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

Chapter 10
Propositional Logic & Reasoning, Part I: Principles

10.1 Introduction: Inference with Structured State Representations

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.

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 $\leftsquigarrow$ too many state!

State Representations in Agents and Algorithms

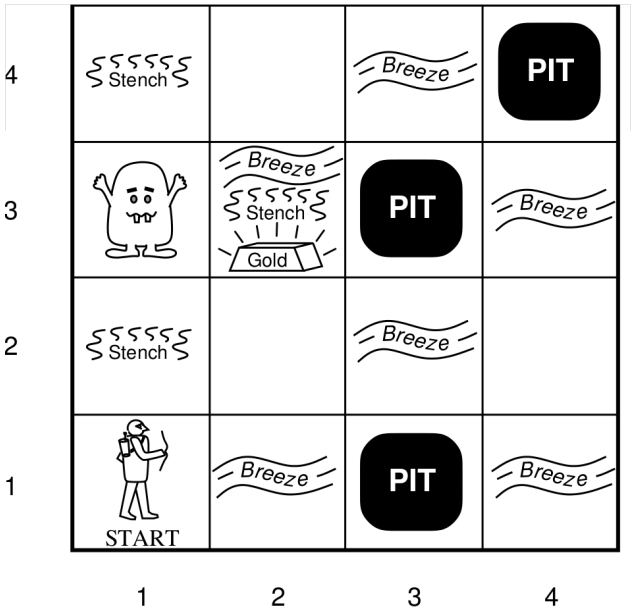
- **Recall:** We call a state representation
  - atomic, iff it has no internal structure (black box)
  - factored, iff each state is characterized by attribute and their values.
  - structured, iff the state includes representations of objects, their properties and relationships.
- **Recall:** We have 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 $\nleftrightsquigarrow$ too many state!
- **Recall:** We have used factored representations in
  - backtracking search for CSPs $\rightsquigarrow$ universally useful heuristics
  - constraint propagation: inference $\rightsquigarrow$ lifting search to the CSP description level.

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 $\leftsquigarrow$ too many state!
- **Recall:** We have used factored representations in
  - backtracking search for CSPs $\rightsquigarrow$ universally useful heuristics
  - constraint propagation: inference $\rightsquigarrow$ lifting search to the CSP description level.
- **Up Next:** Inference for structured state representations.

10.1.1 A Running Example: The Wumpus World

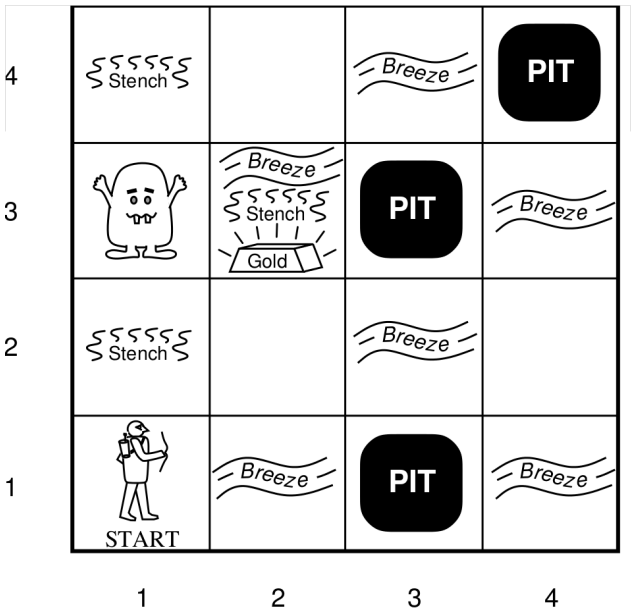
The Wumpus World



Definition 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 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 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.

The Wumpus World



Definition 1.5. 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 1.8 (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).

Reasoning in the Wumpus World

- **Example 1.9 (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

- **Note:** The agent has more knowledge than just the percepts $\rightsquigarrow$ inference!

