sqcap\forall\mathrm{audits\_course}.(\exists\mathrm{hastutorial}.\top\sqsubseteq\forall\mathrm{has\_TA}.\mathrm{woman})$

  $\widehat{=}$ students that only audit courses that either have no tutorial or tutorials that are TAed by women

As before we allow concept definitions so that we can express new concepts from old ones, and obtain more concise descriptions.

$\mathcal{ALC}$ Concept Definitions

- ▷ **Idea:** Define new concepts from known ones.

- ▷ **Definition 16.3.17.** A concept definition is a pair consisting of a new concept name (the definiendum) and an $\mathcal{ALC}$ formula (the definiens). Concepts that are not definienda are called primitive.

▷ We extend the $\mathcal{ALC}$ grammar from ?? by the production

$$CD_{\mathcal{ALC}}::=C=F_{\mathcal{ALC}}$$

▷ **Example 16.3.18.**

Definition	rec?
man = person ⊓ ∃has_chrom.Y_chrom	-
woman = person ⊓ ∀has_chrom.$\overline{\text{Y\_chrom}}$	-
mother = woman ⊓ ∃has_child.person	-
father = man ⊓ ∃has_child.person	-
grandparent = person ⊓ ∃has_child.(mother ⊔ father)	-
german = person ⊓ ∃has_parents.german	+
number_list = empty_list ⊔ ∃is_first.number ⊓ ∃is_rest.number_list	+

As before, we can normalize a TBox by definition expansion if it is acyclic. With the introduction of roles and quantification, concept definitions in $\mathcal{ALC}$ have a more “interesting” way to be cyclic as ?? shows.

TBox Normalization in $\mathcal{ALC}$

▷ **Definition 16.3.19.** We call an $\mathcal{ALC}$ formula φ normalized wrt. a set of concept definitions, iff all concepts occurring in φ are primitive.

▷ **Definition 16.3.20.** Given a set $\mathcal{D}$ of concept definitions, normalization is the process of replacing in an $\mathcal{ALC}$ formula φ all occurrences of definienda in $\mathcal{D}$ with their definientia.

▷ **Example 16.3.21 (Normalizing grandparent).**

grandparent
↦ person ⊓ ∃has_child.(mother ⊔ father)
↦ person ⊓ ∃has_child.(woman ⊓ ∃has_child.person ⊓ man ⊓ ∃has_child.person)
↦ person ⊓ ∃has_child.(person ⊓ ∃has_chrom.Y_chrom ⊓ ∃has_child.person ⊓ person ⊓ ∃has_chrom.Y_chrom ⊓ ∃has_child.person)

▷ **Observation 16.3.22.** *Normalization results can be exponential.* *(contain redundancies)*

▷ **Observation 16.3.23.** *Normalization need not terminate on cyclic TBoxes.*

▷ **Example 16.3.24.**

german ↦ person ⊓ ∃has_parents.german
↦ person ⊓ ∃has_parents.(person ⊓ ∃has_parents.german)
↦ ...

Now that we have motivated and fixed the syntax of $\mathcal{ALC}$, we will give it a formal semantics. The semantics of $\mathcal{ALC}$ is an extension of the set-theoretic semantics for PL^0, thus the interpretation $[\![\cdot]\!]$ assigns subsets of the domain of discourse to concepts and binary relations over the domain of discourse to roles.

Semantics of $\mathcal{ALC}$

- $\mathcal{ALC}$ semantics is an extension of the set-semantics of propositional logic.
- **Definition 16.3.25.** A model for $\mathcal{ALC}$ is a pair $\langle\mathcal{D},[\![\cdot]\!]\rangle$, where $\mathcal{D}$ is a non-empty set called the domain of discourse and $[\![\cdot]\!]$ a mapping called the interpretation, such that

Op.	formula semantics
	$[\![c]\!] \subseteq \mathcal{D} = [\![\top]\!]$ $\quad [\![\bot]\!] = \emptyset$ $\quad [\![r]\!] \subseteq \mathcal{D}\times\mathcal{D}$
$\bar{\cdot}$	$[\![\overline{\varphi}]\!] = \overline{[\![\varphi]\!]} = \mathcal{D}\backslash[\![\varphi]\!]$
$\sqcap$	$[\![\varphi\sqcap\psi]\!] = [\![\varphi]\!]\cap[\![\psi]\!]$
$\sqcup$	$[\![\varphi\sqcup\psi]\!] = [\![\varphi]\!]\cup[\![\psi]\!]$
$\exists$R.	$[\![\exists \mathsf{R}.\varphi]\!] = \{x\in\mathcal{D}\,\vert\,\exists y.\langle x,y\rangle\in[\![\mathsf{R}]\!]$ and $y\in[\![\varphi]\!]\}$
$\forall$R.	$[\![\forall \mathsf{R}.\varphi]\!] = \{x\in\mathcal{D}\,\vert\,\forall y.$if $\langle x,y\rangle\in[\![\mathsf{R}]\!]$ then $y\in[\![\varphi]\!]\}$

- Alternatively we can define the semantics of $\mathcal{ALC}$ by translation into PL^1.
- **Definition 16.3.26.** The translation of $\mathcal{ALC}$ into PL^1 extends the one from ?? by the following quantifier rules:

$$\overline{\forall \mathsf{R}.\varphi}^{fo(x)} := (\forall y.\mathsf{R}(x,y)\Rightarrow\overline{\varphi}^{fo(y)}) \qquad \overline{\exists \mathsf{R}.\varphi}^{fo(x)} := (\exists y.\mathsf{R}(x,y)\wedge\overline{\varphi}^{fo(y)})$$

- **Observation 16.3.27.** *The set-theoretic semantics from ?? and the "semantics-by-translation" from ?? induce the same notion of satisfiability.*

We can now use the $\mathcal{ALC}$ identities above to establish a useful normal form for $\mathcal{ALC}$. This will play a role in the inference procedures we study next.
The following identitieswill be useful later on. They can be proven directly with the settings from ??; we carry this out for one of them below.

$\mathcal{ALC}$ Identities

1	$\overline{\exists \mathsf{R}.\varphi} = \forall \mathsf{R}.\overline{\varphi}$	3	$\overline{\forall \mathsf{R}.\varphi} = \exists \mathsf{R}.\overline{\varphi}$
2	$\forall \mathsf{R}.(\varphi\sqcap\psi) = \forall \mathsf{R}.\varphi\sqcap\forall \mathsf{R}.\psi$	4	$\exists \mathsf{R}.(\varphi\sqcup\psi) = \exists \mathsf{R}.\varphi\sqcup\exists \mathsf{R}.\psi$

- Proof of 1

$$\begin{aligned}
[\![\overline{\exists \mathsf{R}.\varphi}]\!] = \mathcal{D}\backslash[\![\exists \mathsf{R}.\varphi]\!] &= \mathcal{D}\backslash\{x\in\mathcal{D}\,\vert\,\exists y.(\langle x,y\rangle\in[\![\mathsf{R}]\!]) \text{ and } (y\in[\![\varphi]\!])\}\\
&= \{x\in\mathcal{D}\,\vert\,\text{not } \exists y.(\langle x,y\rangle\in[\![\mathsf{R}]\!]) \text{ and } (y\in[\![\varphi]\!])\}\\
&= \{x\in\mathcal{D}\,\vert\,\forall y.\text{if } (\langle x,y\rangle\in[\![\mathsf{R}]\!]) \text{ then } (y\notin[\![\varphi]\!])\}\\
&= \{x\in\mathcal{D}\,\vert\,\forall y.\text{if } (\langle x,y\rangle\in[\![\mathsf{R}]\!]) \text{ then } (y\in(\mathcal{D}\backslash[\![\varphi]\!]))\}\\
&= \{x\in\mathcal{D}\,\vert\,\forall y.\text{if } (\langle x,y\rangle\in[\![\mathsf{R}]\!]) \text{ then } (y\in[\![\overline{\varphi}]\!])\}\\
&= [\![\forall \mathsf{R}.\overline{\varphi}]\!]
\end{aligned}$$

The form of the identities (interchanging quantification with connectives) is reminiscient of identities in PL^1; this is no coincidence as the “semantics by translation” of **??** shows.

Negation Normal Form

- **Definition 16.3.28 (NNF).** An $\mathcal{ALC}$ formula is in negation normal form (NNF), iff complement ($\overline{\cdot}$) is only applied to primitive concept.
- Use the $\mathcal{ALC}$ identities as rules to compute it. (in linear time)
- **Example 16.3.29.**

example	by rule
$\overline{\exists \mathsf{R}.(\forall \mathsf{S}.e \sqcap \overline{\forall \mathsf{S}.d})}$	
$\mapsto \forall \mathsf{R}.\overline{\forall \mathsf{S}.e \sqcap \overline{\forall \mathsf{S}.d}}$	$\overline{\exists \mathsf{R}.\varphi} \mapsto \forall \mathsf{R}.\overline{\varphi}$
$\mapsto \forall \mathsf{R}.(\overline{\forall \mathsf{S}.e} \sqcup \overline{\overline{\forall \mathsf{S}.d}})$	$\overline{\varphi \sqcap \psi} \mapsto \overline{\varphi} \sqcup \overline{\psi}$
$\mapsto \forall \mathsf{R}.(\exists \mathsf{S}.\overline{e} \sqcup \overline{\overline{\forall \mathsf{S}.d}})$	$\overline{\forall \mathsf{R}.\varphi} \mapsto \exists \mathsf{R}.\overline{\varphi}$
$\mapsto \forall \mathsf{R}.(\exists \mathsf{S}.\overline{e} \sqcup \forall \mathsf{S}.d)$	$\overline{\overline{\varphi}} \mapsto \varphi$

Finally, we extend $\mathcal{ALC}$ with an ABox component. This mainly means that we define two new assertions in $\mathcal{ALC}$ and specify their semantics and PL^1 translation.

$\mathcal{ALC}$ with Assertions about Individuals

- **Definition 16.3.30.** We define the ABox assertions for $\mathcal{ALC}$:
  - Role assertions$a{:}\varphi$ (a is a φ)
  - $a\ \mathsf{R}\ b$ (a stands in relation R to b)

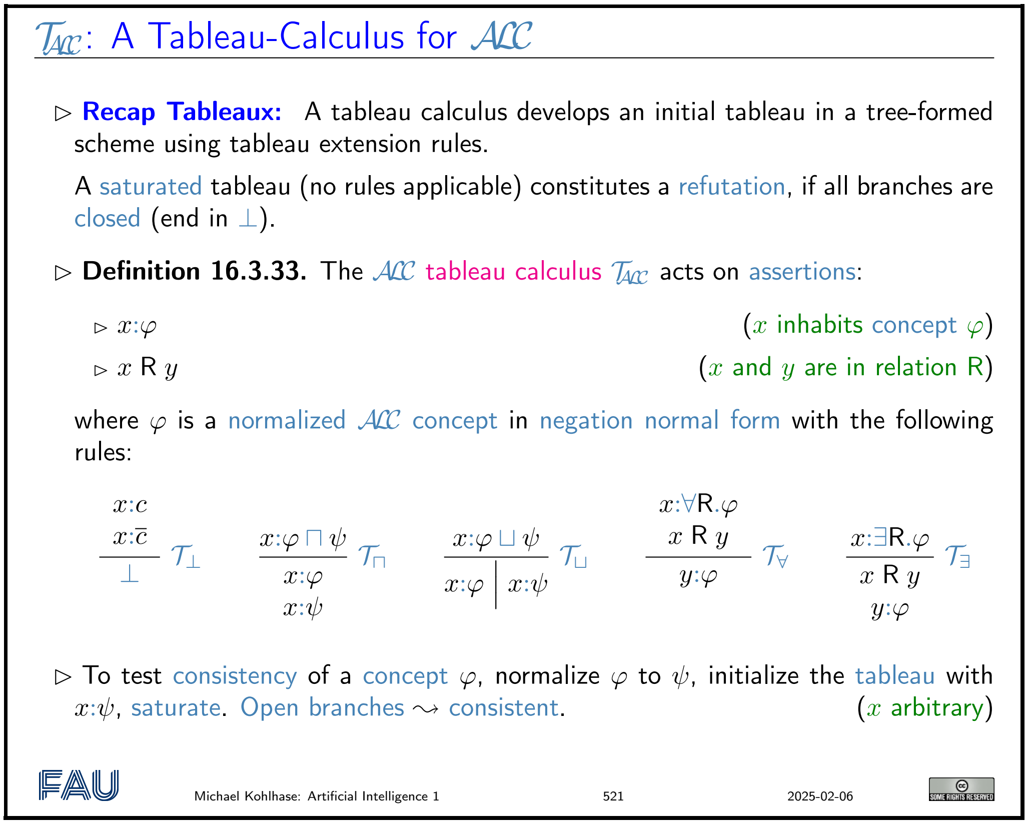
  assertions make up the ABox in $\mathcal{ALC}$.
- **Definition 16.3.31.** Let $\langle \mathcal{D}, [\![\cdot]\!] \rangle$ be a model for $\mathcal{ALC}$, then we define
  - $[\![a{:}\varphi]\!] = \mathsf{T}$, iff $[\![a]\!] \in [\![\varphi]\!]$, and
  - $[\![a\ \mathsf{R}\ b]\!] = \mathsf{T}$, iff $(\,[\![a]\!]\,, [\![b]\!]\,) \in [\![\mathsf{R}]\!]$.
- **Definition 16.3.32.** We extend the PL^1 translation of $\mathcal{ALC}$ to $\mathcal{ALC}$ assertions:
  - $\overline{a{:}\varphi}^{fo} := \overline{\varphi}^{fo(a)}$, and
  - $\overline{a\ \mathsf{R}\ b}^{fo} := \mathsf{R}(a, b)$.

If we take stock of what we have developed so far, then we see that $\mathcal{ALC}$ as a rational reconstruction of semantic networks restricted to the “isa” and “instance” relations – which are the only ones that can really be given a denotational and operational semantics.

16.3.2 Inference for ALC

Video Nuggets covering this subsection can be found at `https://fau.tv/clip/id/27301` and `https://fau.tv/clip/id/27302`.

In this subsection we make good on the motivation from ?? that description logics enjoy tractable inference procedures: We present a tableau calculus for $\mathcal{ALC}$, show that it is a decision procedure, and study its complexity.



$\mathcal{T}_{\mathcal{ALC}}$: A Tableau-Calculus for $\mathcal{ALC}$

- ▷ **Recap Tableaux:** A tableau calculus develops an initial tableau in a tree-formed scheme using tableau extension rules.

  A saturated tableau (no rules applicable) constitutes a refutation, if all branches are closed (end in $\bot$).

- ▷ **Definition 16.3.33.** The $\mathcal{ALC}$ tableau calculus $\mathcal{T}_{\mathcal{ALC}}$ acts on assertions:
  - ▷ $x{:}\varphi$ (x inhabits concept φ)
  - ▷ $x\ \mathsf{R}\ y$ (x and y are in relation R)

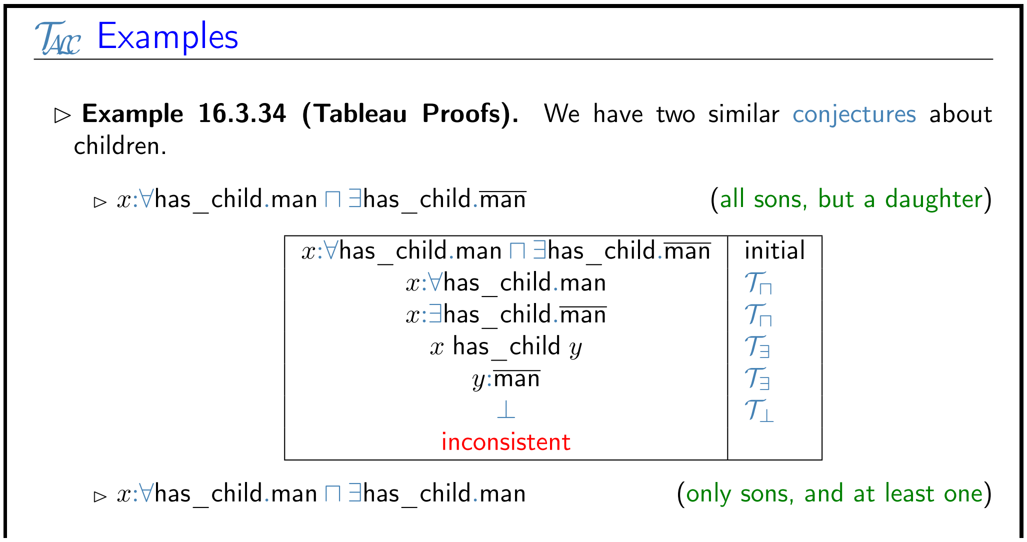
  where φ is a normalized $\mathcal{ALC}$ concept in negation normal form with the following rules:

$$\begin{array}{c} x{:}c \\ x{:}\overline{c} \\ \hline \bot \end{array}\ \mathcal{T}_\bot \qquad \begin{array}{c} x{:}\varphi \sqcap \psi \\ \hline x{:}\varphi \\ x{:}\psi \end{array}\ \mathcal{T}_\sqcap \qquad \begin{array}{c} x{:}\varphi \sqcup \psi \\ \hline x{:}\varphi \;\Big|\; x{:}\psi \end{array}\ \mathcal{T}_\sqcup \qquad \begin{array}{c} x{:}\forall \mathsf{R}.\varphi \\ x\ \mathsf{R}\ y \\ \hline y{:}\varphi \end{array}\ \mathcal{T}_\forall \qquad \begin{array}{c} x{:}\exists \mathsf{R}.\varphi \\ \hline x\ \mathsf{R}\ y \\ y{:}\varphi \end{array}\ \mathcal{T}_\exists$$

- ▷ To test consistency of a concept φ, normalize φ to ψ, initialize the tableau with $x{:}\psi$, saturate. Open branches $\rightsquigarrow$ consistent. (x arbitrary)

In contrast to the tableau tableau calculi for theorem proving we have studied earlier, $\mathcal{T}_{\mathcal{ALC}}$ is run in “model generation mode”. Instead of initializing the tableau with the axioms and the negated conjecture and hope that all branches will close, we initialize the $\mathcal{T}_{\mathcal{ALC}}$ tableau with axioms and the “membership-conjecture” that a given concept φ is satisfiable – i.e. φ h as a member x, and hope for branches that are open, i.e. that make the conjecture true (and at the same time give a model).

Let us now work through two very simple examples; one unsatisfiable, and a satisfiable one.



$\mathcal{T}_{\mathcal{ALC}}$ Examples

- ▷ **Example 16.3.34 (Tableau Proofs).** We have two similar conjectures about children.
  - ▷ $x{:}\forall \mathsf{has\_child}.\mathsf{man} \sqcap \exists \mathsf{has\_child}.\overline{\mathsf{man}}$ (all sons, but a daughter)

$x{:}\forall \mathsf{has\_child}.\mathsf{man} \sqcap \exists \mathsf{has\_child}.\overline{\mathsf{man}}$	initial
$x{:}\forall \mathsf{has\_child}.\mathsf{man}$	$\mathcal{T}_\sqcap$
$x{:}\exists \mathsf{has\_child}.\overline{\mathsf{man}}$	$\mathcal{T}_\sqcap$
$x\ \mathsf{has\_child}\ y$	$\mathcal{T}_\exists$
$y{:}\overline{\mathsf{man}}$	$\mathcal{T}_\exists$
$\bot$	$\mathcal{T}_\bot$
inconsistent	

  - ▷ $x{:}\forall \mathsf{has\_child}.\mathsf{man} \sqcap \exists \mathsf{has\_child}.\mathsf{man}$ (only sons, and at least one)

x:$\forall$has_child.man $\sqcap$ $\exists$has_child.man	initial
x:$\forall$has_child.man	$\mathcal{T}_{\sqcap}$
x:$\exists$has_child.man	$\mathcal{T}_{\sqcap}$
x has_child y	$\mathcal{T}_{\exists}$
y:man	$\mathcal{T}_{\exists}$
open	

This tableau shows a model: there are two persons, x and y. y is the only child of x, y is a man.

Another example: this one is more complex, but the concept is satisfiable.

Another $\mathcal{T}_{\mathcal{ALC}}$ Example

▷ **Example 16.3.35.** $\forall$has_child.(ugrad $\sqcup$ grad) $\sqcap$ $\exists$has_child.$\overline{\text{ugrad}}$ is satisfiable.