Reasoning in the Wumpus World

- **Example 1.10 (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

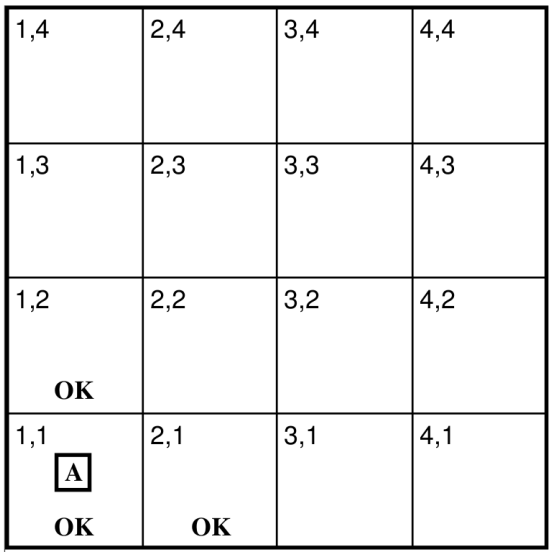
1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2 P?	3,2	4,2
1,1 V OK	2,1 A B OK	3,1 P?	4,1

(2) One step to right

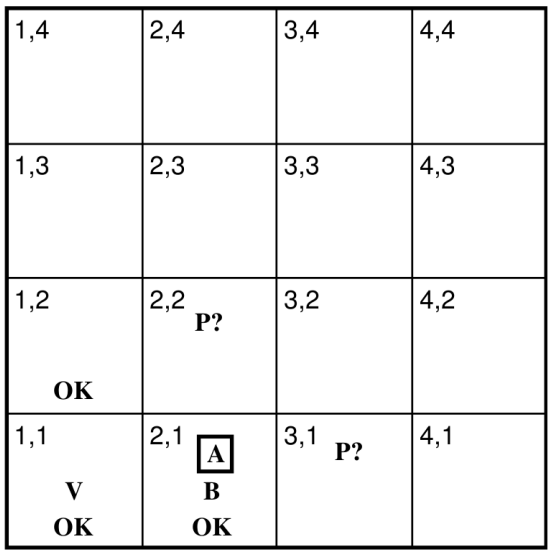
- **Note:** The agent has more knowledge than just the percepts ⇜⇝ inference!

Reasoning in the Wumpus World

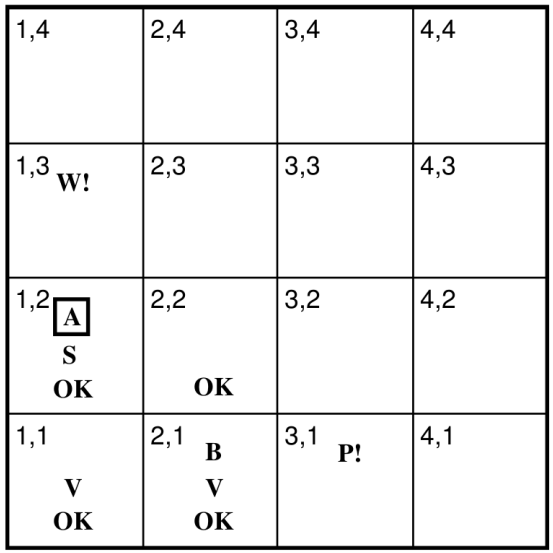
- **Example 1.11 (Reasoning in the Wumpus World).**
  As humans we mark cells with the knowledge inferred so far: **A**: agent, **V**: visited, **OK**: safe, **P**: pit, **W**: Wumpus, **B**: breeze, **S**: stench, **G**: gold.