- ▷ Let's try it on the board
- ▷ Tableau proof for the notes

1	x:$\forall$has_child.(ugrad $\sqcup$ grad) $\sqcap$ $\exists$has_child.$\overline{\text{ugrad}}$		initial
2	x:$\forall$has_child.(ugrad $\sqcup$ grad)		$\mathcal{T}_{\sqcap}$
3	x:$\exists$has_child.$\overline{\text{ugrad}}$		$\mathcal{T}_{\sqcap}$
4	x has_child y		$\mathcal{T}_{\exists}$
5	y:$\overline{\text{ugrad}}$		$\mathcal{T}_{\exists}$
6	y:ugrad $\sqcup$ grad		$\mathcal{T}_{\forall}$
7	y:ugrad	y:grad	$\mathcal{T}_{\sqcup}$
8	$\perp$	open	

The left branch is closed, the right one represents a model: y is a child of x, y is a graduate student, x hat exactly one child: y.

After we got an intuition about $\mathcal{T}_{\mathcal{ALC}}$, we can now study the properties of the calculus to determine that it is a decision procedure for $\mathcal{ALC}$.

Properties of Tableau Calculi

▷ We study the following properties of a tableau calculus $\mathcal{C}$:

- ▷ Termination: there are no infinite sequences of inference rule applications.
- ▷ Soundness: If φ is satisfiable, then $\mathcal{C}$ terminates with an open branch.
- ▷ Completeness: If φ is in unsatisfiable, then $\mathcal{C}$ terminates and all branches are closed.
- ▷ complexity of the algorithm (time and space complexity).

▷ Additionally, we are interested in the complexity of satisfiability itself (as a benchmark)

The soundness result for $\mathcal{T}_{\mathcal{ALC}}$ is as usual: we start with a model of $x{:}\varphi$ and show that an $\mathcal{T}_{\mathcal{ALC}}$ tableau must have an open branch.

Correctness

- **Lemma 16.3.36.** *If φ satisfiable, then $\mathcal{T}_{\mathcal{ALC}}$ terminates on $x{:}\varphi$ with open branch.*
- *Proof:* Let $\mathcal{M} := \langle \mathcal{D}, [\![\cdot]\!] \rangle$ be a model for φ and $w \in [\![\varphi]\!]$.
  1. We define $[\![x]\!] := w$ and $\begin{array}{lll} \mathcal{M} \models (x{:}\psi) & \text{iff} & [\![x]\!] \in [\![\psi]\!] \\ \mathcal{M} \models x \mathrel{\mathsf{R}} y & \text{iff} & \langle x, y \rangle \in [\![\mathsf{R}]\!] \\ \mathcal{M} \models S & \text{iff} & \mathcal{I} \models c \text{ for all } c \in S \end{array}$
  2. This gives us $\mathcal{M} \models (x{:}\varphi)$ (base case)
  3. If the branch is satisfiable, then either
     - no rule applicable to leaf, (open branch)
     - or rule applicable and one new branch satisfiable. (inductive case: next)
  4. There must be an open branch. (by termination)

We complete the proof by looking at all the $\mathcal{T}_{\mathcal{ALC}}$ inference rules in turn.

Case analysis on the rules

$\mathcal{T}_{\sqcap}$ **applies** then $\mathcal{M} \models (x{:}\varphi \sqcap \psi)$, i.e. $[\![x]\!] \in [\![\varphi \sqcap \psi]\!]$
so $[\![x]\!] \in [\![\varphi]\!]$ and $[\![x]\!] \in [\![\psi]\!]$, thus $\mathcal{M} \models (x{:}\varphi)$ and $\mathcal{M} \models (x{:}\psi)$.

$\mathcal{T}_{\sqcup}$ **applies** then $\mathcal{M} \models (x{:}\varphi \sqcup \psi)$, i.e $[\![x]\!] \in [\![\varphi \sqcup \psi]\!]$
so $[\![x]\!] \in [\![\varphi]\!]$ or $[\![x]\!] \in [\![\psi]\!]$, thus $\mathcal{M} \models (x{:}\varphi)$ or $\mathcal{M} \models (x{:}\psi)$,
wlog. $\mathcal{M} \models (x{:}\varphi)$.

$\mathcal{T}_{\forall}$ **applies** then $\mathcal{M} \models (x{:}\forall \mathsf{R}.\varphi)$ and $\mathcal{M} \models x \mathrel{\mathsf{R}} y$, i.e. $[\![x]\!] \in [\![\forall \mathsf{R}.\varphi]\!]$ and $\langle x, y \rangle \in [\![\mathsf{R}]\!]$, so $[\![y]\!] \in [\![\varphi]\!]$

$\mathcal{T}_{\exists}$ **applies** then $\mathcal{M} \models (x{:}\exists \mathsf{R}.\varphi)$, i.e $[\![x]\!] \in [\![\exists \mathsf{R}.\varphi]\!]$,
so there is a $v \in D$ with $\langle [\![x]\!], v \rangle \in [\![\mathsf{R}]\!]$ and $v \in [\![\varphi]\!]$.
We define $[\![y]\!] := v$, then $\mathcal{M} \models x \mathrel{\mathsf{R}} y$ and $\mathcal{M} \models (y{:}\varphi)$

For the completeness result for $\mathcal{T}_{\mathcal{ALC}}$ we have to start with an open tableau branch and construct a model that satisfies all judgments in the branch. We proceed by building a Herbrand model, whose domain consists of all the individuals mentioned in the branch and which interprets all concepts and roles as specified in the branch. Not surprisingly, the model thus constructed satisfies (all judgments on) the branch.

Completeness of the Tableau Calculus

- **Lemma 16.3.37.** *Open saturated tableau branches for φ induce models for φ.*
- *Proof:* construct a model for the branch and verify for φ
  1. Let $\mathcal{B}$ be an open, saturated branch

- ▷ we define

$$\begin{array}{rcl} \mathcal{D} & : = & \{x \mid x{:}\psi \in \mathcal{B} \text{ or } z \; \mathsf{R} \; x \in \mathcal{B}\} \\ [\![c]\!] & : = & \{x \mid x{:}c \in \mathcal{B}\} \\ [\![\mathsf{R}]\!] & : = & \{\langle x,y\rangle \mid x \; \mathsf{R} \; y \in \mathcal{B}\} \end{array}$$

- ▷ well-defined since never $x{:}c, x{:}\overline{c} \in \mathcal{B}$ (otherwise $\mathcal{T}_\perp$ applies)
- ▷ $\mathcal{M}$ satisfies all assertions $x{:}c$, $x{:}\overline{c}$ and $x \; \mathsf{R} \; y$, (by construction)

2. $\mathcal{M}\models(y{:}\psi)$, for all $y{:}\psi \in \mathcal{B}$ (on $k = size(\psi)$ next slide)
3. $\mathcal{M}\models(x{:}\varphi)$.

We complete the proof by looking at all the $\mathcal{T}_{\mathcal{ALC}}$ inference rules in turn.

Case Analysis for Induction

case $y{:}\psi = y{:}\psi_1 \sqcap \psi_2$ Then $\{y{:}\psi_1, y{:}\psi_2\} \subseteq \mathcal{B}$ ($\mathcal{T}_\sqcap$-rule, saturation)
so $\mathcal{M}\models(y{:}\psi_1)$ and $\mathcal{M}\models(y{:}\psi_2)$ and $\mathcal{M}\models(y{:}\psi_1 \sqcap \psi_2)$ (IH, Definition)

case $y{:}\psi = y{:}\psi_1 \sqcup \psi_2$ Then $y{:}\psi_1 \in \mathbf{B}$ or $y{:}\psi_2 \in \mathbf{B}$ ($\mathcal{T}_\sqcup$, saturation)
so $\mathcal{M}\models(y{:}\psi_1)$ or $\mathcal{M}\models(y{:}\psi_2)$ and $\mathcal{M}\models(y{:}\psi_1 \sqcup \psi_2)$ (IH, Definition)

case $y{:}\psi = y{:}\exists \mathbf{R}.\theta$ then $\{y \; \mathsf{R} \; z, z{:}\theta\} \subseteq \mathbf{B}$ (z new variable) ($\mathcal{T}_\exists$-rules, saturation)
so $\mathcal{M}\models(z{:}\theta)$ and $\mathcal{M}\models y \; \mathsf{R} \; z$, thus $\mathcal{M}\models(y{:}\exists \mathsf{R}.\theta)$. (IH, Definition)

case $y{:}\psi = y{:}\forall \mathbf{R}.\theta$ Let $\langle [\![y]\!], v\rangle \in [\![\mathsf{R}]\!]$ for some $r \in \mathcal{D}$
then $v = z$ for some variable z with $y \; \mathsf{R} \; z \in \mathbf{B}$ (construction of $[\![\mathsf{R}]\!]$)
So $z{:}\theta \in \mathcal{B}$ and $\mathcal{M}\models(z{:}\theta)$. ($\mathcal{T}_\forall$-rule, saturation, Def)
As v was arbitrary we have $\mathcal{M}\models(y{:}\forall \mathsf{R}.\theta)$.

Termination

- ▷ **Theorem 16.3.38.** *$\mathcal{T}_{\mathcal{ALC}}$ terminates.*
- ▷ To prove termination of a tableau algorithm, find a well-founded measure (function) that is decreased by all rules

$$\frac{\begin{array}{c} x{:}c \\ x{:}\overline{c} \end{array}}{\perp}\mathcal{T}_\perp \qquad \frac{x{:}\varphi \sqcap \psi}{\begin{array}{c} x{:}\varphi \\ x{:}\psi \end{array}}\mathcal{T}_\sqcap \qquad \frac{x{:}\varphi \sqcup \psi}{x{:}\varphi \mid x{:}\psi}\mathcal{T}_\sqcup \qquad \frac{\begin{array}{c} x{:}\forall \mathsf{R}.\varphi \\ x \; \mathsf{R} \; y \end{array}}{y{:}\varphi}\mathcal{T}_\forall \qquad \frac{x{:}\exists \mathsf{R}.\varphi}{\begin{array}{c} x \; \mathsf{R} \; y \\ y{:}\varphi \end{array}}\mathcal{T}_\exists$$

- ▷ *Proof:* Sketch (full proof very technical)
  1. Any rule except $\mathcal{T}_\forall$ can only be applied once to $x{:}\psi$.
  2. Rule $\mathcal{T}_\forall$ applicable to $x{:}\forall \mathsf{R}.\psi$ at most as the number of R-successors of x. (those y with $x \; \mathsf{R} \; y$ above)

3. The R-successors are generated by $x{:}\exists R.\theta$ above, (number bounded by size of input formula)
4. Every rule application to $x{:}\psi$ generates constraints $z{:}\psi'$, where ψ' a proper sub-formula of ψ.

We can turn the termination result into a worst-case complexity result by examining the sizes of branches.

Complexity of $\mathcal{T}_{\mathcal{ALC}}$

- **Idea:** Work off tableau branches one after the other. (Branch size $\widehat{=}$ space complexity)
- **Observation 16.3.39.** *The size of the branches is polynomial in the size of the input formula:*

$$\text{branchsize} = \#(\textit{input formulae}) + \#(\exists\textit{-formulae}) \cdot \#(\forall\textit{-formulae})$$

- *Proof sketch:* Re-examine the termination proof and count: the first summand comes from ??, the second one from ?? and ??
- **Theorem 16.3.40.** *The satisfiability problem for $\mathcal{ALC}$ is in* **PSPACE**.
- **Theorem 16.3.41.** *The satisfiability problem for $\mathcal{ALC}$ is* **PSPACE***-Complete.*
- *Proof sketch:* Reduce a **PSPACE**-complete problem to $\mathcal{ALC}$-satisfiability
- **Theorem 16.3.42 (Time Complexity).** *The $\mathcal{ALC}$ satisfiability problem is in* **EXPTIME**.
- *Proof sketch:* There can be exponentially many branches (already for PL^0)

In summary, the theoretical complexity of $\mathcal{ALC}$ is the same as that for PL^0, but in practice $\mathcal{ALC}$ is much more expressive. So this is a clear win.
But the description of the tableau algorithm $\mathcal{T}_{\mathcal{ALC}}$ is still quite abstract, so we look at an exemplary implementation in a functional programming language.

The functional Algorithm for $\mathcal{ALC}$

- **Observation:** (leads to a better treatment for $\exists$)
  - the $\mathcal{T}_{\exists}$-rule generates the constraints x R y and $y{:}\psi$ from $x{:}\exists R.\psi$
  - this triggers the $\mathcal{T}_{\forall}$-rule for $x{:}\forall R.\theta_i$, which generate $y{:}\theta_1$, . . . , $y{:}\theta_n$
  - for y we have $y{:}\psi$ and $y{:}\theta_1$, . . . , $y{:}\theta_n$. (do all of this in a single step)
  - we are only interested in non-emptiness, not in particular witnesses (leave them out)
- **Definition 16.3.43.** The functional algorithm for $\mathcal{T}_{\mathcal{ALC}}$ is

```
consistent(S) =
  if $\{c,\overline{c}\} \subseteq S$ then false
  elif '$\varphi \sqcap \psi' \in S$ and ('$\varphi' \notin S$ or '$\psi' \notin S$)
    then consistent($S \cup \{\varphi,\psi\}$)
  elif '$\varphi \sqcup \psi' \in S$ and $\{\varphi,\psi\} \notin S$
     then consistent($S \cup \{\varphi\}$) or consistent($S \cup \{\psi\}$)
  elif forall '$\exists \mathsf{R}.\psi' \in S$
   consistent($\{\psi\} \cup \{\theta \in \theta \,|\, '\forall \mathsf{R}.\theta' \in S\}$)
  else true
```

- ▷ Relatively simple to implement. (good implementations optimized)
- ▷ **But:** This is restricted to $\mathcal{ALC}$. (extension to other DL difficult)

Note that we have (so far) only considered an empty TBox: we have initialized the tableau with a normalized concept; so we did not need to include the concept definitions. To cover "real" ontologies, we need to consider the case of concept axioms as well.
We now extend $\mathcal{T}_{\mathcal{ALC}}$ with concept axioms. The key idea here is to realize that the concept axioms apply to all individuals. As the individuals are generated by the $\mathcal{T}_{\exists}$ rule, we can simply extend that rule to apply all the concept axioms to the newly introduced individual.

Extending the Tableau Algorithm by Concept Axioms

- ▷ concept axioms, e.g. child $\sqsubseteq$ son $\sqcup$ daughter cannot be handled in $\mathcal{T}_{\mathcal{ALC}}$ yet.
- ▷ **Idea:** Whenever a new variable y is introduced (by $\mathcal{T}_{\exists}$-rule) add the information that axioms hold for y.
  - ▷ Initialize tableau with $\{x{:}\varphi\} \cup \mathcal{CA}$ ($\mathcal{CA}$: = set of concept axioms)
  - ▷ New rule for $\exists$: $$\frac{x{:}\exists \mathsf{R}.\varphi \quad \mathcal{CA} = \{\alpha_1,\ldots,\alpha_n\}}{\begin{array}{c} y{:}\varphi \\ x\ \mathsf{R}\ y \\ y{:}\alpha_1 \\ \vdots \\ y{:}\alpha_n \end{array}}\,\mathcal{T}_{\mathcal{CA}}^{\exists}$$ (instead of $\mathcal{T}_{\exists}$)
- ▷ **Problem:** $\mathcal{CA} := \{\exists \mathsf{R}.c\}$ and start tableau with $x{:}d$ (non-termination)

The problem of this approach is that it spoils termination, since we cannot control the number of rule applications by (fixed) properties of the input formulae. The example shows this very nicely. We only sketch a path towards a solution.

Non-Termination of $\mathcal{T}_{\mathcal{ALC}}$ with Concept Axioms

- ▷ **Problem:** $\mathcal{CA} := \{\exists \mathsf{R}.c\}$ and start tableau with $x{:}d$. (non-termination)

$x{:}d$	start
$x{:}\exists \mathsf{R}.c$	in $\mathcal{CA}$
$x\ \mathsf{R}\ y_1$	$\mathcal{T}_\exists$
$y_1{:}c$	$\mathcal{T}_\exists$
$y_1{:}\exists \mathsf{R}.c$	$\mathcal{T}^\exists_{\mathcal{CA}}$
$y_1\ \mathsf{R}\ y_2$	$\mathcal{T}_\exists$
$y_2{:}c$	$\mathcal{T}_\exists$
$y_2{:}\exists \mathsf{R}.c$	$\mathcal{T}^\exists_{\mathcal{CA}}$
...	

Solution: Loop-Check:

- ▷ Instead of a new variable y take an old variable z, if we can guarantee that whatever holds for y already holds for z.
- ▷ We can only do this, iff the $\mathcal{T}_\forall$-rule has been exhaustively applied.

▷ **Theorem 16.3.44.** *The consistency problem of $\mathcal{ALC}$ with concept axioms is decidable.*

Proof sketch: $\mathcal{T}_{\mathcal{ALC}}$ with a suitable loop check terminates.

16.3.3 ABoxes, Instance Testing, and ALC

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/27303`.

Now that we have a decision problem for $\mathcal{ALC}$ with concept axioms, we can go the final step to the general case of inference in description logics: we add an ABox with assertional axioms that describe the individuals.

We will now extend the description logic $\mathcal{ALC}$ with assertions that can express concept membership.

▷ Instance Test: Concept Membership

▷ **Definition 16.3.45.** An instance test computes whether given an $\mathcal{ALC}$ ontology an individual is a member of a given concept.

▷ **Example 16.3.46 (An Ontology).**

TBox (terminological Box)			ABox (assertional Box, data base)	
woman	=	person ⊓ has_Y	tony:person	Tony is a person
man	=	person ⊓ $\overline{\text{has\_Y}}$	tony:has_Y	Tony has a y-chrom

This entails: tony:man (Tony is a man).

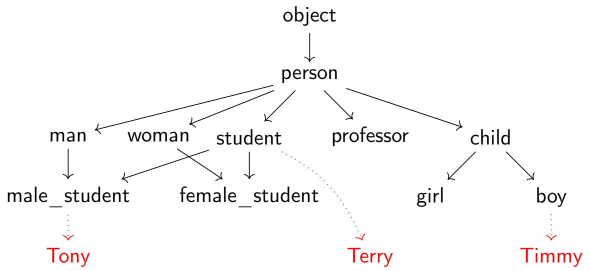
▷ **Problem:** Can we compute this?

If we combine classification with the instance test, then we get the full picture of how concepts and individuals relate to each other. We see that we get the full expressivity of semantic networks in $\mathcal{ALC}$.

Realization

▷ **Definition 16.3.47.** Realization is the computation of all instance relations between ABox objects and TBox concepts.

▷ **Observation:** It is sufficient to remember the lowest concepts in the subsumption graph. (rest by subsumption)



▷ **Example 16.3.48.** If tony:male_student is known, we do not need tony:man.

Michael Kohlhase: Artificial Intelligence 1 535 2025-02-06

Let us now get an intuition on what kinds of interactions between the various parts of an ontology.

ABox Inference in $\mathcal{ALC}$: Phenomena

▷ There are different kinds of interactions between TBox and ABox in $\mathcal{ALC}$ and in description logics in general.

▷ **Example 16.3.49.**

property	example
internally inconsistent	tony:student, tony:$\overline{\text{student}}$
inconsistent with a TBox	TBox: $\overline{\text{student} \sqcap \text{prof}}$ ABox: tony:student, tony:prof
implicit info that is not explicit	ABox: tony:∀has_grad.genius tony has_grad mary ⊨ mary:genius
information that can be combined with TBox info	TBox: happy_prof = prof ⊓ ∀has_grad.genius ABox: tony:happy_prof, tony has_grad mary ⊨ mary:genius

Michael Kohlhase: Artificial Intelligence 1 536 2025-02-06

Again, we ask ourselves whether all of these are computable.
Fortunately, it is very simple to add assertions to $\mathcal{T}_{\mathcal{ALC}}$. In fact, we do not have to change anything, as the judgments used in the tableau are already of the form of ABox assertion.

Tableau-based Instance Test and Realization

▷ **Query:** Do the ABox and TBox together entail $a{:}\varphi$? ($a \in \varphi$?)

▷ **Algorithm:** Test $a{:}\overline{\varphi}$ for consistency with ABox and TBox. (use our tableau algorithm)

▷ **Necessary changes:** (no big deal)

- ▹ Normalize ABox wrt. TBox. (definition expansion)
- ▹ Initialize the tableau with ABox in NNF. (so it can be used)

▷ **Example 16.3.50.**

Example: add mary:$\overline{\text{genius}}$ to determine $ABox, TBox \models$ mary:genius

TBox	happy_prof = prof ⊓ ∀has_grad.genius	tony:prof ⊓ ∀has_grad.genius tony has_grad mary mary:$\overline{\text{genius}}$ tony:prof tony:∀has_grad.genius mary:genius $\bot$	TBox ABox Query $\mathcal{T}_{\sqcap}$ $\mathcal{T}_{\sqcap}$ $\mathcal{T}_{\forall}$ $\mathcal{T}_{\bot}$
ABox	tony:happy_prof tony has_grad mary		

▷ **Note:** The instance test is the base for realization. (remember?)

▷ **Idea:** Extend to more complex ABox queries. (e.g. give me all instances of φ)

This completes our investigation of inference for $\mathcal{ALC}$. We summarize that $\mathcal{ALC}$ is a logic-based ontology language where the inference problems are all decidable/computable via $\mathcal{T}_{\mathcal{ALC}}$. But of course, while we have reached the expressivity of basic semantic networks, there are still things that we cannot express in $\mathcal{ALC}$, so we will try to extend $\mathcal{ALC}$ without losing decidability/computability.

16.4 Description Logics and the Semantic Web

A Video Nugget covering this section can be found at `https://fau.tv/clip/id/27289`.
In this section we discuss how we can apply description logics in the real world, in particular, as a conceptual and algorithmic basis of the semantic web. That tries to transform the World Wide Web from a human-understandable web of multimedia documents into a "web of machine-understandable data". In this context, "machine-understandable" means that machines can draw inferences from data they have access to. Note that the discussion in this digression is not a full-blown introduction to RDF and OWL, we leave that to [SR14; Her+13a; Hit+12] and the respective W3C recommendations. Instead we introduce the ideas behind the mappings from a perspective of the description logics we have discussed above.
The most important component of the semantic web is a standardized language that can represent "data" about information on the Web in a machine-oriented way.

Resource Description Framework

▷ **Definition 16.4.1.** The Resource Description Framework (RDF) is a framework for describing resources on the web. It is an XML vocabulary developed by the W3C.

▷ **Note:** RDF is designed to be read and understood by computers, not to be displayed to people. (it shows)

▷ **Example 16.4.2.** RDF can be used for describing (all "objects on the WWW")

- ▷ properties for shopping items, such as price and availability
- ▷ time schedules for web events
- ▷ information about web pages (content, author, created and modified date)
- ▷ content and rating for web pictures
- ▷ content for search engines
- ▷ electronic libraries

FAU Michael Kohlhase: Artificial Intelligence 1 538 2025-02-06

Note that all these examples have in common that they are about "objects on the Web", which is an aspect we will come to now.
"Objects on the Web" are traditionally called "resources", rather than defining them by their intrinsic properties – which would be ambitious and prone to change – we take an external property to define them: everything that has a URI is a web resource. This has repercussions on the design of RDF.

Resources and URIs

- ▷ RDF describes resources with properties and property values.
- ▷ RDF uses Web identifiers (URIs) to identify resources.
- ▷ **Definition 16.4.3.** A resource is anything that can have a URI, such as `http://www.fau.de`.
- ▷ **Definition 16.4.4.** A property is a resource that has a name, such as *author* or *homepage*, and a property value is the value of a property, such as *Michael Kohlhase* or `http://kwarc.info/kohlhase`. (a property value can be another resource)
- ▷ **Definition 16.4.5.** A RDF statement s (also known as a triple) consists of a resource (the subject of s), a property (the predicate of s), and a property value (the object of s). A set of RDF triples is called an RDF graph.
- ▷ **Example 16.4.6.** Statements: *[This slide]^{subj} has been [author]^{pred}ed by [Michael Kohlhase]^{obj}*

FAU Michael Kohlhase: Artificial Intelligence 1 539 2025-02-06

The crucial observation here is that if we map "subjects" and "objects" to "individuals", and "predicates" to "relations", the RDF triples are just relational ABox statements of description logics. As a consequence, the techniques we developed apply.
Note: Actually, a RDF graph is technically a labeled multigraph, which allows multiple edges between any two nodes (the resources) and where nodes and edges are labeled by URIs.
We now come to the concrete syntax of RDF. This is a relatively conventional XML syntax that combines RDF statements with a common subject into a single "description" of that resource.

XML Syntax for RDF

- ▷ RDF is a concrete XML vocabulary for writing statements
- ▷ **Example 16.4.7.** The following RDF document could describe the slides as a resource

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc= "http://purl.org/dc/elements/1.1/">
  <rdf:Description about="https://.../CompLog/kr/en/rdf.tex">
    <dc:creator>Michael Kohlhase</dc:creator>
    <dc:source>http://www.w3schools.com/rdf</dc:source>
  </rdf:Description>
</rdf:RDF>
```

This RDF document makes two statements:

- ▷ The subject of both is given in the about attribute of the rdf:Description element
- ▷ The predicates are given by the element names of its children
- ▷ The objects are given in the elements as URIs or literal content.

▷ **Intuitively:** RDF is a web-scalable way to write down ABox information.

Note that XML namespaces play a crucial role in using element to encode the predicate URIs. Recall that an element name is a qualified name that consists of a namespace URI and a proper element name (without a colon character). Concatenating them gives a URI in our example the predicate URI induced by the dc:creator element is `http://purl.org/dc/elements/1.1/creator`. Note that as URIs go RDF URIs do not have to be URLs, but this one is and it references (is redirected to) the relevant part of the Dublin Core elements specification [DCM12].

RDF was deliberately designed as a standoff markup format, where URIs are used to annotate web resources by pointing to them, so that it can be used to give information about web resources without having to change them. But this also creates maintenance problems, since web resources may change or be deleted without warning.

RDFa gives authors a way to embed RDF triples into web resources and make keeping RDF statements about them more in sync.

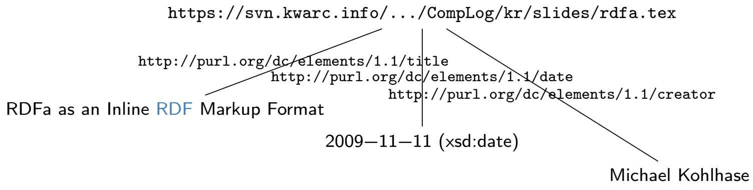
RDFa as an Inline RDF Markup Format

▷ **Problem:** RDF is a standoff markup format (annotate by URIs pointing into other files)

Definition 16.4.8. RDFa (RDF annotations) is a markup scheme for inline annotation (as XML attributes) of RDF triples.

▷ **Example 16.4.9.**

```
<div xmlns:dc="http://purl.org/dc/elements/1.1/" id="address">
  <h2 about="#address" property="dc:title">RDF as an Inline RDF Markup Format</h2>
  <h3 about="#address" property="dc:creator">Michael Kohlhase</h3>
  <em about="#address" property="dc:date" datatype="xsd:date"
      content="2009−11−11">November 11., 2009</em>
</div>
```



https://svn.kwarc.info/.../CompLog/kr/slides/rdfa.tex

http://purl.org/dc/elements/1.1/title

http://purl.org/dc/elements/1.1/date

http://purl.org/dc/elements/1.1/creator

RDFa as an Inline RDF Markup Format

2009−11−11 (xsd:date)

Michael Kohlhase

In the example above, the about and property attributes are reserved by RDFa and specify the subject and predicate of the RDF statement. The object consists of the body of the element, unless otherwise specified e.g. by the content and datatype attributes for literals content.

Let us now come back to the fact that RDF is just an XML syntax for ABox statements.

RDF as an ABox Language for the Semantic Web

- ▷ **Idea:** RDF triples are ABox entries h R s or $h{:}\varphi$.
- ▷ **Example 16.4.10.** h is the resource for Ian Horrocks, s is the resource for Ulrike Sattler, R is the relation "hasColleague", and φ is the class foaf:Person

```
<rdf:Description about="some.uri/person/ian_horrocks">
  <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  <hasColleague resource="some.uri/person/uli_sattler"/>
</rdf:Description>
```

- ▷ **Idea:** Now, we need an similar language for TBoxes (based on $\mathcal{ALC}$)

Michael Kohlhase: Artificial Intelligence 1 542 2025-02-06

In this situation, we want a standardized representation language for TBox information; OWL does just that: it standardizes a set of knowledge representation primitives and specifies a variety of concrete syntaxes for them. OWL is designed to be compatible with RDF, so that the two together can form an ontology language for the web.

OWL as an Ontology Language for the Semantic Web

- ▷ **Task:** Complement RDF (ABox) with a TBox language.
- ▷ **Idea:** Make use of resources that are values in rdf:type. (called Classes)
- ▷ **Definition 16.4.11.** OWL (the ontology web language) is a language for encoding TBox information about RDF classes.
- ▷ **Example 16.4.12 (A concept definition for "Mother").** Mother$=$Woman $\sqcap$ Parent is represented as

XML Syntax	Functional Syntax

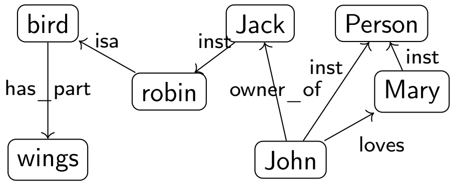
`<EquivalentClasses>`
`  <Class IRI="Mother"/>`
`  <ObjectIntersectionOf>`
`    <Class IRI="Woman"/>`
`    <Class IRI="Parent"/>`
`  </ObjectIntersectionOf>`
`</EquivalentClasses>`
	`EquivalentClasses(`
	`  :Mother`
	`  ObjectIntersectionOf(`
	`    :Woman`
	`    :Parent`
	`  )`
	`)`

Michael Kohlhase: Artificial Intelligence 1 543 2025-02-06

But there are also other syntaxes in regular use. We show the functional syntax which is inspired by the mathematical notation of relations.

Extended OWL Example in Functional Syntax

- ▷ **Example 16.4.13.** The semantic network from ?? can be expressed in OWL (in functional syntax)



```
ClassAssertion (:Jack :robin)
ClassAssertion(:John :person)
ClassAssertion (:Mary :person)
ObjectPropertyAssertion(:loves :John :Mary)
ObjectPropertyAssertion(:owner :John :Jack)
SubClassOf(:robin :bird)
SubClassOf (:bird ObjectSomeValuesFrom(:hasPart :wing))
```

- ▷ ClassAssertion formalizes the “inst” relation,
- ▷ ObjectPropertyAssertion formalizes relations,
- ▷ SubClassOf formalizes the “isa” relation,
- ▷ for the “has_part” relation, we have to specify that *all birds have a part that is a wing* or equivalently *the class of birds is a subclass of all objects that have some wing*.

We have introduced the ideas behind using description logics as the basis of a “machine-oriented web of data”. While the first OWL specification (2004) had three sublanguages “OWL Lite”, “OWL DL” and “OWL Full”, of which only the middle was based on description logics, with the OWL2 Recommendation from 2009, the foundation in description logics was nearly universally accepted.

The semantic web hype is by now nearly over, the technology has reached the “plateau of productivity” with many applications being pursued in academia and industry. We will not go into these, but briefly instroduce one of the tools that make this work.

SPARQL an RDF Query language

- ▷ **Definition 16.4.14.** SPARQL, the “SPARQL Protocol and RDF Query Language” is an RDF query language, able to retrieve and manipulate data stored in RDF. The SPARQL language was standardized by the World Wide Web Consortium in 2008 [PS08].
- ▷ SPARQL is pronounced like the word “*sparkle*”.
- ▷ **Definition 16.4.15.** A system is called a SPARQL endpoint, iff it answers SPARQL queries.
- ▷ **Example 16.4.16.** Query for person names and their e-mails from a triplestore with FOAF data.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person a foaf:Person.
  ?person foaf:name ?name.
  ?person foaf:mbox ?email.
}
```

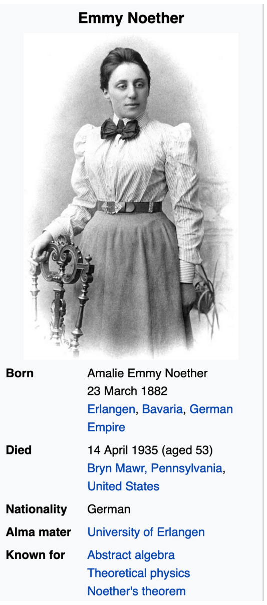
SPARQL end-points can be used to build interesting applications, if fed with the appropriate data. An interesting – and by now paradigmatic – example is the DBPedia project, which builds a large ontology by analyzing Wikipedia fact boxes. These are in a standard HTML form which can be analyzed e.g. by regular expressions, and their entries are essentially already in triple form: The subject is the Wikipedia page they are on, the predicate is the key, and the object is either the URI on the object value (if it carries a link) or the value itself.

SPARQL Applications: DBPedia

- ▷ **Typical Application:** DBPedia screen-scrapes Wikipedia fact boxes for RDF triples and uses SPARQL for querying the induced triplestore.
- ▷ **Example 16.4.17 (DBPedia Query).** People who were born in Erlangen before 1900 (http://dbpedia.org/snorql)

```
SELECT ?name ?birth ?death ?person WHERE {
       ?person dbo:birthPlace :Erlangen .
       ?person dbo:birthDate ?birth .
       ?person foaf:name ?name .
       ?person dbo:deathDate ?death .
       FILTER (?birth < "1900—01—01"^^xsd:date) .
}
ORDER BY ?name
```

- ▷ The answers include Emmy Noether and Georg Simon Ohm.

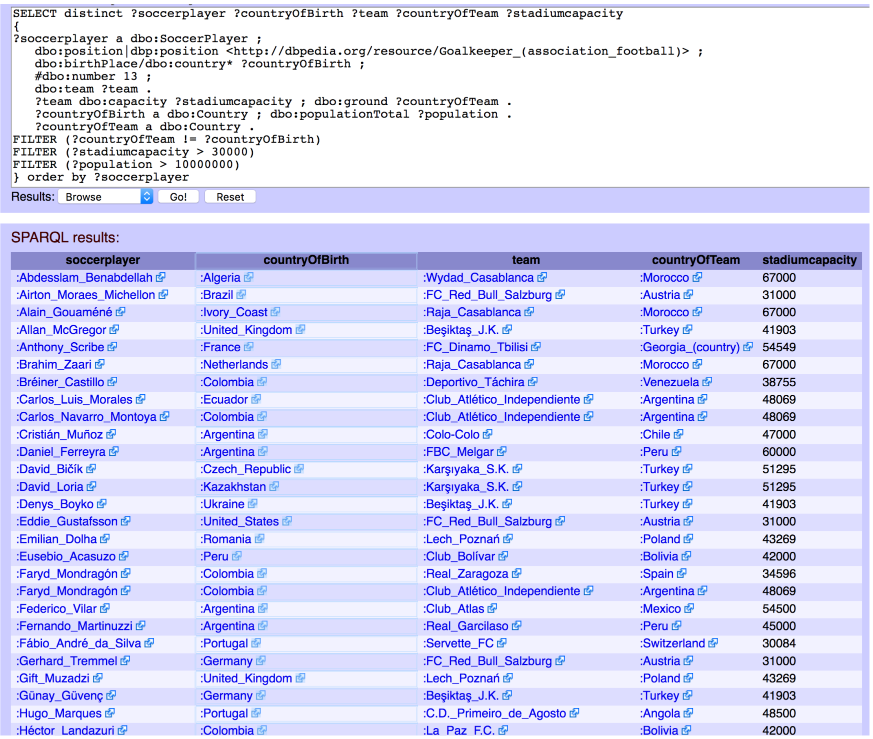


Emmy Noether

Born	Amalie Emmy Noether 23 March 1882 Erlangen, Bavaria, German Empire
Died	14 April 1935 (aged 53) Bryn Mawr, Pennsylvania, United States
Nationality	German
Alma mater	University of Erlangen
Known for	Abstract algebra Theoretical physics Noether's theorem

A more complex DBPedia Query

- ▷ **Demo:** DBPedia http://dbpedia.org/snorql/
  Query: Soccer players born in a country with more than 10 M inhabitants, who play as goalie in a club that has a stadium with more than 30.000 seats.
  Answer: computed by DBPedia from a SPARQL query



```
SELECT distinct ?soccerplayer ?countryOfBirth ?team ?countryOfTeam ?stadiumcapacity
{
?soccerplayer a dbo:SoccerPlayer ;
   dbo:position|dbp:position <http://dbpedia.org/resource/Goalkeeper_(association_football)> ;
   dbo:birthPlace/dbo:country* ?countryOfBirth ;
   #dbo:number 13 ;
   dbo:team ?team .
   ?team dbo:capacity ?stadiumcapacity ; dbo:ground ?countryOfTeam .
   ?countryOfBirth a dbo:Country ; dbo:populationTotal ?population .
   ?countryOfTeam a dbo:Country .
FILTER (?countryOfTeam != ?countryOfBirth)
FILTER (?stadiumcapacity > 30000)
FILTER (?population > 10000000)
} order by ?soccerplayer
```

SPARQL results:

soccerplayer	countryOfBirth	team	countryOfTeam	stadiumcapacity
:Abdesslam_Benabdellah	:Algeria	:Wydad_Casablanca	:Morocco	67000
:Airton_Moraes_Michellon	:Brazil	:FC_Red_Bull_Salzburg	:Austria	31000
:Alain_Gouaméné	:Ivory_Coast	:Raja_Casablanca	:Morocco	67000
:Allan_McGregor	:United_Kingdom	:Beşiktaş_J.K.	:Turkey	41903
:Anthony_Scribe	:France	:FC_Dinamo_Tbilisi	:Georgia_(country)	54549
:Brahim_Zaari	:Netherlands	:Raja_Casablanca	:Morocco	67000
:Bréiner_Castillo	:Colombia	:Deportivo_Táchira	:Venezuela	38755
:Carlos_Luis_Morales	:Ecuador	:Club_Atlético_Independiente	:Argentina	48069
:Carlos_Navarro_Montoya	:Colombia	:Club_Atlético_Independiente	:Argentina	48069
:Cristián_Muñoz	:Argentina	:Colo-Colo	:Chile	47000
:Daniel_Ferreyra	:Argentina	:FBC_Melgar	:Peru	60000
:David_Bičík	:Czech_Republic	:Karşıyaka_S.K.	:Turkey	51295
:David_Loria	:Kazakhstan	:Karşıyaka_S.K.	:Turkey	51295
:Denys_Boyko	:Ukraine	:Beşiktaş_J.K.	:Turkey	41903
:Eddie_Gustafsson	:United_States	:FC_Red_Bull_Salzburg	:Austria	31000
:Emilian_Dolha	:Romania	:Lech_Poznań	:Poland	43269
:Eusebio_Acasuzo	:Peru	:Club_Bolívar	:Bolivia	42000
:Faryd_Mondragón	:Colombia	:Real_Zaragoza	:Spain	34596
:Faryd_Mondragón	:Colombia	:Club_Atlético_Independiente	:Argentina	48069
:Federico_Vilar	:Argentina	:Club_Atlas	:Mexico	54500
:Fernando_Martinuzzi	:Argentina	:Real_Garcilaso	:Peru	45000
:Fábio_André_da_Silva	:Portugal	:Servette_FC	:Switzerland	30084
:Gerhard_Tremmel	:Germany	:FC_Red_Bull_Salzburg	:Austria	31000
:Gift_Muzadzi	:United_Kingdom	:Lech_Poznań	:Poland	43269
:Günay_Güvenç	:Germany	:Beşiktaş_J.K.	:Turkey	41903
:Hugo_Marques	:Portugal	:C.D._Primeiro_de_Agosto	:Angola	48500
:Héctor_Landazuri	:Colombia	:La_Paz_F.C.	:Bolivia	42000

We conclude our survey of the semantic web technology stack with the notion of a triplestore, which refers to the database component, which stores vast collections of ABox triples.

Triple Stores: the Semantic Web Databases

- ▷ **Definition 16.4.18.** A triplestore or RDF store is a purpose-built database for the storage RDF graphs and retrieval of RDF triples usually through variants of SPARQL.
- ▷ Common triplestores include
  - ▹ Virtuoso: `https://virtuoso.openlinksw.com/` (used in DBpedia)
  - ▹ GraphDB: `http://graphdb.ontotext.com/` (often used in WissKI)
  - ▹ blazegraph: `https://blazegraph.com/` (open source; used in WikiData)
- ▷ **Definition 16.4.19.** A description logic reasoner implements of reaonsing services based on a satisfiabiltiy test for description logics.
- ▷ Common description logic reasoners include
  - ▹ FACT++: `http://owl.man.ac.uk/factplusplus/`
  - ▹ HermiT: `http://www.hermit-reasoner.com/`
- ▷ **Intuition:** Triplestores concentrate on querying very large ABoxes with partial consideration of the TBox, while DL reasoners concentrate on the full set of ontology inference services, but fail on large ABoxes.

Bibliography

[BKS04] Paul Beame, Henry A. Kautz, and Ashish Sabharwal. “Towards Understanding and Harnessing the Potential of Clause Learning”. In: *Journal of Artificial Intelligence Research* 22 (2004), pp. 319–351.

[CKT91] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. “Where the *Really* Hard Problems Are”. In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI)*. Ed. by John Mylopoulos and Ray Reiter. Sydney, Australia: Morgan Kaufmann, San Mateo, CA, 1991, pp. 331–337.

[CQ69] Allan M. Collins and M. Ross Quillian. “Retrieval time from semantic memory”. In: *Journal of verbal learning and verbal behavior* 8.2 (1969), pp. 240–247. DOI: `10.1016/S0022-5371(69)80069-1`.