(1) Initial state



(2) One step to right



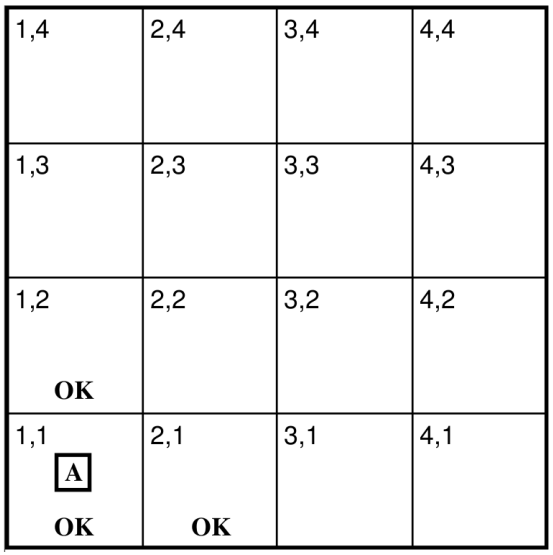
(3) Back, and up to [1,2]

- *The Wumpus is in [1,3]*! How do we know?

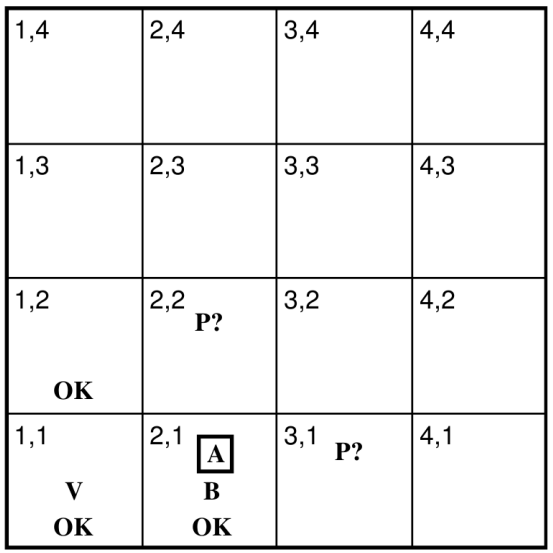
- **Note:** The agent has more knowledge than just the percepts ⇜⇝ inference!

Reasoning in the Wumpus World

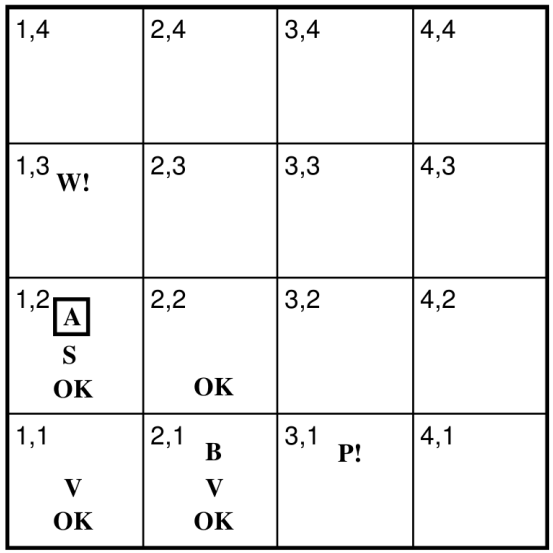
- **Example 1.12 (Reasoning in the Wumpus World).**
  As humans we mark cells with the knowledge inferred so far: **A**: agent, **V**: visited, **OK**: safe, **P**: pit, **W**: Wumpus, **B**: breeze, **S**: stench, **G**: gold.



(1) Initial state



(2) One step to right



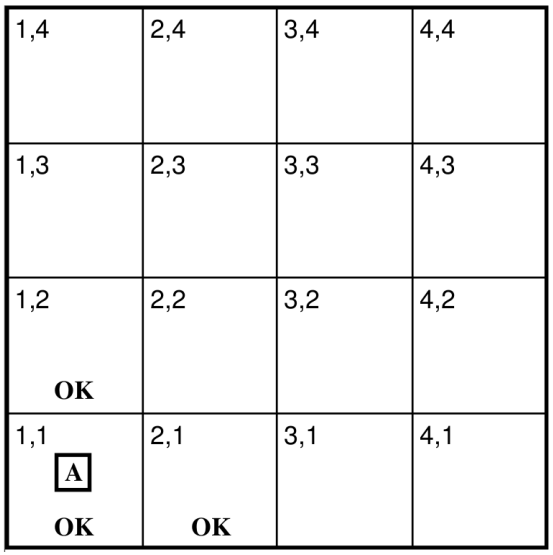
(3) Back, and up to [1,2]

  - *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?