[DCM12] DCMI Usage Board. *DCMI Metadata Terms*. DCMI Recommendation. Dublin Core Metadata Initiative, June 14, 2012. URL: `http://dublincore.org/documents/2012/06/14/dcmi-terms/`.

[Gen34] Gerhard Gentzen. “Untersuchungen über das logische Schließen I”. In: *Mathematische Zeitschrift* 39.2 (1934), pp. 176–210.

[GS05] Carla Gomes and Bart Selman. “Can get satisfaction”. In: *Nature* 435 (2005), pp. 751–752.

[Her+13a] Ivan Herman et al. *RDF 1.1 Primer (Second Edition). Rich Structured Data Markup for Web Documents*. W3C Working Group Note. World Wide Web Consortium (W3C), 2013. URL: `http://www.w3.org/TR/rdfa-primer`.

[Her+13b] Ivan Herman et al. *RDFa 1.1 Primer – Second Edition. Rich Structured Data Markup for Web Documents*. W3C Working Goup Note. World Wide Web Consortium (W3C), Apr. 19, 2013. URL: `http://www.w3.org/TR/xhtml-rdfa-primer/`.

[Hit+12] Pascal Hitzler et al. *OWL 2 Web Ontology Language Primer (Second Edition)*. W3C Recommendation. World Wide Web Consortium (W3C), 2012. URL: `http://www.w3.org/TR/owl-primer`.

[KC04] Graham Klyne and Jeremy J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation. World Wide Web Consortium (W3C), Feb. 10, 2004. URL: `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`.

[Kow97] Robert Kowalski. “Algorithm = Logic + Control”. In: *Communications of the Association for Computing Machinery* 22 (1997), pp. 424–436.

[MSL92] David Mitchell, Bart Selman, and Hector J. Levesque. “Hard and Easy Distributions of SAT Problems”. In: *Proceedings of the 10th National Conference of the American Association for Artificial Intelligence (AAAI’92)*. San Jose, CA: MIT Press, 1992, pp. 459–465.

[OWL09] OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation. World Wide Web Consortium (W3C), Oct. 27, 2009. URL: `http://www.w3.org/TR/2009/REC-owl2-overview-20091027/`.

[PD09] Knot Pipatsrisawat and Adnan Darwiche. “On the Power of Clause-Learning SAT Solvers with Restarts”. In: *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP’09)*. Ed. by Ian P. Gent. Vol. 5732. Lecture Notes in Computer Science. Springer, 2009, pp. 654–668.

[Pro] *Protégé*. Project Home page at `http://protege.stanford.edu`. URL: `http://protege.stanford.edu`.

[PRR97] G. Probst, St. Raub, and Kai Romhardt. *Wissen managen*. 4 (2003). Gabler Verlag, 1997.

[PS08] Eric Prud’hommeaux and Andy Seaborne. *SPARQL Query Language for RDF*. W3C Recommendation. World Wide Web Consortium (W3C), Jan. 15, 2008. URL: `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/`.

[RN09] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. Prentice Hall Press, 2009. ISBN: 0136042597, 9780136042594.

[Smu63] Raymond M. Smullyan. “A Unifying Principle for Quantification Theory”. In: *Proc. Nat. Acad Sciences* 49 (1963), pp. 828–832.

[SR14] Guus Schreiber and Yves Raimond. *RDF 1.1 Primer*. W3C Working Group Note. World Wide Web Consortium (W3C), 2014. URL: `http://www.w3.org/TR/rdf-primer`.

Appendix A

Excursions

As this course is predominantly an overview over the topics of Artificial Intelligence, and not about the theoretical underpinnings, we give the discussion about these as a "suggested readings" chapter here.

A.1 Completeness of Calculi for Propositional Logic

The next step is to analyze the two calculi for completeness. For that we will first give ourselves a very powerful tool: the "model existence theorem" (**??**), which encapsulates the model-theoretic part of completeness theorems. With that, completeness proofs – which are quite tedious otherwise – become a breeze.

A.1.1 Abstract Consistency and Model Existence

We will now come to an important tool in the theoretical study of reasoning calculi: the "abstract consistency"/"model existence" method. This method for analyzing calculi was developed by Jaako Hintikka, Raymond Smullyan, and Peter Andrews in 1950-1970 as an encapsulation of similar constructions that were used in completeness arguments in the decades before. The basis for this method is Smullyan's Observation [Smu63] that completeness proofs based on Hintikka sets only certain properties of consistency and that with little effort one can obtain a generalization "Smullyan's Unifying Principle".

The basic intuition for this method is the following: typically, a logical system $\mathcal{L} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ has multiple calculi, human-oriented ones like the natural deduction calculi and machine-oriented ones like the automated theorem proving calculi. All of these need to be analyzed for completeness (as a basic quality assurance measure).

A completeness proof for a calculus $\mathcal{C}$ for $\mathcal{S}$ typically comes in two parts: one analyzes $\mathcal{C}$-consistency (sets that cannot be refuted in $\mathcal{C}$), and the other construct $\mathcal{K}$-models for $\mathcal{C}$-consistent sets.

In this situtation the "abstract consistency"/"model existence" method encapsulates the model construction process into a meta-theorem: the "model existence" theorem. This provides a set of syntactic ("abstract consistency") conditions for calculi that are sufficient to construct models.

With the model existence theorem it suffices to show that $\mathcal{C}$-consistency is an abstract consistency property (a purely syntactic task that can be done by a $\mathcal{C}$-proof transformation argument) to obtain a completeness result for $\mathcal{C}$.

Model Existence (Overview)

▷ **Definition:** Abstract consistency

- ▷ **Definition:** Hintikka set (maximally abstract consistent)
- ▷ **Theorem:** Hintikka sets are satisfiable
- ▷ **Theorem:** If Φ is abstract consistent, then Φ can be extended to a Hintikka set.
- ▷ **Corollary:** If Φ is abstract consistent, then Φ is satisfiable.
- ▷ **Application:** Let $\mathcal{C}$ be a calculus, if Φ is $\mathcal{C}$-consistent, then Φ is abstract consistent.
- ▷ **Corollary:** $\mathcal{C}$ is complete.

The proof of the model existence theorem goes via the notion of a Hintikka set, a set of formulae with very strong syntactic closure properties, which allow to read off models. Jaako Hintikka's original idea for completeness proofs was that for every complete calculus $\mathcal{C}$ and every $\mathcal{C}$-consistent set one can induce a Hintikka set, from which a model can be constructed. This can be considered as a first model existence theorem. However, the process of obtaining a Hintikka set for a $\mathcal{C}$-consistent set Φ of sentences usually involves complicated calculus dependent constructions.

In this situation, Raymond Smullyan was able to formulate the sufficient conditions for the existence of Hintikka sets in the form of "abstract consistency properties" by isolating the calculus independent parts of the Hintikka set construction. His technique allows to reformulate Hintikka sets as maximal elements of abstract consistency classes and interpret the Hintikka set construction as a maximizing limit process.

To carry out the "model-existence"/"abstract consistency" method, we will first have to look at the notion of consistency.

Consistency and refutability are very important notions when studying the completeness for calculi; they form syntactic counterparts of satisfiability.

Consistency

- ▷ Let $\mathcal{C}$ be a calculus,. . .
- ▷ **Definition A.1.1.** Let $\mathcal{C}$ be a calculus, then a formula set Φ is called $\mathcal{C}$-refutable, if there is a refutation, i.e. a derivation of a contradiction from Φ. The act of finding a refutation for Φ is called refuting Φ.
- ▷ **Definition A.1.2.** We call a pair of formulae $\mathbf{A}$ and $\neg\mathbf{A}$ a contradiction.
- ▷ So a set Φ is $\mathcal{C}$-refutable, if $\mathcal{C}$ canderive a contradiction from it.
- ▷ **Definition A.1.3.** Let $\mathcal{C}$ be a calculus, then a formula set Φ is called $\mathcal{C}$-consistent, iff there is a formula $\mathbf{B}$, that is not derivable from Φ in $\mathcal{C}$.
- ▷ **Definition A.1.4.** We call a calculus $\mathcal{C}$ reasonable, iff implication elimination and conjunction introduction are admissible in $\mathcal{C}$ and $\mathbf{A} \wedge \neg\mathbf{A} \Rightarrow \mathbf{B}$ is a $\mathcal{C}$-theorem.
- ▷ **Theorem A.1.5.** *$\mathcal{C}$-inconsistency and $\mathcal{C}$-refutability coincide for reasonable calculi.*

It is very important to distinguish the syntactic $\mathcal{C}$-refutability and $\mathcal{C}$-consistency from satisfiability, which is a property of formulae that is at the heart of semantics. Note that the former have the calculus (a syntactic device) as a parameter, while the latter does not. In fact we should actually say $\mathcal{S}$-satisfiability, where $\langle\mathcal{L},\mathcal{K},\models\rangle$ is the current logical system.

Even the word "contradiction" has a syntactical flavor to it, it translates to "saying against each other" from its Latin root.

Abstract Consistency

▷ **Definition A.1.6.** Let ∇ be a collection of sets. We call ∇ closed under subsets, iff for each $\Phi \in \nabla$, all subsets $\Psi \subseteq \Phi$ are elements of ∇.

▷ **Definition A.1.7 (Notation).** We will use $\Phi * \mathbf{A}$ for $\Phi \cup \{\mathbf{A}\}$.

▷ **Definition A.1.8.** A collection ∇ of sets of propositional formulae is called an abstract consistency class, iff it is closed under subsets, and for each $\Phi \in \nabla$

∇_c) $P \notin \Phi$ or $\neg P \notin \Phi$ for $P \in \mathcal{V}_0$

$\nabla_\neg$) $\neg\neg\mathbf{A} \in \Phi$ implies $\Phi * \mathbf{A} \in \nabla$

$\nabla_\vee$) $\mathbf{A} \vee \mathbf{B} \in \Phi$ implies $\Phi * \mathbf{A} \in \nabla$ or $\Phi * \mathbf{B} \in \nabla$

$\nabla_\wedge$) $\neg(\mathbf{A} \vee \mathbf{B}) \in \Phi$ implies $\Phi \cup \{\neg\mathbf{A}, \neg\mathbf{B}\} \in \nabla$

▷ **Example A.1.9.** The empty set is an abstract consistency class.

▷ **Example A.1.10.** The set $\{\emptyset, \{Q\}, \{P \vee Q\}, \{P \vee Q, Q\}\}$ is an abstract consistency class.

▷ **Example A.1.11.** The family of satisfiable sets is an abstract consistency class.

FAU Michael Kohlhase: Artificial Intelligence 1 551 2025-02-06

So a family of sets (we call it a family, so that we do not have to say "set of sets" and we can distinguish the levels) is an abstract consistency class, iff it fulfills five simple conditions, of which the last three are closure conditions.

Think of an abstract consistency class as a family of "consistent" sets (e.g. $\mathcal{C}$-consistent for some calculus $\mathcal{C}$), then the properties make perfect sense: They are naturally closed under subsets — if we cannot derive a contradiction from a large set, we certainly cannot from a subset, furthermore,

∇_c) If both $P \in \Phi$ and $\neg P \in \Phi$, then Φ cannot be "consistent".

$\nabla_\neg$) If we cannot derive a contradiction from Φ with $\neg\neg\mathbf{A} \in \Phi$ then we cannot from $\Phi * \mathbf{A}$, since they are logically equivalent.

The other two conditions are motivated similarly. We will carry out the proof here, since it gives us practice in dealing with the abstract consistency properties.

The main result here is that abstract consistency classes can be extended to compact ones. The proof is quite tedious, but relatively straightforward. It allows us to assume that all abstract consistency classes are compact in the first place (otherwise we pass to the compact extension).

Actually we are after abstract consistency classes that have an even stronger property than just being closed under subsets. This will allow us to carry out a limit construction in the Hintikka set extension argument later.

Compact Collections

▷ **Definition A.1.12.** We call a collection ∇ of sets compact, iff for any set Φ we have

$\Phi \in \nabla$, iff $\Psi \in \nabla$ for every finite subset Ψ of Φ.

▷ **Lemma A.1.13.** *If ∇ is compact, then ∇ is closed under subsets.*

▷ *Proof:*

1. Suppose $S \subseteq T$ and $T \in \nabla$.
2. Every finite subset A of S is a finite subset of T.
3. As ∇ is compact, we know that $A \in \nabla$.
4. Thus $S \in \nabla$.

The property of being closed under subsets is a "downwards-oriented" property: We go from large sets to small sets, compactness (the interesting direction anyways) is also an "upwards-oriented" property. We can go from small (finite) sets to large (infinite) sets. The main application for the compactness condition will be to show that infinite sets of formulae are in a collection ∇ by testing all their finite subsets (which is much simpler).

Compact Abstract Consistency Classes

▷ **Lemma A.1.14.** *Any abstract consistency class can be extended to a compact one.*

▷ *Proof:*

1. We choose $\nabla' := \{\Phi \subseteq \mathit{wff}_0(\mathcal{V}_0) \,|\, \text{every finite subset of } \Phi \text{ is in } \nabla\}$.

2. Now suppose that $\Phi \in \nabla$. ∇ is closed under subsets, so every finite subset of Φ is in ∇ and thus $\Phi \in \nabla'$. Hence $\nabla \subseteq \nabla'$.

3. Next let us show that each ∇ is compact.'

3.1. Suppose $\Phi \in \nabla'$ and Ψ is an arbitrary finite subset of Φ.

3.2. By definition of ∇' all finite subsets of Φ are in ∇ and therefore $\Psi \in \nabla'$.

3.3. Thus all finite subsets of Φ are in ∇' whenever Φ is in ∇'.

3.4. On the other hand, suppose all finite subsets of Φ are in ∇'.

3.5. Then by the definition of ∇' the finite subsets of Φ are also in ∇, so $\Phi \in \nabla'$. Thus ∇' is compact.

4. Note that ∇' is closed under subsets by the Lemma above.

5. Now we show that if ∇ satisfies ∇_*, then ∇ satisfies ∇_*.'

5.1. To show ∇_c, let $\Phi \in \nabla'$ and suppose there is an atom $\mathbf{A}$, such that $\{\mathbf{A}, \neg\mathbf{A}\} \subseteq \Phi$. Then $\{\mathbf{A}, \neg\mathbf{A}\} \in \nabla$ contradicting ∇_c.

5.2. To show $\nabla_\neg$, let $\Phi \in \nabla'$ and $\neg\neg\mathbf{A} \in \Phi$, then $\Phi * \mathbf{A} \in \nabla'$.

5.2.1. Let Ψ be any finite subset of $\Phi * \mathbf{A}$, and $\Theta := (\Psi \backslash \{\mathbf{A}\}) * \neg\neg\mathbf{A}$.

5.2.2. Θ is a finite subset of Φ, so $\Theta \in \nabla$.

5.2.3. Since ∇ is an abstract consistency class and $\neg\neg\mathbf{A} \in \Theta$, we get $\Theta * \mathbf{A} \in \nabla$ by $\nabla_\neg$.

5.2.4. We know that $\Psi \subseteq \Theta * \mathbf{A}$ and ∇ is closed under subsets, so $\Psi \in \nabla$.

5.2.5. Thus every finite subset Ψ of $\Phi * \mathbf{A}$ is in ∇ and therefore by definition $\Phi * \mathbf{A} \in \nabla'$.

5.3. the other cases are analogous to $\nabla_\neg$.

Hintikka sets are sets of sentences with very strong analytic closure conditions. These are motivated as maximally consistent sets i.e. sets that already contain everything that can be consistently added to them.

∇-Hintikka Set

▷ **Definition A.1.15.** Let ∇ be an abstract consistency class, then we call a set $\mathcal{H}\in\nabla$ a ∇ Hintikka Set, iff $\mathcal{H}$ is maximal in ∇, i.e. for all $\mathbf{A}$ with $\mathcal{H}*\mathbf{A}\in\nabla$ we already have $\mathbf{A}\in\mathcal{H}$.

▷ **Theorem A.1.16 (Hintikka Properties).** *Let ∇ be an abstract consistency class and $\mathcal{H}$ be a ∇-Hintikka set, then*

$\mathcal{H}_c$) *For all $\mathbf{A}\in wff_0(\mathcal{V}_0)$ we have $\mathbf{A}\notin\mathcal{H}$ or $\neg\mathbf{A}\notin\mathcal{H}$*

$\mathcal{H}_\neg$) *If $\neg\neg\mathbf{A}\in\mathcal{H}$ then $\mathbf{A}\in\mathcal{H}$*

$\mathcal{H}_\vee$) *If $\mathbf{A}\vee\mathbf{B}\in\mathcal{H}$ then $\mathbf{A}\in\mathcal{H}$ or $\mathbf{B}\in\mathcal{H}$*

$\mathcal{H}_\wedge$) *If $\neg(\mathbf{A}\vee\mathbf{B})\in\mathcal{H}$ then $\neg\mathbf{A},\neg\mathbf{B}\in\mathcal{H}$*

∇-Hintikka Set

▷ *Proof:*

We prove the properties in turn

1. $\mathcal{H}_c$ *by induction on the structure of* $\mathbf{A}$
   1.1. $\mathbf{A}\in\mathcal{V}_0$ Then $\mathbf{A}\notin\mathcal{H}$ or $\neg\mathbf{A}\notin\mathcal{H}$ by ∇_c.
   1.2. $\mathbf{A}=\neg\mathbf{B}$
      1.2.1. Let us assume that $\neg\mathbf{B}\in\mathcal{H}$ and $\neg\neg\mathbf{B}\in\mathcal{H}$,
      1.2.2. then $\mathcal{H}*\mathbf{B}\in\nabla$ by $\nabla_\neg$, and therefore $\mathbf{B}\in\mathcal{H}$ by maximality.
      1.2.3. So both $\mathbf{B}$ and $\neg\mathbf{B}$ are in $\mathcal{H}$, which contradicts the induction hypothesis.
   1.3. $\mathbf{A}=\mathbf{B}\vee\mathbf{C}$ *similar to the previous case*
2. We prove $\mathcal{H}_\neg$ by maximality of $\mathcal{H}$ in ∇.
   2.1. If $\neg\neg\mathbf{A}\in\mathcal{H}$, then $\mathcal{H}*\mathbf{A}\in\nabla$ by $\nabla_\neg$.
   2.2. The maximality of $\mathcal{H}$ now gives us that $\mathbf{A}\in\mathcal{H}$.

Proof sketch: *other $\mathcal{H}_*$ are similar*

The following theorem is one of the main results in the “abstract consistency”/”model existence” method. For any abstract consistent set Φ it allows us to construct a Hintikka set $\mathcal{H}$ with $\Phi\in\mathcal{H}$.

Extension Theorem

▷ **Theorem A.1.17.** *If ∇ is an abstract consistency class and $\Phi\in\nabla$, then there is a ∇-Hintikka set $\mathcal{H}$ with $\Phi\subseteq\mathcal{H}$.*

▷ *Proof:*

1. Wlog. we assume that ∇ is compact (otherwise pass to compact extension)
2. We choose an enumeration $\mathbf{A}_1,\ldots$ of the set $wff_0(\mathcal{V}_0)$

3. and construct a sequence of sets $\mathbf{H}_i$ with $\mathbf{H}_0 := \Phi$ and

$$\mathbf{H}_{n+1} := \begin{cases} \mathbf{H}_n & \text{if } \mathbf{H}_n * \mathbf{A}_n \notin \nabla \\ \mathbf{H}_n * \mathbf{A}_n & \text{if } \mathbf{H}_n * \mathbf{A}_n \in \nabla \end{cases}$$

4. Note that all $\mathbf{H}_i \in \nabla$, choose $\mathcal{H} := \bigcup_{i \in \mathbb{N}} \mathbf{H}_i$
5. $\Psi \subseteq \mathcal{H}$ finite implies there is a $j \in \mathbb{N}$ such that $\Psi \subseteq \mathbf{H}_j$,
6. so $\Psi \in \nabla$ as ∇ is closed under subsets and $\mathcal{H} \in \nabla$ as ∇ is compact.
7. Let $\mathcal{H} * \mathbf{B} \in \nabla$, then there is a $j \in \mathbb{N}$ with $\mathbf{B} = \mathbf{A}_j$, so that $\mathbf{B} \in \mathbf{H}_{j+1}$ and $\mathbf{H}_{j+1} \subseteq \mathcal{H}$
8. Thus $\mathcal{H}$ is ∇-maximal

Note that the construction in the proof above is non-trivial in two respects. First, the limit construction for $\mathcal{H}$ is not executed in our original abstract consistency class ∇, but in a suitably extended one to make it compact — the original would not have contained $\mathcal{H}$ in general. Second, the set $\mathcal{H}$ is not unique for Φ, but depends on the choice of the enumeration of $\mathit{wff}_0(\mathcal{V}_0)$. If we pick a different enumeration, we will end up with a different $\mathcal{H}$. Say if $\mathbf{A}$ and $\neg\mathbf{A}$ are both ∇-consistent[1] with Φ, then depending on which one is first in the enumeration $\mathcal{H}$, will contain that one; with all the consequences for subsequent choices in the construction process.

Valuation

▷ **Definition A.1.18.** A function $\nu\colon \mathit{wff}_0(\mathcal{V}_0) \to \mathcal{D}_o$ is called a (propositional) valuation, iff

- ▷ $\nu(\neg\mathbf{A}) = \mathsf{T}$, iff $\nu(\mathbf{A}) = \mathsf{F}$
- ▷ $\nu(\mathbf{A} \wedge \mathbf{B}) = \mathsf{T}$, iff $\nu(\mathbf{A}) = \mathsf{T}$ and $\nu(\mathbf{B}) = \mathsf{T}$

▷ **Lemma A.1.19.** *If $\nu\colon \mathit{wff}_0(\mathcal{V}_0) \to \mathcal{D}_o$ is a valuation and $\Phi \subseteq \mathit{wff}_0(\mathcal{V}_0)$ with $\nu(\Phi) = \{\mathsf{T}\}$, then Φ is satisfiable.*

▷ *Proof sketch:* $\nu|_{\mathcal{V}_0} : \mathcal{V}_0 \to \mathcal{D}_o$ is a satisfying variable assignment.

▷ **Lemma A.1.20.** *If $\varphi\colon \mathcal{V}_0 \to \mathcal{D}_o$ is a variable assignment, then $\mathcal{I}_\varphi\colon \mathit{wff}_0(\mathcal{V}_0) \to \mathcal{D}_o$ is a valuation.*

Now, we only have to put the pieces together to obtain the model existence theorem we are after.

Model Existence

▷ **Lemma A.1.21 (Hintikka-Lemma).** *If ∇ is an abstract consistency class and $\mathcal{H}$ a ∇-Hintikka set, then $\mathcal{H}$ is satisfiable.*

▷ *Proof:*

1. We define $\nu(\mathbf{A}) := \mathsf{T}$, iff $\mathbf{A} \in \mathcal{H}$
2. then ν is a valuation by the Hintikka properties
3. and thus $\nu|_{\mathcal{V}_0}$ is a satisfying assignment.

[1] EdNote: introduce this above

▷ **Theorem A.1.22 (Model Existence).** *If ∇ is an abstract consistency class and $\Phi \in \nabla$, then Φ is satisfiable.*

Proof:

▷ 1. There is a ∇-Hintikka set $\mathcal{H}$ with $\Phi \subseteq \mathcal{H}$ (Extension Theorem)
2. We know that $\mathcal{H}$ is satisfiable. (Hintikka-Lemma)
3. In particular, $\Phi \subseteq \mathcal{H}$ is satisfiable.

Michael Kohlhase: Artificial Intelligence 1 559 2025-02-06

A.1.2 A Completeness Proof for Propositional Tableaux

With the model existence proof we have introduced in the last section, the completeness proof for first-order natural deduction is rather simple, we only have to check that Tableaux-consistency is an abstract consistency property.

We encapsulate all of the technical difficulties of the problem in a technical Lemma. From that, the completeness proof is just an application of the high-level theorems we have just proven.

Abstract Completeness for $\mathcal{T}_0$

▷ **Lemma A.1.23.** *$\{\Phi \mid \Phi^{\mathsf{T}}$ has no closed tableau$\}$ is an abstract consistency class.*

▷ *Proof:* Let's call the set above ∇

We have to convince ourselves of the abstract consistency properties

1. $\nabla_c P, \neg P \in \Phi$ *implies* $P^{\mathsf{F}}, P^{\mathsf{T}} \in \Phi^{\mathsf{T}}$.
2. $\nabla_\neg$ *Let* $\neg\neg\mathbf{A} \in \Phi$.
   2.1. For the proof of the contrapositive we assume that $\Phi{*}\mathbf{A}$ has a closed tableau $\mathcal{T}$ and show that already Φ has one:
   2.2. applying each of $\mathcal{T}_0\neg^{\mathsf{T}}$ and $\mathcal{T}_0\neg^{\mathsf{F}}$ once allows to extend any tableau with $\neg\neg\mathbf{B}^\alpha$ by $\mathbf{B}^\alpha$.
   2.3. any path in $\mathcal{T}$ that is closed with $\neg\neg\mathbf{A}^\alpha$, can be closed by $\mathbf{A}^\alpha$.
3. $\nabla_\vee$ *Suppose* $\mathbf{A} \vee \mathbf{B} \in \Phi$ *and both* $\Phi{*}\mathbf{A}$ *and* $\Phi{*}\mathbf{B}$ *have closed tableaux*
   3.1. consider the tableaux:

$$\begin{array}{c} \Phi^{\mathsf{T}} \\ \mathbf{A}^{\mathsf{T}} \\ Rest^1 \end{array} \qquad \begin{array}{c} \Phi^{\mathsf{T}} \\ \mathbf{B}^{\mathsf{T}} \\ Rest^2 \end{array} \qquad \begin{array}{c|c} \multicolumn{2}{c}{\Psi^{\mathsf{T}}} \\ \multicolumn{2}{c}{(\mathbf{A} \vee \mathbf{B})^{\mathsf{T}}} \\ \mathbf{A}^{\mathsf{T}} & \mathbf{B}^{\mathsf{T}} \\ Rest^1 & Rest^2 \end{array}$$

4. $\nabla_\wedge$ *suppose,* $\neg(\mathbf{A} \vee \mathbf{B}) \in \Phi$ *and* $\Phi\{\neg\mathbf{A}, \neg\mathbf{B}\}$ *have closed tableau* $\mathcal{T}$.
   4.1. We consider

$$\begin{array}{c} \Phi^{\mathsf{T}} \\ \mathbf{A}^{\mathsf{F}} \\ \mathbf{B}^{\mathsf{F}} \\ Rest \end{array} \qquad \begin{array}{c} \Psi^{\mathsf{T}} \\ (\mathbf{A} \vee \mathbf{B})^{\mathsf{F}} \\ \mathbf{A}^{\mathsf{F}} \\ \mathbf{B}^{\mathsf{F}} \\ Rest \end{array}$$

   where $\Phi = \Psi{*}\neg(\mathbf{A} \vee \mathbf{B})$.

Michael Kohlhase: Artificial Intelligence 1 561 2025-02-06

Observation: If we look at the completeness proof below, we see that the Lemma above is the only place where we had to deal with specific properties of the $\mathcal{T}_0$.

So if we want to prove completeness of any other calculus with respect to propositional logic, then we only need to prove an analogon to this lemma and can use the rest of the machinery we have already established "off the shelf".

This is one great advantage of the "abstract consistency method"; the other is that the method can be extended transparently to other logics.

Completeness of $\mathcal{T}_0$

▷ **Corollary A.1.24.** *$\mathcal{T}_0$ is complete.*

▷ *Proof:* by contradiction

1. We assume that $\mathbf{A} \in wff_0(\mathcal{V}_0)$ is valid, but there is no closed tableau for $\mathbf{A}^{\mathsf{F}}$.
2. We have $\{\neg\mathbf{A}\} \in \nabla$ as $\neg\mathbf{A}^{\mathsf{T}} = \mathbf{A}^{\mathsf{F}}$.
3. so $\neg\mathbf{A}$ is satisfiable by the model existence theorem (which is applicable as ∇ is an abstract consistency class by our Lemma above)
4. this contradicts our assumption that $\mathbf{A}$ is valid.

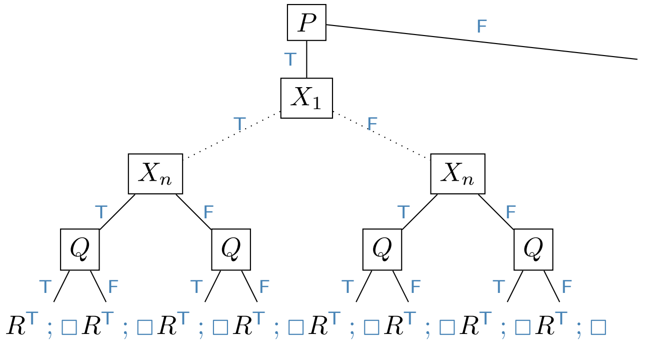
A.2 Conflict Driven Clause Learning

A.2.1 *Why* Did Unit Propagation Yield a Conflict?

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/27026`.

DPLL: Example (Redundance1)

▷ **Example A.2.1.** We introduce some nasty redundance to make DPLL slow.
$\Delta := P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{F}}$
DPLL on $\Delta ; \Theta$ with $\Theta := {X_1}^{\mathsf{T}} \vee \ldots \vee {X_n}^{\mathsf{T}} ; {X_1}^{\mathsf{F}} \vee \ldots \vee {X_n}^{\mathsf{F}}$



How To *Not* Make the Same Mistakes Over Again?

▷ **It's not that difficult, really:**

(A) Figure out what went wrong.

(B) Learn to not do that again in the future.

▷ **And now for DPLL:**

(A) *Why* did unit propagation yield a Conflict?

- ▷ This Section. We will capture the "what went wrong" in terms of graphs over literals set during the search, and their dependencies.

▷ **What can we learn from that information?:**

- ▷ A new clause! Next section.

Implication Graphs for DPLL

▷ **Definition A.2.2.** Let β be a branch in a DPLL derivation and P a variable on β then we call

- ▷ P^α a choice literal if its value is set to α by the splitting rule.
- ▷ P^α an implied literal, if the value of P is set to α by the UP rule.
- ▷ P^α a conflict literal, if it contributes to a derivation of the empty clause.

▷ **Definition A.2.3 (Implication Graph).**

Let Δ be a clause set, β a DPLL search branch on Δ. The implication graph G_β^{impl} is the directed graph whose vertices are labeled with the choice and implied literals along β, as well as a separate conflict vertex $\Box_C$ for every clause C that became empty on β.

Whereever a clause $l_1, \ldots, l_k \vee l' \in \Delta$ became unit with implied literal l', G_β^{impl} includes the edges $(\overline{l_i}, l')$.

Where $C = l_1 \vee \ldots \vee l_k \in \Delta$ became empty, G_β^{impl} includes the edges $(\overline{l_i}, \Box_C)$.

▷ **Question:** How do we know that $\overline{l_i}$ are vertices in G_β^{impl}?

▷ **Answer:** Because $l_1 \vee \ldots \vee l_k \vee l'$ became unit/empty.

▷ **Observation A.2.4.** G_β^{impl} *is acyclic.*

▷ *Proof sketch:* UP can't derive l' whose value was already set beforehand.

▷ **Intuition:** The initial vertices are the choice literals and unit clauses of Δ.

Implication Graphs: Example (Vanilla1) in Detail

▷ **Example A.2.5.** Let $\Delta := P^{\mathsf{T}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{F}} ; P^{\mathsf{F}} \vee Q^{\mathsf{F}} ; R^{\mathsf{T}} ; P^{\mathsf{T}} \vee Q^{\mathsf{F}}$.

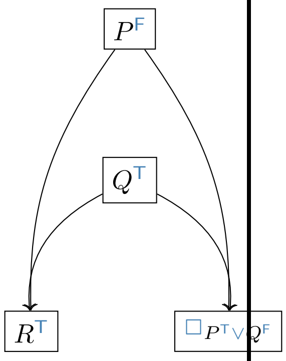
We look at the left branch of the derivation from ??:

1. UP Rule: $R \mapsto \mathsf{T}$
   Implied literal R^{T}.
   $P^{\mathsf{T}} \vee Q^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{F}} ; P^{\mathsf{T}} \vee Q^{\mathsf{F}}$
2. Splitting Rule:

   2a. $P \mapsto \mathsf{F}$
   Choice literal P^{F}.
   $Q^{\mathsf{T}} ; Q^{\mathsf{F}}$

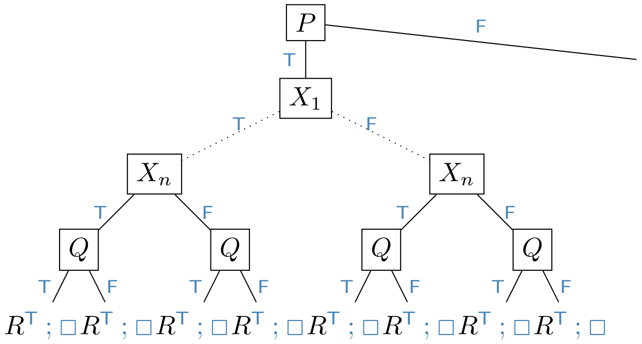
   3a. UP Rule: $Q \mapsto \mathsf{T}$
   Implied literal Q^{T}
   edges $(R^{\mathsf{T}}, Q^{\mathsf{T}})$ and $(P^{\mathsf{F}}, Q^{\mathsf{T}})$.
   $\Box$
   Conflict vertex $\Box_{P^{\mathsf{T}} \vee Q^{\mathsf{F}}}$
   edges $(P^{\mathsf{F}}, \Box_{P^{\mathsf{T}} \vee Q^{\mathsf{F}}})$ and $(Q^{\mathsf{T}}, \Box_{P^{\mathsf{T}} \vee Q^{\mathsf{F}}})$.

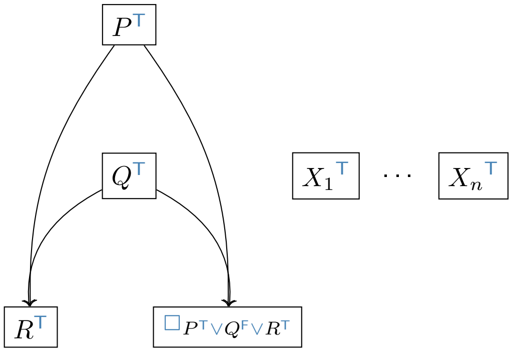
Implication graph:



Implication Graphs: Example (Redundance1)

▷ **Example A.2.6.** Continuing from ??: $\Delta := P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{F}}$
DPLL on $\Delta ; \Theta$ with $\Theta := {X_1}^{\mathsf{T}} \vee \ldots \vee {X_n}^{\mathsf{T}} ; {X_1}^{\mathsf{F}} \vee \ldots \vee {X_n}^{\mathsf{F}}$
Choice literals: P^{T}, $({X_1}^{\mathsf{T}}), \ldots, ({X_n}^{\mathsf{T}})$, Q^{T}. Implied literal: R^{T}.


P
F
T
X_1
T
F
X_n
X_n
T
F
T
F
Q
Q
Q
Q
T
F
T
F
T
F
T
F
$R^{\mathsf{T}} ; \Box R^{\mathsf{T}} ; \Box R^{\mathsf{T}} ; \Box R^{\mathsf{T}} ; \Box R^{\mathsf{T}} ; \Box R^{\mathsf{T}} ; \Box R^{\mathsf{T}} ; \Box R^{\mathsf{T}} ; \Box$

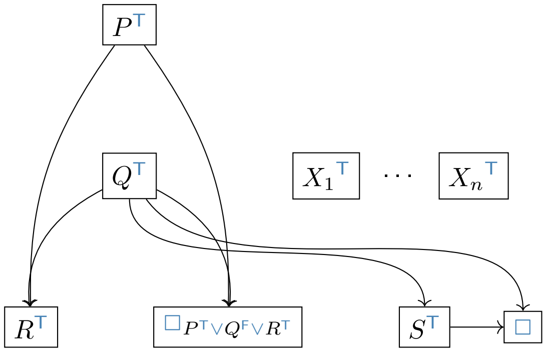


Implication Graphs: Example (Redundance2)

▷ **Example A.2.7.** Continuing from ??:

$$\begin{array}{lcl} \Delta & := & P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}} \,;\, P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} \,;\, P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{T}} \,;\, P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{F}} \\ \Theta & := & {X_1}^{\mathsf{T}} \vee \ldots \vee {X_n}^{\mathsf{T}} \,;\, {X_1}^{\mathsf{F}} \vee \ldots \vee {X_n}^{\mathsf{F}} \end{array}$$

DPLL on $\Delta \,;\, \Theta \,;\, \Phi$ with $\Phi := Q^{\mathsf{F}} \vee S^{\mathsf{T}} \,;\, Q^{\mathsf{F}} \vee S^{\mathsf{F}}$
Choice literals: P^{T}, $({X_1}^{\mathsf{T}}), \ldots, ({X_n}^{\mathsf{T}})$, Q^{T}. Implied literals:

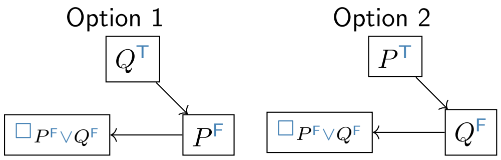


Implication Graphs: A Remark

▷ The implication graph is *not* uniquely determined by the Choice literals.

▷ It depends on "ordering decisions" during UP: Which unit clause is picked first.

▷ **Example A.2.8.** $\Delta = P^{\mathsf{F}} \vee Q^{\mathsf{F}} \,;\, Q^{\mathsf{T}} \,;\, P^{\mathsf{T}}$


Option 1
Q^{T}
$\Box_{P^{\mathsf{F}} \vee Q^{\mathsf{F}}}$
P^{F}
Option 2
P^{T}
$\Box_{P^{\mathsf{F}} \vee Q^{\mathsf{F}}}$
Q^{F}

Conflict Graphs

▷ A conflict graph captures "what went wrong" in a failed node.

▷ **Definition A.2.9 (Conflict Graph).** Let Δ be a clause set, and let G_β^{impl} be the implication graph for some search branch β of DPLL on Δ. A subgraph C of G_β^{impl} is a conflict graph if:

(i) C contains exactly one conflict vertex $\Box_C$.

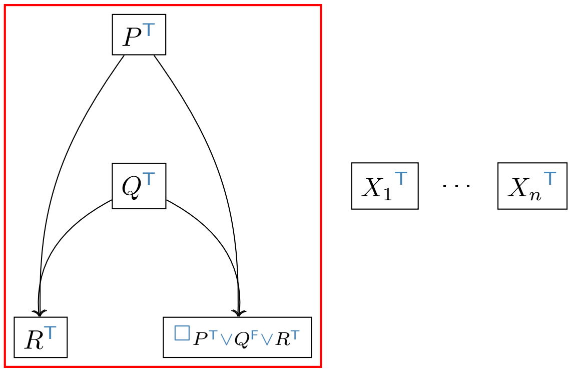
(ii) If l' is a vertex in C, then all parents of l', i.e. vertices $\overline{l_i}$ with a I edge $(\overline{l_i},l')$, are vertices in C as well.

(iii) All vertices in C have a path to $\square_C$.

▷ Conflict graph $\widehat{=}$ Starting at a conflict vertex, backchain through the implication graph until reaching choice literals.

Conflict-Graphs: Example (Redundance1)

▷ **Example A.2.10.** Continuing from ??: $\Delta := P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{F}}$
DPLL on $\Delta ; \Theta$ with $\Theta := {X_1}^{\mathsf{T}} \vee \ldots \vee {X_{100}}^{\mathsf{T}} ; {X_1}^{\mathsf{F}} \vee \ldots \vee {X_{100}}^{\mathsf{F}}$
Choice literals: P^{T}, $({X_1}^{\mathsf{T}}), \ldots, ({X_{100}}^{\mathsf{T}})$, Q^{T}. Implied literals: R^{T}.

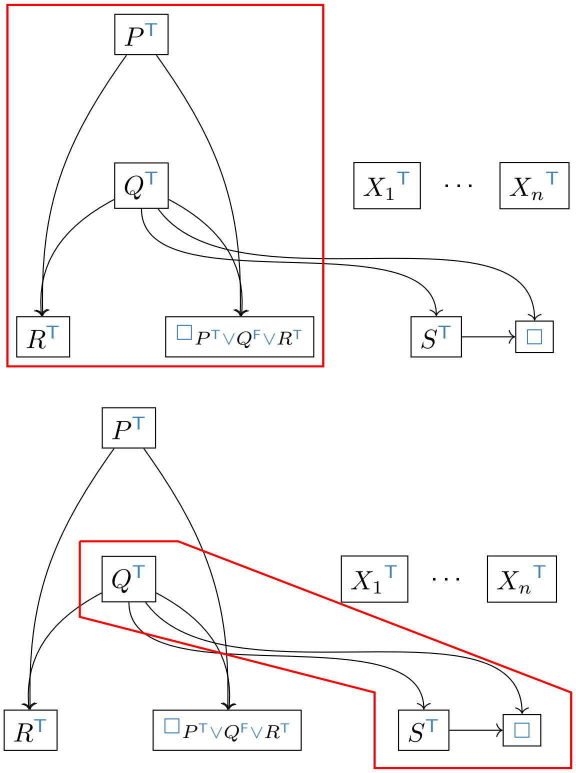


Conflict Graphs: Example (Redundance2)

▷ **Example A.2.11.** Continuing from ?? and ??:

$$\begin{aligned}\Delta &:= P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{F}} \\ \Theta &:= {X_1}^{\mathsf{T}} \vee \ldots \vee {X_n}^{\mathsf{T}} ; {X_1}^{\mathsf{F}} \vee \ldots \vee {X_n}^{\mathsf{F}}\end{aligned}$$

DPLL on $\Delta ; \Theta ; \Phi$ with $\Phi := Q^{\mathsf{F}} \vee S^{\mathsf{T}} ; Q^{\mathsf{F}} \vee S^{\mathsf{F}}$
Choice literals: P^{T}, $({X_1}^{\mathsf{T}}), \ldots, ({X_n}^{\mathsf{T}})$, Q^{T}. Implied literals: R^{T}.