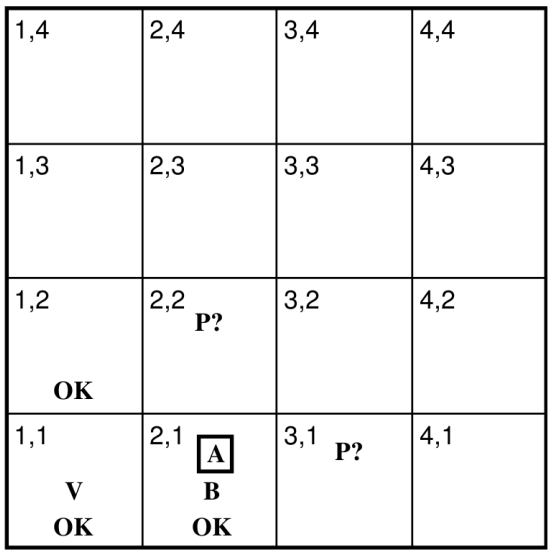
- **Note:** The agent has more knowledge than just the percepts $\rightsquigarrow$ inference!

Reasoning in the Wumpus World

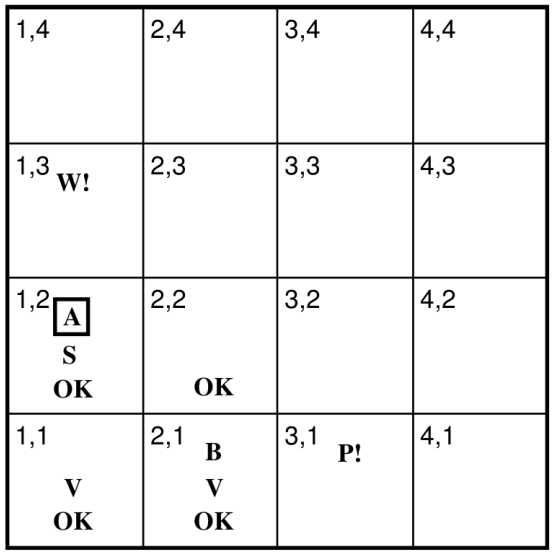
- **Example 1.13 (Reasoning in the Wumpus World).**
  As humans we mark cells with the knowledge inferred so far: **A**: agent, **V**: visited, **OK**: safe, **P**: pit, **W**: Wumpus, **B**: breeze, **S**: stench, **G**: gold.



(1) Initial state



(2) One step to right



(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 $\rightsquigarrow$ inference!