A.2.2 Clause Learning

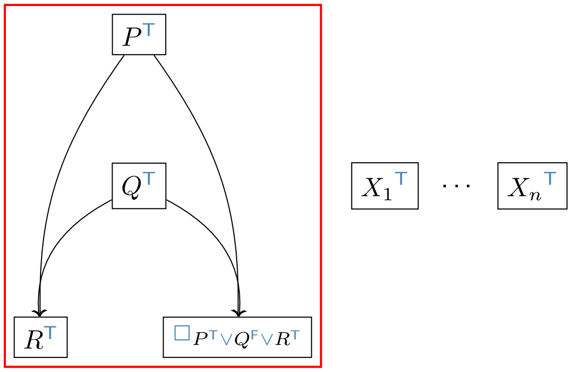
Clause Learning

- ▷ **Observation:** Conflict graphs encode the entailment relation.
- ▷ **Definition A.2.12.** Let Δ be a clause set, C be a conflict graph at some time point during a run of DPLL on Δ, and L be the choice literals in C, then we call $c := \bigvee_{l \in L} \overline{l}$ the learned clause for C.
- ▷ **Theorem A.2.13.** *Let Δ, C, and c as in ??, then $\Delta \vDash c$.*
- ▷ **Idea:** We can add learned clauses to DPLL derivations at any time without losing soundness. (maybe this helps, if we have a good notion of learned clauses)
- ▷ **Definition A.2.14.** Clause learning is the process of adding learned clauses to DPLL clause sets at specific points. (details coming up)

Clause Learning: Example (Redundance1)

- ▷ **Example A.2.15.** Continuing from ??:

$\Delta := P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{F}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} \vee R^{\mathsf{F}}$
DPLL on $\Delta ; \Theta$ with $\Theta := X_1^{\mathsf{T}} \vee \ldots \vee X_n^{\mathsf{T}} ; X_1^{\mathsf{F}} \vee \ldots \vee X_n^{\mathsf{F}}$
Choice literals: P^{T}, $(X_1^{\mathsf{T}}), \ldots, (X_n^{\mathsf{T}})$, Q^{T}. Implied literals: R^{T}.



Learned clause: $P^{\mathsf{F}} \vee Q^{\mathsf{F}}$

The Effect of Learned Clauses (in Redundance1)

▷ What happens after we learned a new clause C?

1. We add C into Δ. e.g. $C = P^{\mathsf{F}} \vee Q^{\mathsf{F}}$.
2. We retract the last choice l'. e.g. the choice $l' = Q$.

▷ **Observation:** Let C be a learned clause, i.e. $C = \bigvee_{l \in L} \overline{l}$, where L is the set of conflict literals in a conflict graph G.

Before we learn C, G must contain the most recent choice l': otherwise, the conflict would have occured earlier on.

So $C = l_1^{\mathsf{T}} \vee \ldots \vee l_k^{\mathsf{T}} \vee \overline{l'}$ where $l_1, \ldots, l_k$ are earlier choices.

▷ **Example A.2.16.** $l_1 = P$, $C = P^{\mathsf{F}} \vee Q^{\mathsf{F}}$, $l' = Q$.

▷ **Observation:** Given the earlier choices $l_1, \ldots, l_k$, after we learned the new clause $C = \overline{l_1} \vee \ldots \vee \overline{l_k} \vee \overline{l'}$, the value of $\overline{l'}$ is now set by UP!

▷ So we can continue:

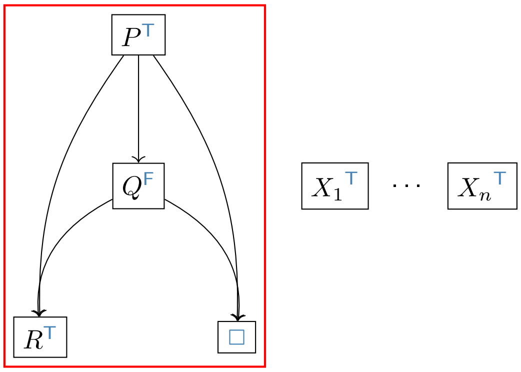
3. We set the opposite choice $\overline{l'}$ as an implied literal.
   e.g. Q^{F} as an implied literal.
4. We run UP and analyze conflicts.
   Learned clause: earlier choices only! e.g. $C = P^{\mathsf{F}}$, see next slide.

The Effect of Learned Clauses: Example (Redundance1)

▷ **Example A.2.17.** Continuing from ??:

$$\begin{aligned}\Delta &:= P^F \vee Q^F \vee R^T \,;\, P^F \vee Q^F \vee R^F \,;\, P^F \vee Q^T \vee R^T \,;\, P^F \vee Q^T \vee R^F\\ \Theta &:= {X_1}^T \vee \ldots \vee {X_{100}}^T \,;\, {X_1}^F \vee \ldots \vee {X_{100}}^F\end{aligned}$$

DPLL on $\Delta \,;\, \Theta \,;\, \Phi$ with $\Phi := P^F \vee Q^F$
Choice literals: P^T, $({X_1}^T), \ldots, ({X_{100}}^T)$, Q^T. Implied literals: Q^F, R^T.



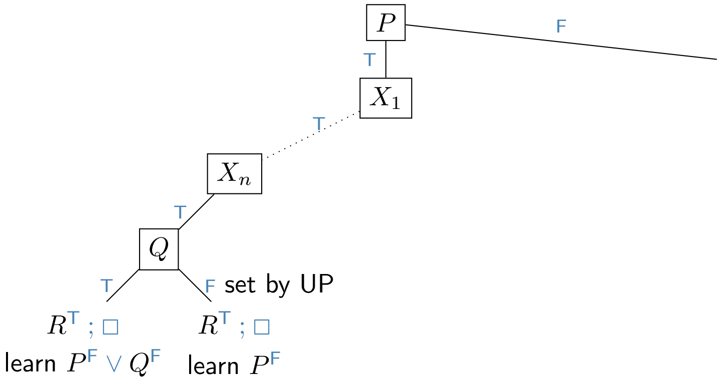
Learned clause: P^F

NOT the same Mistakes over Again: (Redundance1)

▷ **Example A.2.18.** Continuing from ??:

$$\Delta := P^F \vee Q^F \vee R^T \,;\, P^F \vee Q^F \vee R^F \,;\, P^F \vee Q^T \vee R^T \,;\, P^F \vee Q^T \vee R^F$$

DPLL on $\Delta \,;\, \Theta$ with $\Theta := {X_1}^T \vee \ldots \vee {X_n}^T \,;\, {X_1}^F \vee \ldots \vee {X_n}^F$



▷ **Note:** Here, the problem could be avoided by splitting over different variables.

▷ **Problem:** This is not so in general! (see next slide)

Clause Learning vs. Resolution

▷ **Recall:** DPLL $\widehat{=}$ tree resolution (from slide 400)

1. **in particular**: each derived clause C (not in Δ) is derived anew every time it is used.
2. **Problem**: there are Δ whose shortest tree resolution proof is exponentially longer than their shortest (general) resolution proof.

▷ **Good News:** This is no longer the case with clause learning!

1. We add each learned clause C to Δ, can use it as often as we like.
2. Clause learning renders DPLL equivalent to full resolution [BKS04; PD09]. (Inhowfar exactly this is the case was an open question for ca. 10 years, so it's not as easy as I made it look here ...)

▷ **In particular:** Selecting different variables/values to split on can *provably* not bring DPLL up to the power of DPLL+Clause Learning. (cf. slide 577, and previous slide)

"DPLL + Clause Learning"?

▷ **Disclaimer:** We have only seen *how to learn a clause from a conflict*.

▷ We will *not* cover how the overall DPLL algorithm changes, given this learning. Slides 575 – 577 are merely meant to give a *rough intuition* on "backjumping".

▷ **Definition A.2.19 (Just for the record).** (not exam or exercises relevant)

- One *could* run "DPLL + Clause Learning" by always backtracking to the maximal-level choice variable contained in the learned clause.
- The actual algorithm is called Conflict Directed Clause Learning (CDCL), and differs from DPLL more radically:

```
let $L := 0$; $I := \emptyset$
repeat
  execute UP
  if a conflict was reached then /∗ learned clause $C = \overline{l_1} \vee \ldots \vee \overline{l_k} \vee \overline{l'}$ ∗/
    if $L = 0$ then return UNSAT
    $L := \max_{i=1}^{k}$ level($l_i$); erase $I$ below $L$
    add $C$ into $\Delta$; add $\overline{l'}$ to $I$ at level $L$
  else
    if $I$ is a total interpretation then return $I$
    choose a new decision literal $l$; add $l$ to $I$ at level $L$
    $L := L + 1$
```

Remarks

- **Which clause(s) to learn?:**
  - While we only select choice literals, much more can be done.
  - For any cut through the conflict graph, with Choice literals on the "left hand" side of the cut and the conflict literals on the right-hand side, the literals on the left border of the cut yield a learnable clause.
  - Must take care to *not learn too many clauses* ...
- **Origins of clause learning:**
  - Clause learning originates from "explanation-based (no-good) learning" developed in the CSP community.
  - The distinguishing feature here is that the "no-good" is a clause:
    - The exact same type of constraint as the rest of Δ.

FAU Michael Kohlhase: Artificial Intelligence 1 580 2025-02-06

A.2.3 Phase Transitions: Where the *Really* Hard Problems Are

A Video Nugget covering this subsection can be found at `https://fau.tv/clip/id/25088`.

Where Are the Hard Problems?

- SAT is **NP** hard. Worst case for DPLL is $\mathcal{O}(2^n)$, with n propositions.
- Imagine I gave you as homework to make a formula family $\{\varphi\}$ where DPLL running time necessarily is in the order of $\mathcal{O}(2^n)$.
  - I promise you're not gonna find this easy ... (although it is of course possible: e.g., the "Pigeon Hole Problem").
- People noticed by the early 90s that, in practice, the DPLL worst case does not tend to happen.
- Modern SAT solvers successfully tackle practical instances where $n > 1.000.000$.

FAU Michael Kohlhase: Artificial Intelligence 1 581 2025-02-06

Where Are the Hard Problems?

- **So, what's the problem:** Science is about *understanding the world*.
  - Are "hard cases" just pathological outliers?
  - Can we say something about the *typical case*?
- **Difficulty 1:** What is the "typical case" in applications? E.g., what is the "average" hardware verification instance?
  - Consider precisely defined random distributions instead.

▷ **Difficulty 2:** Search trees get very complex, and are difficult to analyze mathematically, even in trivial examples. Never mind examples of practical relevance . . .

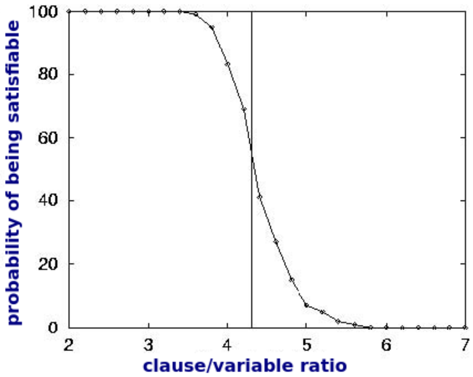
▹ The most successful works are empirical. (Interesting theory is mainly concerned with *hand-crafted* formulas, like the Pigeon Hole Problem.)

Phase Transitions in SAT [MSL92]

▷ **Fixed clause length model:** Fix clause length k; n variables. Generate m clauses, by uniformly choosing k variables P for each clause C, and for each variable P deciding uniformly whether to add P or P^{F} into C.

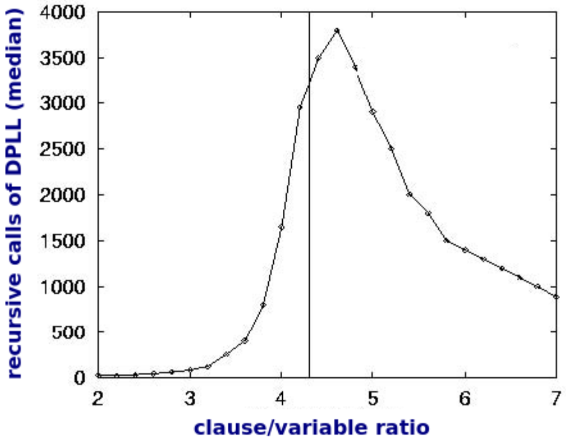
▷ **Order parameter:** Clause/variable ratio $\frac{m}{n}$.

▷ **Phase transition:** (Fixing $k = 3$, $n = 50$)



Does DPLL Care?

▷ **Oh yes, it does:** Extreme running time peak at the phase transition!



Why Does DPLL Care?

▷ **Intuition:**

Under-Constrained: Satisfiability likelihood close to 1. Many solutions, first DPLL search path usually successful. ("Deep but narrow")

Over-Constrained: Satisfiability likelihood close to 0. Most DPLL search paths short, conflict reached after few applications of splitting rule. ("Broad but shallow")

Critically Constrained: At the phase transition, many *almost-successful* DPLL search paths. ("Close, but no cigar")

The Phase Transition Conjecture

▷ **Definition A.2.20.** We say that a class P of problems exhibits a phase transition, if there is an order parameter o, i.e. a structural parameter of P, so that almost all the hard problems of P cluster around a critical value c of o and c separates one region of the problem space from another, e.g. over-constrained and under-constrained regions.

▷ All $\mathbf{NP}$-complete problems exhibit at least one phase transition.

▷ [CKT91] confirmed this for Graph Coloring and Hamiltonian Circuits. Later work confirmed it for SAT (see previous slides), and for numerous other $\mathbf{NP}$-complete problems.

Why Should *We* Care?

- ▷ **Enlightenment:**
  - ▷ Phase transitions contribute to the fundamental understanding of the behavior of search, even if it's only in random distributions.
  - ▷ There are interesting theoretical connections to phase transition phenomena in physics. (See [GS05] for a short summary.)
- ▷ **Ok, but what can we use these results for?:**
  - ▷ **Benchmark design**: Choose instances from phase transition region.
    - ▷ Commonly used in competitions etc. (In SAT, random phase transition formulas are the most difficult for DPLL style searches.)
  - ▷ **Predicting solver performance**: Yes, but very limited because:
- ▷ All this works only for the particular considered *distributions of instances*! Not meaningful for any other instances.

A.3 Completeness of Calculi for First-Order Logic

We will now analyze the first-order calculi for completeness. Just as in the case of the propositional calculi, we prove a model existence theorem for the first-order model theory and then use that EdN:2 for the completeness proofs[2]. The proof of the first-order model existence theorem is completely analogous to the propositional one; indeed, apart from the model construction itself, it is just an EdN:3 extension by a treatment for the first-order quantifiers.[3]

A.3.1 Abstract Consistency and Model Existence

We will now come to an important tool in the theoretical study of reasoning calculi: the "abstract consistency"/"model existence" method. This method for analyzing calculi was developed by Jaako Hintikka, Raymond Smullyan, and Peter Andrews in 1950-1970 as an encapsulation of similar constructions that were used in completeness arguments in the decades before. The basis for this method is Smullyan's Observation [Smu63] that completeness proofs based on Hintikka sets only certain properties of consistency and that with little effort one can obtain a generalization "Smullyan's Unifying Principle".

The basic intuition for this method is the following: typically, a logical system $\mathcal{L} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ has multiple calculi, human-oriented ones like the natural deduction calculi and machine-oriented ones like the automated theorem proving calculi. All of these need to be analyzed for completeness (as a basic quality assurance measure).

A completeness proof for a calculus $\mathcal{C}$ for $\mathcal{S}$ typically comes in two parts: one analyzes $\mathcal{C}$-consistency (sets that cannot be refuted in $\mathcal{C}$), and the other construct $\mathcal{K}$-models for $\mathcal{C}$-consistent sets.

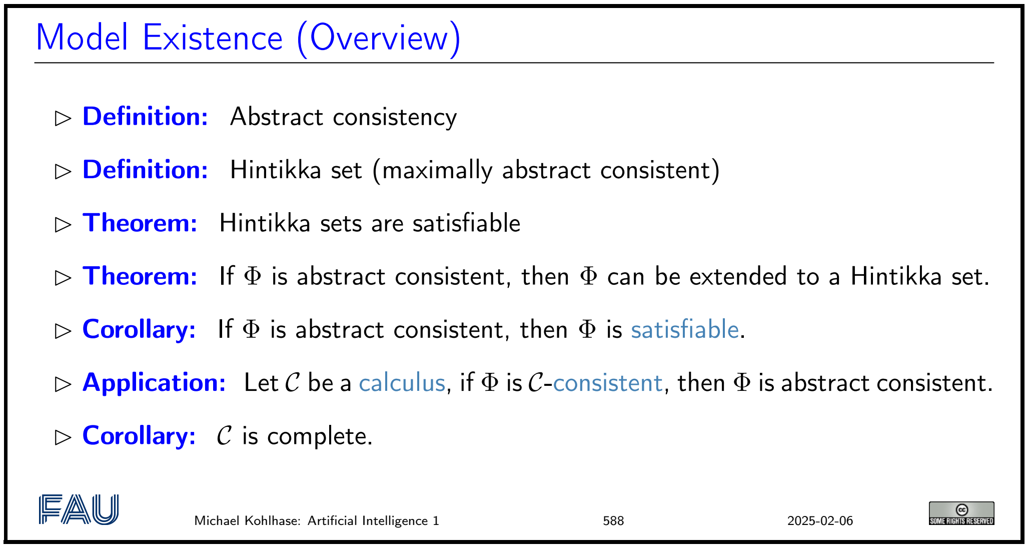
In this situtation the "abstract consistency"/"model existence" method encapsulates the model construction process into a meta-theorem: the "model existence" theorem. This provides a set of syntactic ("abstract consistency") conditions for calculi that are sufficient to construct models.

With the model existence theorem it suffices to show that $\mathcal{C}$-consistency is an abstract consistency property (a purely syntactic task that can be done by a $\mathcal{C}$-proof transformation argument)

[2] EdNote: reference the theorems

[3] EdNote: MK: what about equality?

to obtain a completeness result for $\mathcal{C}$.



Model Existence (Overview)

- ▷ **Definition:** Abstract consistency
- ▷ **Definition:** Hintikka set (maximally abstract consistent)
- ▷ **Theorem:** Hintikka sets are satisfiable
- ▷ **Theorem:** If Φ is abstract consistent, then Φ can be extended to a Hintikka set.
- ▷ **Corollary:** If Φ is abstract consistent, then Φ is satisfiable.
- ▷ **Application:** Let $\mathcal{C}$ be a calculus, if Φ is $\mathcal{C}$-consistent, then Φ is abstract consistent.
- ▷ **Corollary:** $\mathcal{C}$ is complete.

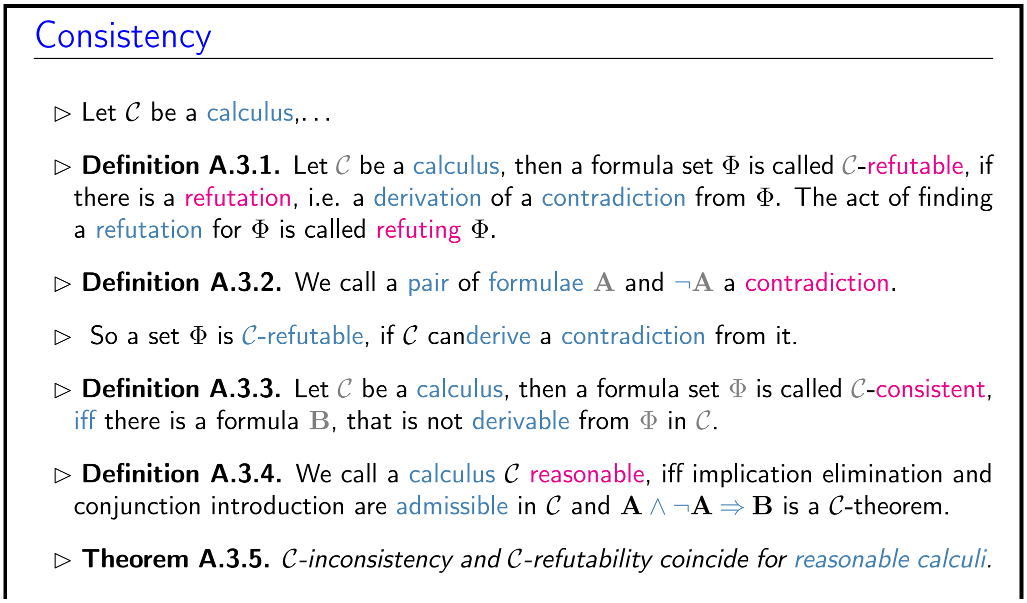
Michael Kohlhase: Artificial Intelligence 1 588 2025-02-06

The proof of the model existence theorem goes via the notion of a Hintikka set, a set of formulae with very strong syntactic closure properties, which allow to read off models. Jaako Hintikka's original idea for completeness proofs was that for every complete calculus $\mathcal{C}$ and every $\mathcal{C}$-consistent set one can induce a Hintikka set, from which a model can be constructed. This can be considered as a first model existence theorem. However, the process of obtaining a Hintikka set for a $\mathcal{C}$-consistent set Φ of sentences usually involves complicated calculus dependent constructions.

In this situation, Raymond Smullyan was able to formulate the sufficient conditions for the existence of Hintikka sets in the form of "abstract consistency properties" by isolating the calculus independent parts of the Hintikka set construction. His technique allows to reformulate Hintikka sets as maximal elements of abstract consistency classes and interpret the Hintikka set construction as a maximizing limit process.

To carry out the "model-existence"/"abstract consistency" method, we will first have to look at the notion of consistency.

Consistency and refutability are very important notions when studying the completeness for calculi; they form syntactic counterparts of satisfiability.



Consistency

- ▷ Let $\mathcal{C}$ be a calculus,. . .
- ▷ **Definition A.3.1.** Let $\mathcal{C}$ be a calculus, then a formula set Φ is called $\mathcal{C}$-refutable, if there is a refutation, i.e. a derivation of a contradiction from Φ. The act of finding a refutation for Φ is called refuting Φ.
- ▷ **Definition A.3.2.** We call a pair of formulae $\mathbf{A}$ and $\neg\mathbf{A}$ a contradiction.
- ▷ So a set Φ is $\mathcal{C}$-refutable, if $\mathcal{C}$ canderive a contradiction from it.
- ▷ **Definition A.3.3.** Let $\mathcal{C}$ be a calculus, then a formula set Φ is called $\mathcal{C}$-consistent, iff there is a formula $\mathbf{B}$, that is not derivable from Φ in $\mathcal{C}$.
- ▷ **Definition A.3.4.** We call a calculus $\mathcal{C}$ reasonable, iff implication elimination and conjunction introduction are admissible in $\mathcal{C}$ and $\mathbf{A} \wedge \neg\mathbf{A} \Rightarrow \mathbf{B}$ is a $\mathcal{C}$-theorem.
- ▷ **Theorem A.3.5.** *$\mathcal{C}$-inconsistency and $\mathcal{C}$-refutability coincide for reasonable calculi.*

It is very important to distinguish the syntactic $\mathcal{C}$-refutability and $\mathcal{C}$-consistency from satisfiability, which is a property of formulae that is at the heart of semantics. Note that the former have the calculus (a syntactic device) as a parameter, while the latter does not. In fact we should actually say $\mathcal{S}$-satisfiability, where $\langle\mathcal{L},\mathcal{K},\models\rangle$ is the current logical system.

Even the word "contradiction" has a syntactical flavor to it, it translates to "saying against each other" from its Latin root.

The notion of an "abstract consistency class" provides the a calculus-independent notion of consistency: A set Φ of sentences is considered "consistent in an abstract sense", iff it is a member of an abstract consistency class ∇.

Abstract Consistency

▷ **Definition A.3.6.** Let ∇ be a collection of sets. We call ∇ closed under subsets, iff for each $\Phi\in\nabla$, all subsets $\Psi\subseteq\Phi$ are elements of ∇.

▷ **Notation:** We will use $\Phi*\mathbf{A}$ for $\Phi\cup\{\mathbf{A}\}$.

▷ **Definition A.3.7.** A family $\nabla\subseteq wff_o(\Sigma_\iota,\mathcal{V}_\iota)$ of sets of formulae is called a (first-order) abstract consistency class, iff it is closed under subsets, and for each $\Phi\in\nabla$

∇_c) $\mathbf{A}\notin\Phi$ or $\neg\mathbf{A}\notin\Phi$ for atomic $\mathbf{A}\in wff_o(\Sigma_\iota,\mathcal{V}_\iota)$.

$\nabla_\neg$) $\neg\neg\mathbf{A}\in\Phi$ implies $\Phi*\mathbf{A}\in\nabla$

$\nabla_\wedge$) $\mathbf{A}\wedge\mathbf{B}\in\Phi$ implies $\Phi\cup\{\mathbf{A},\mathbf{B}\}\in\nabla$

$\nabla_\vee$) $\neg(\mathbf{A}\wedge\mathbf{B})\in\Phi$ implies $\Phi*\neg\mathbf{A}\in\nabla$ or $\Phi*\neg\mathbf{B}\in\nabla$

$\nabla_\forall$) If $\forall X.\mathbf{A}\in\Phi$, then $\Phi*([\mathbf{B}/X](\mathbf{A}))\in\nabla$ for each closed term $\mathbf{B}$.

$\nabla_\exists$) If $\neg(\forall X.\mathbf{A})\in\Phi$ and c is an individual constant that does not occur in Φ, then $\Phi*\neg([c/X](\mathbf{A}))\in\nabla$

The conditions are very natural: Take for instance ∇_c, it would be foolish to call a set Φ of sentences "consistent under a complete calculus", if it contains an elementary contradiction. The next condition $\nabla_\neg$ says that if a set Φ that contains a sentence $\neg\neg\mathbf{A}$ is "consistent", then we should be able to extend it by $\mathbf{A}$ without losing this property; in other words, a complete calculus should be able to recognize $\mathbf{A}$ and $\neg\neg\mathbf{A}$ to be equivalent. We will carry out the proof here, since it gives us practice in dealing with the abstract consistency properties.

The main result here is that abstract consistency classes can be extended to compact ones. The proof is quite tedious, but relatively straightforward. It allows us to assume that all abstract consistency classes are compact in the first place (otherwise we pass to the compact extension).

Actually we are after abstract consistency classes that have an even stronger property than just being closed under subsets. This will allow us to carry out a limit construction in the Hintikka set extension argument later.

Compact Collections

▷ **Definition A.3.8.** We call a collection ∇ of sets compact, iff for any set Φ we have

$\Phi\in\nabla$, iff $\Psi\in\nabla$ for every finite subset Ψ of Φ.

▷ **Lemma A.3.9.** *If ∇ is compact, then ∇ is closed under subsets.*

▷ *Proof:*

1. Suppose $S \subseteq T$ and $T \in \nabla$.
2. Every finite subset A of S is a finite subset of T.
3. As ∇ is compact, we know that $A \in \nabla$.
4. Thus $S \in \nabla$.

The property of being closed under subsets is a “downwards-oriented” property: We go from large sets to small sets, compactness (the interesting direction anyways) is also an “upwards-oriented” property. We can go from small (finite) sets to large (infinite) sets. The main application for the compactness condition will be to show that infinite sets of formulae are in a collection ∇ by testing all their finite subsets (which is much simpler).

Compact Abstract Consistency Classes

▷ **Lemma A.3.10.** *Any first-order abstract consistency class can be extended to a compact one.*

▷ *Proof:*

1. We choose $\nabla' := \{\Phi \subseteq \mathit{cwff}_o(\Sigma_\iota) \mid$ every finite subset of $\Phi$is in $\nabla\}$.
2. Now suppose that $\Phi \in \nabla$. ∇ is closed under subsets, so every finite subset of Φ is in ∇ and thus $\Phi \in \nabla'$. Hence $\nabla \subseteq \nabla'$.
3. Let us now show that each ∇ is compact.'
   - 3.1. Suppose $\Phi \in \nabla'$ and Ψ is an arbitrary finite subset of Φ.
   - 3.2. By definition of ∇' all finite subsets of Φ are in ∇ and therefore $\Psi \in \nabla'$.
   - 3.3. Thus all finite subsets of Φ are in ∇' whenever Φ is in ∇'.
   - 3.4. On the other hand, suppose all finite subsets of Φ are in ∇'.
   - 3.5. Then by the definition of ∇' the finite subsets of Φ are also in ∇, so $\Phi \in \nabla'$. Thus ∇' is compact.
4. Note that ∇' is closed under subsets by the Lemma above.
5. Next we show that if ∇ satisfies ∇_*, then ∇ satisfies ∇_*.'
   - 5.1. To show ∇_c, let $\Phi \in \nabla'$ and suppose there is an atom $\mathbf{A}$, such that $\{\mathbf{A}, \neg\mathbf{A}\} \subseteq \Phi$. Then $\{\mathbf{A}, \neg\mathbf{A}\} \in \nabla$ contradicting ∇_c.
   - 5.2. To show $\nabla_\neg$, let $\Phi \in \nabla'$ and $\neg\neg\mathbf{A} \in \Phi$, then $\Phi{*}\mathbf{A} \in \nabla'$.
     - 5.2.1. Let Ψ be any finite subset of $\Phi{*}\mathbf{A}$, and $\Theta := (\Psi \backslash \{\mathbf{A}\}){*}\neg\neg\mathbf{A}$.
     - 5.2.2. Θ is a finite subset of Φ, so $\Theta \in \nabla$.
     - 5.2.3. Since ∇ is an abstract consistency class and $\neg\neg\mathbf{A} \in \Theta$, we get $\Theta{*}\mathbf{A} \in \nabla$ by $\nabla_\neg$.
     - 5.2.4. We know that $\Psi \subseteq \Theta{*}\mathbf{A}$ and ∇ is closed under subsets, so $\Psi \in \nabla$.
     - 5.2.5. Thus every finite subset Ψ of $\Phi{*}\mathbf{A}$ is in ∇ and therefore by definition $\Phi{*}\mathbf{A} \in \nabla'$.
   - 5.3. the other cases are analogous to $\nabla_\neg$.

Hintikka sets are sets of sentences with very strong analytic closure conditions. These are motivated as maximally consistent sets i.e. sets that already contain everything that can be consistently added to them.

∇-Hintikka Set

- **Definition A.3.11.** Let ∇ be an abstract consistency class, then we call a set $\mathcal{H} \in \nabla$ a ∇ Hintikka Set, iff $\mathcal{H}$ is maximal in ∇, i.e. for all $\mathbf{A}$ with $\mathcal{H}*\mathbf{A} \in \nabla$ we already have $\mathbf{A} \in \mathcal{H}$.

- **Theorem A.3.12 (Hintikka Properties).** *Let ∇ be an abstract consistency class and $\mathcal{H}$ be a ∇-Hintikka set, then*

  $\mathcal{H}_c$) *For all $\mathbf{A} \in wff_o(\Sigma_\iota, \mathcal{V}_\iota)$ we have $\mathbf{A} \notin \mathcal{H}$ or $\neg\mathbf{A} \notin \mathcal{H}$.*

  $\mathcal{H}_\neg$) *If $\neg\neg\mathbf{A} \in \mathcal{H}$ then $\mathbf{A} \in \mathcal{H}$.*

  $\mathcal{H}_\wedge$) *If $\mathbf{A} \wedge \mathbf{B} \in \mathcal{H}$ then $\mathbf{A}, \mathbf{B} \in \mathcal{H}$.*

  $\mathcal{H}_\vee$) *If $\neg(\mathbf{A} \wedge \mathbf{B}) \in \mathcal{H}$ then $\neg\mathbf{A} \in \mathcal{H}$ or $\neg\mathbf{B} \in \mathcal{H}$.*

  $\mathcal{H}_\forall$) *If $\forall X.\mathbf{A} \in \mathcal{H}$, then $[\mathbf{B}/X](\mathbf{A}) \in \mathcal{H}$ for each closed term $\mathbf{B}$.*

  $\mathcal{H}_\exists$) *If $\neg(\forall X.\mathbf{A}) \in \mathcal{H}$ then $\neg([\mathbf{B}/X](\mathbf{A})) \in \mathcal{H}$ for some term closed term $\mathbf{B}$.*

- *Proof:*

  We prove the properties in turn $\mathcal{H}_c$ goes by induction on the structure of $\mathbf{A}$

  1. $\mathbf{A}$ atomic
     1.1. Then $\mathbf{A} \notin \mathcal{H}$ or $\neg\mathbf{A} \notin \mathcal{H}$ by ∇_c.
  2. $\mathbf{A} = \neg\mathbf{B}$
     2.1. Let us assume that $\neg\mathbf{B} \in \mathcal{H}$ and $\neg\neg\mathbf{B} \in \mathcal{H}$,
     2.2. then $\mathcal{H}*\mathbf{B} \in \nabla$ by $\nabla_\neg$, and therefore $\mathbf{B} \in \mathcal{H}$ by maximality.
     2.3. So $\{\mathbf{B}, \neg\mathbf{B}\} \subseteq \mathcal{H}$, which contradicts the induction hypothesis.
  3. $\mathbf{A} = \mathbf{B} \vee \mathbf{C}$ *similar to the previous case*
  4. We prove $\mathcal{H}_\neg$ by maximality of $\mathcal{H}$ in ∇.
     4.1. If $\neg\neg\mathbf{A} \in \mathcal{H}$, then $\mathcal{H}*\mathbf{A} \in \nabla$ by $\nabla_\neg$.
     4.2. The maximality of $\mathcal{H}$ now gives us that $\mathbf{A} \in \mathcal{H}$.
  5. The other $\mathcal{H}_*$ are similar

The following theorem is one of the main results in the “abstract consistency”/“model existence” method. For any abstract consistent set Φ it allows us to construct a Hintikka set $\mathcal{H}$ with $\Phi \in \mathcal{H}$.

Extension Theorem

- **Theorem A.3.13.** *If ∇ is an abstract consistency class and $\Phi \in \nabla$ finite, then there is a ∇-Hintikka set $\mathcal{H}$ with $\Phi \subseteq \mathcal{H}$.*

- *Proof:*

  1. Wlog. assume that ∇ compact (else use compact extension)
  2. Choose an enumeration $\mathbf{A}_1, \ldots$ of $cwff_o(\Sigma_\iota)$ and $c_1, \ldots$ of Σ_0^{sk}.
  3. and construct a sequence of sets $\mathbf{H}_i$ with $\mathbf{H}_0 := \Phi$ and

$$\mathbf{H}_{n+1} := \begin{cases} \mathbf{H}_n & \text{if } \mathbf{H}_n*\mathbf{A}_n \notin \nabla \\ \mathbf{H}_n \cup \{\mathbf{A}_n, \neg([c_n/X](\mathbf{B}))\} & \text{if } \mathbf{H}_n*\mathbf{A}_n \in \nabla \text{ and } \mathbf{A}_n = \neg(\forall X.\mathbf{B}) \\ \mathbf{H}_n*\mathbf{A}_n & \text{else} \end{cases}$$

  4. Note that all $\mathbf{H}_i \in \nabla$, choose $\mathcal{H} := \bigcup_{i\in\mathbb{N}}\mathbf{H}_i$

5. $\Psi \subseteq \mathcal{H}$ finite implies there is a $j \in \mathbb{N}$ such that $\Psi \subseteq \mathbf{H}_j$,
6. so $\Psi \in \nabla$ as ∇ closed under subsets and $\mathcal{H} \in \nabla$ as ∇ is compact.
7. Let $\mathcal{H}{*}\mathbf{B} \in \nabla$, then there is a $j \in \mathbb{N}$ with $\mathbf{B} = \mathbf{A}_j$, so that $\mathbf{B} \in \mathbf{H}_{j+1}$ and $\mathbf{H}_{j+1} \subseteq \mathcal{H}$
8. Thus $\mathcal{H}$ is ∇-maximal

Note that the construction in the proof above is non-trivial in two respects. First, the limit construction for $\mathcal{H}$ is not executed in our original abstract consistency class ∇, but in a suitably extended one to make it compact — the original would not have contained $\mathcal{H}$ in general. Second, the set $\mathcal{H}$ is not unique for Φ, but depends on the choice of the enumeration of $\mathit{cwff}_o(\Sigma_\iota)$. If we pick a different enumeration, we will end up with a different $\mathcal{H}$. Say if $\mathbf{A}$ and $\neg\mathbf{A}$ are both ∇-consistent[4] with Φ, then depending on which one is first in the enumeration $\mathcal{H}$, will contain that one; with all the consequences for subsequent choices in the construction process.

Valuations

▷ **Definition A.3.14.** A function $\nu\colon \mathit{cwff}_o(\Sigma_\iota) \to \mathcal{D}_0$ is called a (first-order) valuation, iff ν is a propositional valuation and

- ▷ $\nu(\forall X.\mathbf{A}) = \mathsf{T}$, iff $\nu([\mathbf{B}/X](\mathbf{A})) = \mathsf{T}$ for all closed terms $\mathbf{B}$.

▷ **Lemma A.3.15.** *If $\varphi\colon \mathcal{V}_\iota \to U$ is a variable assignment, then $\mathcal{I}_\varphi\colon \mathit{cwff}_o(\Sigma_\iota) \to \mathcal{D}_0$ is a valuation.*

▷ *Proof sketch:* Immediate from the definitions

Thus a valuation is a weaker notion of evaluation in first-order logic; the other direction is also true, even though the proof of this result is much more involved: The existence of a first-order valuation that makes a set of sentences true entails the existence of a model that satisfies it.[5]

Valuation and Satisfiability

▷ **Lemma A.3.16.** *If $\nu\colon \mathit{cwff}_o(\Sigma_\iota) \to \mathcal{D}_0$ is a valuation and $\Phi \subseteq \mathit{cwff}_o(\Sigma_\iota)$ with $\nu(\Phi) = \{\mathsf{T}\}$, then Φ is satisfiable.*

▷ *Proof:* We construct a model for Φ.

1. Let $\mathcal{D}_\iota := \mathit{cwff}_\iota(\Sigma_\iota)$, and
   - ▷ $\mathcal{I}(f)\colon {\mathcal{D}_\iota}^k \to \mathcal{D}_\iota\,; \langle \mathbf{A}_1, \ldots, \mathbf{A}_k \rangle \mapsto f(\mathbf{A}_1, \ldots, \mathbf{A}_k)$ for $f \in \Sigma^f$
   - ▷ $\mathcal{I}(p)\colon {\mathcal{D}_\iota}^k \to \mathcal{D}_0\,; \langle \mathbf{A}_1, \ldots, \mathbf{A}_k \rangle \mapsto \nu(p(\mathbf{A}_1, \ldots, \mathbf{A}_k))$ for $p \in \Sigma^p$.
2. Then variable assignments into $\mathcal{D}_\iota$ are ground substitutions.
3. We show $\mathcal{I}_\varphi(\mathbf{A}) = \varphi(\mathbf{A})$ for $\mathbf{A} \in \mathit{wff}_\iota(\Sigma_\iota, \mathcal{V}_\iota)$ by induction on $\mathbf{A}$:

   3.1. $\mathbf{A} = X$

   3.1.1. then $\mathcal{I}_\varphi(\mathbf{A}) = \varphi(X)$ by definition.

   3.2. $\mathbf{A} = f(\mathbf{A}_1, \ldots, \mathbf{A}_k)$

   3.2.1. then $\mathcal{I}_\varphi(\mathbf{A}) = \mathcal{I}(f)(\mathcal{I}_\varphi(\mathbf{A}_1), \ldots, \mathcal{I}_\varphi(\mathbf{A}_n)) = \mathcal{I}(f)(\varphi(\mathbf{A}_1), \ldots, \varphi(\mathbf{A}_n)) = f(\varphi(\mathbf{A}_1), \ldots, \varphi(\mathbf{A}_n)) = \varphi(f(\mathbf{A}_1, \ldots, \mathbf{A}_k)) = \varphi(\mathbf{A})$

[4] EdNote: introduce this above

[5] EdNote: I think that we only get a semivaluation, look it up in Andrews.

We show $\mathcal{I}_\varphi(\mathbf{A}) = \nu(\varphi(\mathbf{A}))$ for $\mathbf{A} \in wff_o(\Sigma_\iota, \mathcal{V}_\iota)$ by induction on $\mathbf{A}$.