Agents that Think Rationally

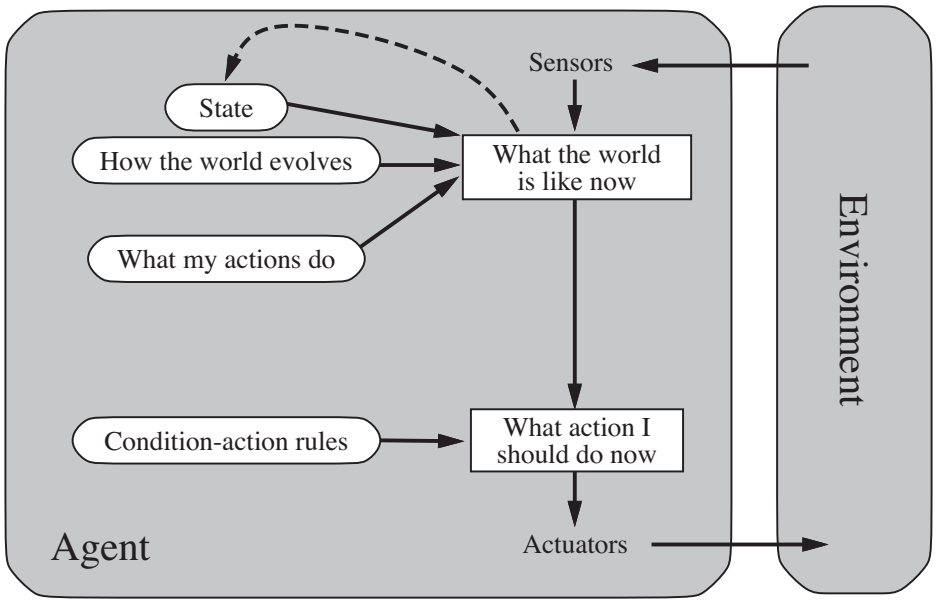
- **Problem:** But how can we build an agent that can do the necessary inferences?

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.

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 1.16.** 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.

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 1.17.** 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 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

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.

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 1.24.** Syntax: What are legal formulae A in the logic?
- **Example 1.25.** "W" and "$W \Rightarrow S$".
  ($W \widehat{=}$ *Wumpus is here*, $S \widehat{=}$ *it stinks*, $W \Rightarrow S \widehat{=}$ If W, then S)

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 1.30.** Syntax: What are legal formulae A in the logic?
- **Example 1.31.** "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 1.32.** Semantics: Which formulae 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 1.33.** For a variable assignment φ, write $\varphi \models A$ if φ is true in the Wumpus world described by φ.
- **Example 1.34.** 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!)

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 1.36.** Syntax: What are legal formulae A in the logic?
- **Example 1.37.** "W" and "$W \Rightarrow S$".
  ($W \widehat{=}$ *Wumpus is here*, $S \widehat{=}$ *it stinks*, $W \Rightarrow S \widehat{=}$ If W, then S)
- **Definition 1.38.** Semantics: Which formulae 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 1.39.** For a variable assignment φ, write $\varphi \models A$ if φ is true in the Wumpus world described by φ.
- **Example 1.40.** If $\varphi := \{W \mapsto T, S \mapsto F\}$, 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 1.41.** 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!

Logic: Basic Concepts (Reasoning about Knowledge)

- **Definition 1.42.** Entailment: Which B follow from A, written $A \models B$, meaning that, for all φ with $\varphi \models A$, we have $\varphi \models B$? 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

Logic: Basic Concepts (Reasoning about Knowledge)

- **Definition 1.47.** Entailment: Which B follow from A, written $A \models B$, meaning that, for all φ with $\varphi \models A$, we have $\varphi \models B$? 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 1.48.** Deduction: Which formulas B can be derived from A using a set $\mathcal{C}$ of inference rules (a calculus), written $A \vdash_{\mathcal{C}} B$?
- **Example 1.49.** If $\mathcal{C}$ contains $\dfrac{A \quad A \Rightarrow B}{B}$ then $P, P \Rightarrow Q \vdash_{\mathcal{C}} Q$

Logic: Basic Concepts (Reasoning about Knowledge)

- **Definition 1.52.** Entailment: Which B follow from A, written $A \models B$, meaning that, for all φ with $\varphi \models A$, we have $\varphi \models B$? 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 1.53.** Deduction: Which formulas B can be derived from A using a set $\mathcal{C}$ of inference rules (a calculus), written $A \vdash_{\mathcal{C}} B$?
- **Example 1.54.** If $\mathcal{C}$ contains $\dfrac{A \quad A \Rightarrow B}{B}$ 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.