3.3. $\mathbf{A} = p(\mathbf{A}_1, \ldots, \mathbf{A}_k)$

3.3.1. then $\mathcal{I}_\varphi(\mathbf{A}) = \mathcal{I}(p)(\mathcal{I}_\varphi(\mathbf{A}_1), \ldots, \mathcal{I}_\varphi(\mathbf{A}_n)) = \mathcal{I}(p)(\varphi(\mathbf{A}_1), \ldots, \varphi(\mathbf{A}_n)) = \nu(p(\varphi(\mathbf{A}_1), \ldots, \varphi(\mathbf{A}_n))) = \nu(\varphi(p(\mathbf{A}_1, \ldots, \mathbf{A}_k))) = \nu(\varphi(\mathbf{A}))$

3.4. $\mathbf{A} = \neg\mathbf{B}$

3.4.1. then $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$, iff $\mathcal{I}_\varphi(\mathbf{B}) = \nu(\varphi(\mathbf{B})) = \mathsf{F}$, iff $\nu(\varphi(\mathbf{A})) = \mathsf{T}$.

3.5. $\mathbf{A} = \mathbf{B} \wedge \mathbf{C}$

3.5.1. similar

3.6. $\mathbf{A} = \forall X.\mathbf{B}$

3.6.1. then $\mathcal{I}_\varphi(\mathbf{A}) = \mathsf{T}$, iff $\mathcal{I}_\psi(\mathbf{B}) = \nu(\psi(\mathbf{B})) = \mathsf{T}$, for all $\mathbf{C} \in \mathcal{D}_\iota$, where $\psi = \varphi,[\mathbf{C}/X]$. This is the case, iff $\nu(\varphi(\mathbf{A})) = \mathsf{T}$.

4. Thus $\mathcal{I}_\varphi(\mathbf{A})\nu(\varphi(\mathbf{A})) = \nu(\mathbf{A}) = \mathsf{T}$ for all $\mathbf{A} \in \Phi$.
5. Hence $\mathcal{M} \models \mathbf{A}$ for $\mathcal{M} := \langle \mathcal{D}_\iota, \mathcal{I} \rangle$.

Now, we only have to put the pieces together to obtain the model existence theorem we are after.

Model Existence

▷ **Theorem A.3.17 (Hintikka-Lemma).** *If ∇ is an abstract consistency class and $\mathcal{H}$ a ∇-Hintikka set, then $\mathcal{H}$ is satisfiable.*

▷ *Proof:*

1. we define $\nu(\mathbf{A}) := \mathsf{T}$, iff $\mathbf{A} \in \mathcal{H}$,
2. then ν is a valuation by the Hintikka set properties.
3. We have $\nu(\mathcal{H}) = \{\mathsf{T}\}$, so $\mathcal{H}$ is satisfiable.

▷ **Theorem A.3.18 (Model Existence).** *If ∇ is an abstract consistency class and $\Phi \in \nabla$, then Φ is satisfiable.*

Proof:

▷ 1. There is a ∇-Hintikka set $\mathcal{H}$ with $\Phi \subseteq \mathcal{H}$ (Extension Theorem)
2. We know that $\mathcal{H}$ is satisfiable. (Hintikka-Lemma)
3. In particular, $\Phi \subseteq \mathcal{H}$ is satisfiable.

A.3.2 A Completeness Proof for First-Order ND

With the model existence proof we have introduced in the last section, the completeness proof for first-order natural deduction is rather simple, we only have to check that ND-consistency is an abstract consistency property.

Consistency, Refutability and Abstract Consistency

▷ **Theorem A.3.19 (Non-Refutability is an Abstract Consistency Property).** $\Gamma := \{\Phi \subseteq cwff_o(\Sigma_\iota) \mid \Phi \text{ not } \mathcal{ND}^1\text{−refutable}\}$ *is an abstract consistency class.*

▷ *Proof:* We check the properties of an ACC

1. If Φ is non-refutable, then any subset is as well, so Γ is closed under subsets.

We show the abstract consistency conditions ∇_* for $\Phi \in \Gamma$.

2. ∇_c
   2.1. We have to show that $\mathbf{A} \notin \Phi$ or $\neg\mathbf{A} \notin \Phi$ for atomic $\mathbf{A} \in wff_o(\Sigma_\iota, \mathcal{V}_\iota)$.
   2.2. Equivalently, we show the contrapositive: If $\{\mathbf{A}, \neg\mathbf{A}\} \subseteq \Phi$, then $\Phi \notin \Gamma$.
   2.3. So let $\{\mathbf{A}, \neg\mathbf{A}\} \subseteq \Phi$, then Φ is $\mathcal{ND}^1$-refutable by construction.
   2.4. So $\Phi \notin \Gamma$.
3. $\nabla_\neg$ *We show the contrapositive again*
   3.1. Let $\neg\neg\mathbf{A} \in \Phi$ and $\Phi * \mathbf{A} \notin \Gamma$
   3.2. Then we have a refutation $\mathcal{D}\colon \Phi * \mathbf{A} \vdash_{\mathcal{ND}^1} F$
   3.3. By prepending an application of $\mathcal{ND}_0\neg E$ for $\neg\neg\mathbf{A}$ to $\mathcal{D}$, we obtain a refutation $\mathcal{D}\colon \Phi \vdash_{\mathcal{ND}^1} F'$.
   3.4. Thus $\Phi \notin \Gamma$.

Proof sketch: *other ∇_* similar*

This directly yields two important results that we will use for the completeness analysis.

Henkin's Theorem

- **Corollary A.3.20 (Henkin's Theorem).** *Every $\mathcal{ND}^1$-consistent set of sentences has a model.*
- *Proof:*
  1. Let Φ be a $\mathcal{ND}^1$-consistent set of sentences.
  2. The class of sets of $\mathcal{ND}^1$-consistent propositions constitute an abstract consistency class.
  3. Thus the model existence theorem guarantees a model for Φ.
- **Corollary A.3.21 (Löwenheim&Skolem Theorem).** *Satisfiable set Φ of first-order sentences has a countable model.*

  Proof sketch: The model we constructed is countable, since the set of ground terms is.

Now, the completeness result for first-order natural deduction is just a simple argument away. We also get a compactness theorem (almost) for free: logical systems with a complete calculus are always compact.

▷ Completeness and Compactness

- **Theorem A.3.22 (Completeness Theorem for $\mathcal{ND}^1$).** *If $\Phi \models \mathbf{A}$, then $\Phi \vdash_{\mathcal{ND}^1} \mathbf{A}$.*
- *Proof:* We prove the result by playing with negations.
  1. If $\mathbf{A}$ is valid in all models of Φ, then $\Phi * \neg\mathbf{A}$ has no model
  2. Thus $\Phi * \neg\mathbf{A}$ is inconsistent by (the contrapositive of) Henkins Theorem.
  3. So $\Phi \vdash_{\mathcal{ND}^1} \neg\neg\mathbf{A}$ by $\mathcal{ND}_0\neg I$ and thus $\Phi \vdash_{\mathcal{ND}^1} \mathbf{A}$ by $\mathcal{ND}_0\neg E$.
- **Theorem A.3.23 (Compactness Theorem for first-order logic).** *If $\Phi \models \mathbf{A}$, then there is already a finite set $\Psi \subseteq \Phi$ with $\Psi \models \mathbf{A}$.*

Proof: This is a direct consequence of the completeness theorem

▷ 1. We have $\Phi \models \mathbf{A}$, iff $\Phi \vdash_{\mathcal{ND}^1} \mathbf{A}$.

2. As a proof is a finite object, only a finite subset $\Psi \subseteq \Phi$ can appear as leaves in the proof.

A.3.3 Soundness and Completeness of First-Order Tableaux

The soundness of the first-order free-variable tableaux calculus can be established a simple induction over the size of the tableau.

Soundness of $\mathcal{T}_1^f$

▷ **Lemma A.3.24.** *Tableau rules transform satisfiable tableaux into satisfiable ones.*

▷ *Proof:*

we examine the tableau rules in turn

1. propositional rules *as in propositional tableaux*
2. $\mathcal{T}_1^f\exists$ *by* **??**
3. $\mathcal{T}_1^f\bot$ *by* **??** *(substitution value lemma)*
4. $\mathcal{T}_1^f\forall$

   4.1. $\mathcal{I}_\varphi(\forall X.\mathbf{A}) = \mathsf{T}$, iff $\mathcal{I}_\psi(\mathbf{A}) = \mathsf{T}$ for all $a \in \mathcal{D}_\iota$

   4.2. so in particular for some $a \in \mathcal{D}_\iota \neq \emptyset$.

▷ **Corollary A.3.25.** $\mathcal{T}_1^f$ *is correct.*

The only interesting steps are the cut rule, which can be directly handled by the substitution value lemma, and the rule for the existential quantifier, which we do in a separate lemma.

Soundness of $\mathcal{T}_1^f\exists$

▷ **Lemma A.3.26.** $\mathcal{T}_1^f\exists$ *transforms satisfiable tableaux into satisfiable ones.*

▷ *Proof:* Let $\mathcal{T}'$ be obtained by applying $\mathcal{T}_1^f\exists$ to $(\forall X.\mathbf{A})^{\mathsf{F}}$ in $\mathcal{T}$, extending it with $([f(X^1,\ldots,X^k)/X](\mathbf{A}))^{\mathsf{F}}$, where $W := \mathrm{free}(\forall X.\mathbf{A}) = \{X^1,\ldots,X^k\}$

1. Let $\mathcal{T}$ be satisfiable in $\mathcal{M} := \langle\mathcal{D},\mathcal{I}\rangle$, then $\mathcal{I}_\varphi(\forall X.\mathbf{A}) = \mathsf{F}$.

   We need to find a model $\mathcal{M}'$ that satisfies $\mathcal{T}'$ *(find interpretation for f)*
2. By definition $\mathcal{I}_{\varphi,[a/X]}(\mathbf{A}) = \mathsf{F}$ for some $a \in \mathcal{D}$ (depends on $\varphi|_W$)
3. Let $g\colon \mathcal{D}^k \to \mathcal{D}$ be defined by $g(a_1,\ldots,a_k) := a$, if $\varphi(X^i) = a_i$
4. choose $\mathcal{M} = \langle\mathcal{D},\mathcal{I}'\rangle'$ with $\mathcal{I}' := \mathcal{I},[g/f]$, then by subst. value lemma

$$\begin{aligned}\mathcal{I}'_\varphi([f(X^1,\ldots,X^k)/X](\mathbf{A})) &= \mathcal{I}'_{\varphi,[\mathcal{I}'_\varphi(f(X^1,\ldots,X^k))/X]}(\mathbf{A}) \\ &= \mathcal{I}'_{\varphi,[a/X]}(\mathbf{A}) = \mathsf{F}\end{aligned}$$

5. So $([f(X^1,\ldots,X^k)/X](\mathbf{A}))^{\mathsf{F}}$ satisfiable in $\mathcal{M}'$

This proof is paradigmatic for soundness proofs for calculi with Skolemization. We use the axiom of choice at the meta-level to choose a meaning for the Skolem constant. Armed with the Model Existence Theorem for first-order logic (**??**), the completeness of first-order tableaux is similarly straightforward. We just have to show that the collection of tableau-irrefutable sentences is an abstract consistency class, which is a simple proof-transformation exercise in all but the universal quantifier case, which we postpone to its own Lemma (**??**).

Completeness of ($\mathcal{T}_1^f$)

▷ **Theorem A.3.27.** *$\mathcal{T}_1^f$ is refutation complete.*

▷ *Proof:* We show that $\nabla := \{\Phi \,|\, \Phi^{\mathsf{T}}$ has no closed Tableau$\}$ is an abstract consistency class

1. as for propositional case.
2. by the lifting lemma below
3. Let $\mathcal{T}$ be a closed tableau for $\neg(\forall X.\mathbf{A}) \in \Phi$ and $\Phi^{\mathsf{T}} * ([c/X](\mathbf{A}))^{\mathsf{F}} \in \nabla$.

$$\begin{array}{c} \Psi^{\mathsf{T}} \\ (\forall X.\mathbf{A})^{\mathsf{F}} \\ ([c/X](\mathbf{A}))^{\mathsf{F}} \\ \mathit{Rest} \end{array} \qquad \begin{array}{c} \Psi^{\mathsf{T}} \\ (\forall X.\mathbf{A})^{\mathsf{F}} \\ ([f(X_1,\ldots,X_k)/X](\mathbf{A}))^{\mathsf{F}} \\ [f(X_1,\ldots,X_k)/c](\mathit{Rest}) \end{array}$$

So we only have to treat the case for the universal quantifier. This is what we usually call a "lifting argument", since we have to transform ("lift") a proof for a formula $\theta(\mathbf{A})$ to one for $\mathbf{A}$. In the case of tableaux we do that by an induction on the tableau refutation for $\theta(\mathbf{A})$ which creates a tableau-isomorphism to a tableau refutation for $\mathbf{A}$.

Tableau-Lifting

▷ **Theorem A.3.28.** *If $\mathcal{T}_\theta$ is a closed tableau for a set $\theta(\Phi)$ of formulae, then there is a closed tableau $\mathcal{T}$ for Φ.*

▷ *Proof:* by induction over the structure of $\mathcal{T}_\theta$ we build an isomorphic tableau $\mathcal{T}$, and a tableau-isomorphism $\omega\colon \mathcal{T} \to \mathcal{T}_\theta$, such that $\omega(\mathbf{A}) = \theta(\mathbf{A})$.

only the tableau-substitution rule is interesting.

1. Let $(\theta(\mathbf{A}_i))^{\mathsf{T}}$ and $(\theta(\mathbf{B}_i))^{\mathsf{F}}$ cut formulae in the branch Θ_θ^i of $\mathcal{T}_\theta$
2. there is a joint unifier σ of $(\theta(\mathbf{A}_1))=^?(\theta(\mathbf{B}_1)) \wedge \ldots \wedge (\theta(\mathbf{A}_n))=^?(\theta(\mathbf{B}_n))$
3. thus $\sigma \circ \theta$ is a unifier of $\mathbf{A}$ and $\mathbf{B}$
4. hence there is a most general unifier ρ of $\mathbf{A}_1=^?\mathbf{B}_1 \wedge \ldots \wedge \mathbf{A}_n=^?\mathbf{B}_n$
5. so Θ is closed.

Again, the "lifting lemma for tableaux" is paradigmatic for lifting lemmata for other refutation calculi.

A.3.4 Soundness and Completeness of First-Order Resolution

Correctness (CNF)

- ▷ **Lemma A.3.29.** *A set Φ of sentences is satisfiable, iff $CNF_1(\Phi)$ is.*
- ▷ *Proof:* propositional rules and $\forall$-rule are trivial; do the $\exists$-rule
  1. Let $(\forall X.\mathbf{A})^{\mathsf{F}}$ satisfiable in $\mathcal{M} := \langle \mathcal{D}, \mathcal{I} \rangle$ and $\text{free}(\mathbf{A}) = \{X^1, \ldots, X^n\}$
  2. $\mathcal{I}_\varphi(\forall X.\mathbf{A}) = \mathsf{F}$, so there is an $a \in \mathcal{D}$ with $\mathcal{I}_{\varphi,[a/X]}(\mathbf{A}) = \mathsf{F}$ (only depends on $\varphi|_{\text{free}(\mathbf{A})}$)
  3. let $g \colon \mathcal{D}^n \to \mathcal{D}$ be defined by $g(a_1, \ldots, a_n) := a$, iff $\varphi(X^i) = a_i$.
  4. choose $\mathcal{M}' := \langle \mathcal{D}, \mathcal{I}' \rangle$ with $\mathcal{I}(f)' := g$, then $\mathcal{I}'_\varphi([f(X^1, \ldots, X^k)/X](\mathbf{A})) = \mathsf{F}$
  5. Thus $([f(X^1, \ldots, X^k)/X](\mathbf{A}))^{\mathsf{F}}$ is satisfiable in $\mathcal{M}'$

FAU Michael Kohlhase: Artificial Intelligence 1 609 2025-02-06

Resolution (Correctness)

- ▷ **Definition A.3.30.** A clause is called satisfiable, iff $\mathcal{I}_\varphi(\mathbf{A}) = \alpha$ for one of its literals $\mathbf{A}^\alpha$.
- ▷ **Lemma A.3.31.** $\square$ *is unsatisfiable*
- ▷ **Lemma A.3.32.** *CNF transformations preserve satisfiability* (*see above*)
- ▷ **Lemma A.3.33.** *Resolution and factorization too!*

FAU Michael Kohlhase: Artificial Intelligence 1 610 2025-02-06

Completeness ($\mathcal{R}_1$)

- ▷ **Theorem A.3.34.** $\mathcal{R}_1$ *is refutation complete.*
- ▷ *Proof:* $\nabla := \{\Phi \,|\, \Phi^{\mathsf{T}}$ has no closed tableau$\}$ is an abstract consistency class
  1. as for propositional case.
  2. by the lifting lemma below
  3. Let $\mathcal{T}$ be a closed tableau for $\neg(\forall X.\mathbf{A}) \in \Phi$ and $\Phi^{\mathsf{T}} * ([c/X](\mathbf{A}))^{\mathsf{F}} \in \nabla$.
  4. $CNF_1(\Phi^{\mathsf{T}}) = CNF_1(\Psi^{\mathsf{T}}) \cup CNF_1((([f(X_1, \ldots, X_k)/X](\mathbf{A}))^{\mathsf{F}})$
  5. $([f(X_1, \ldots, X_k)/c](CNF_1(\Phi^{\mathsf{T}}))) * ([c/X](\mathbf{A}))^{\mathsf{F}} = CNF_1(\Phi^{\mathsf{T}})$
  6. so $\mathcal{R}_1 \colon CNF_1(\Phi^{\mathsf{T}}) \vdash_{\mathcal{D}'} \square$, where $\mathcal{D} = [f(X'_1, \ldots, X'_k)/c](\mathcal{D})$.

FAU Michael Kohlhase: Artificial Intelligence 1 611 2025-02-06

Clause Set Isomorphism

- ▷ **Definition A.3.35.** Let $\mathbf{B}$ and $\mathbf{C}$ be clauses, then a clause isomorphism $\omega \colon \mathbf{C} \to \mathbf{D}$ is a bijection of the literals of $\mathbf{C}$ and $\mathbf{D}$, such that $\omega(\mathbf{L})^\alpha = \mathbf{M}^\alpha$ (conserves labels)
  We call ω θ compatible, iff $\omega(\mathbf{L}^\alpha) = (\theta(\mathbf{L}))^\alpha$

- ▷ **Definition A.3.36.** Let Φ and Ψ be clause sets, then we call a bijection $\Omega\colon\Phi\to\Psi$ a clause set isomorphism, iff there is a clause isomorphism $\omega\colon\mathbf{C}\to\Omega(\mathbf{C})$ for each $\mathbf{C}\in\Phi$.
- ▷ **Lemma A.3.37.** *If $\theta(\Phi)$ is set of formulae, then there is a θ-compatible clause set isomorphism $\Omega\colon CNF_1(\Phi)\to CNF_1(\theta(\Phi))$.*
- ▷ *Proof sketch:* by induction on the CNF derivation of $CNF_1(\Phi)$.

Lifting for $\mathcal{R}_1$

- ▷ **Theorem A.3.38.** *If $\mathcal{R}_1 : (\theta(\Phi))\vdash_{\mathcal{D}_\theta}\Box$ for a set $\theta(\Phi)$ of formulae, then there is a $\mathcal{R}_1$-refutation for Φ.*
- ▷ *Proof:* by induction over $\mathcal{D}_\theta$ we construct a $\mathcal{R}_1$-derivation $\mathcal{R}_1 : \Phi\vdash_{\mathcal{D}}\mathbf{C}$ and a θ-compatible clause set isomorphism $\Omega\colon\mathcal{D}\to\mathcal{D}_\theta$

1. If $\mathcal{D}_\theta$ ends in $$\cfrac{\cfrac{\mathcal{D}'_\theta}{((\theta(\mathbf{A}))\vee(\theta(\mathbf{C})))^{\mathsf{T}}}\qquad\cfrac{\mathcal{D}''_\theta}{(\theta(\mathbf{B}))^{\mathsf{F}}\vee(\theta(\mathbf{D}))}}{(\sigma(\theta(\mathbf{C})))\vee(\sigma(\theta(\mathbf{B})))}\;res$$

   then we have (IH) clause isormorphisms $\omega'\colon\mathbf{A}^{\mathsf{T}}\vee\mathbf{C}\to(\theta(\mathbf{A}))^{\mathsf{T}}\vee(\theta(\mathbf{C}))$ and $\omega'\colon\mathbf{B}^{\mathsf{T}}\vee\mathbf{D}\to(\theta(\mathbf{B}))^{\mathsf{T}},\theta(\mathbf{D})$

2. thus $\dfrac{\mathbf{A}^{\mathsf{T}}\vee\mathbf{C}\quad\mathbf{B}^{\mathsf{F}}\vee\mathbf{D}}{(\rho(\mathbf{C}))\vee(\rho(\mathbf{B}))}\;Res$ where $\rho=\mathbf{mgu}(\mathbf{A},\mathbf{B})$(exists, as $\sigma\circ\theta$ unifier)