Logic: Basic Concepts (Reasoning about Knowledge)

- **Definition 1.57.** Entailment: Which B follow from A, written $A \models B$, meaning that, for all φ with $\varphi \models A$, we have $\varphi \models B$? 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 1.58.** Deduction: Which formulas B can be derived from A using a set $\mathcal{C}$ of inference rules (a calculus), written $A \vdash_{\mathcal{C}} B$?
- **Example 1.59.** If $\mathcal{C}$ contains $\dfrac{A \quad A \Rightarrow B}{B}$ 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 1.60.** Soundness: whenever $A \vdash_{\mathcal{C}} B$, we also have $A \models B$.
- **Definition 1.61.** Completeness: whenever $A \models B$, we also have $A \vdash_{\mathcal{C}} B$.

General Problem Solving using Logic

- **Idea:** Any problem that can be formulated as reasoning about logic. $\rightsquigarrow$ 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.

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.

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.

10.2 Propositional Logic (Syntax/Semantics)

Propositional Logic (Syntax)

- **Definition 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 $wff_0(\mathcal{V}_0)$ of well-formed propositional formula (wffs) as
  - propositional variables,
  - the logical constants T and F,
  - negations $\neg$A,
  - conjunctions A $\wedge$ B(A and B are called conjuncts),
  - disjunctions A $\vee$ B (A and B are called disjuncts),
  - implications A $\Rightarrow$ B, and
  - equivalences (or biimplication). A $\Leftrightarrow$ B,

  where A, B $\in wff_0(\mathcal{V}_0)$ themselves.
- **Example 2.2.** $P \wedge Q, P \vee Q, \neg P \vee Q \Leftrightarrow P \Rightarrow Q \in wff_0(\mathcal{V}_0)$
- **Definition 2.3.** Propositional formulae without connectives are called atomic (or an atom) and complex otherwise.

Propositional Logic Grammar Overview

- **Grammar for Propositional Logic:**

propositional variables	X	$::=$	$\mathcal{V}_0 = \{P, Q, R, \ldots, \ldots\}$	variables
propositional formulae	A	$::=$	X	variable
		$\mid$	$T \mid F$	truth values
		$\mid$	$\neg \mathsf{A}$	negation
		$\mid$	$\mathsf{A}_1 \wedge \mathsf{A}_2$	conjunction
		$\mid$	$\mathsf{A}_1 \vee \mathsf{A}_2$	disjunction
		$\mid$	$\mathsf{A}_1 \Rightarrow \mathsf{A}_2$	implication
		$\mid$	$\mathsf{A}_1 \Leftrightarrow \mathsf{A}_2$	equivalence

Alternative Notations for Connectives

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

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)

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 2.5.** 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.
  - $\mathcal{I}(\neg) : \mathcal{D}_o \to \mathcal{D}_o; \mathsf{T} \mapsto \mathsf{F}, \mathsf{F} \mapsto \mathsf{T}$
  - $\mathcal{I}(\wedge) : \mathcal{D}_o \times \mathcal{D}_o \to \mathcal{D}_o; \langle \alpha, \beta \rangle \mapsto \mathsf{T}$, 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. $\mathsf{A} \vee \mathsf{B} \widehat{=} \neg(\neg\mathsf{A} \wedge \neg\mathsf{B})$ and $\mathsf{A} \Rightarrow \mathsf{B} \widehat{=} \neg\mathsf{A} \vee \mathsf{B}$, and $T \widehat{=} P \vee \neg P$ (only need to treat $\neg, \wedge$ directly)

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 2.6.** 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.
  - $\mathcal{I}(\neg) : \mathcal{D}_o \to \mathcal{D}_o; \mathsf{T} \mapsto \mathsf{F}, \mathsf{F} \mapsto \mathsf{T}$
  - $\mathcal{I}(\wedge) : \mathcal{D}_o \times \mathcal{D}_o \to \mathcal{D}_o; \langle \alpha, \beta \rangle \mapsto \mathsf{T}$, 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. $\mathsf{A} \vee \mathsf{B} \widehat{=} \neg(\neg\mathsf{A} \wedge \neg\mathsf{B})$ and $\mathsf{A} \Rightarrow \mathsf{B} \widehat{=} \neg\mathsf{A} \vee \mathsf{B}$, 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 $\langle \mathcal{D}_o, \mathcal{I} \rangle$.

Semantics of PL^0 (Evaluation)

- **Problem:** The interpretation function $\mathcal{I}$ only assigns meaning to connectives.
- **Definition 2.7.** A variable assignment $\varphi\colon \mathcal{V}_0 \to \mathcal{D}_o$ assigns values to propositional variables.
- **Definition 2.8.** 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)
  - $\mathcal{I}_\varphi(\neg \mathsf{A}) = \mathcal{I}(\neg)(\mathcal{I}_\varphi(\mathsf{A}))$.
  - $\mathcal{I}_\varphi(\mathsf{A} \wedge \mathsf{B}) = \mathcal{I}(\wedge)(\mathcal{I}_\varphi(\mathsf{A}), \mathcal{I}_\varphi(\mathsf{B}))$.
- **Note:** $\mathcal{I}_\varphi(\mathsf{A} \vee \mathsf{B}) = \mathcal{I}_\varphi(\neg(\neg\mathsf{A} \wedge \neg\mathsf{B}))$ is only determined by $\mathcal{I}_\varphi(\mathsf{A})$ and $\mathcal{I}_\varphi(\mathsf{B})$, so we think of the defined connectives as logical constants as well.
- **Alternative Notation:** Write $[\![\mathsf{A}]\!]_\varphi$ for $\mathcal{I}_\varphi(\mathsf{A})$. (and $[\![\mathsf{A}]\!]$, if A is ground)
- **Definition 2.9.** Two formulae A and B are called equivalent, iff $\mathcal{I}_\varphi(\mathsf{A}) = \mathcal{I}_\varphi(\mathsf{B})$ for all variable assignments φ.

Computing Semantics

- **Example 2.10.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

$$\mathcal{I}_\varphi(P_1 \vee P_2 \vee \neg(\neg P_1 \wedge P_2) \vee P_3 \wedge P_4)$$

Computing Semantics

- **Example 2.11.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

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

Computing Semantics

- **Example 2.12.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

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

Computing Semantics

- **Example 2.13.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

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

Computing Semantics

- **Example 2.14.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

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

Computing Semantics

- **Example 2.15.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

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

Computing Semantics

- **Example 2.16.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

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

Computing Semantics

- **Example 2.17.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

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

Computing Semantics

- **Example 2.18.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

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

Computing Semantics

- **Example 2.19.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

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

Computing Semantics

- **Example 2.20.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

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

Computing Semantics

- **Example 2.21.** Let $\varphi := [\mathsf{T}/P_1], [\mathsf{F}/P_2], [\mathsf{T}/P_3], [\mathsf{F}/P_4], \ldots$ then

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

Computing Semantics

- **Example 2.22.** 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!

Propositional Identities

- **Definition 2.23.** 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)

Semantic Properties of Propositional Formulae

- **Definition 2.24.** Let $\mathcal{M} := \langle \mathcal{U}, \mathcal{I} \rangle$ be our model, then we call A
  - true under φ (φ satisfies A) in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathsf{A}) = \mathsf{T}$, (write $\mathcal{M} \models^\varphi \mathsf{A}$)
  - false under φ (φ falsifies A) in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathsf{A}) = \mathsf{F}$, (write $\mathcal{M} \not\models^\varphi \mathsf{A}$)
  - satisfiable in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathsf{A}) = \mathsf{T}$ for some assignment φ,
  - valid in $\mathcal{M}$, iff $\mathcal{M} \models^\varphi \mathsf{A}$ for all variable assignments φ,
  - falsifiable in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathsf{A}) = \mathsf{F}$ for some assignments φ, and
  - unsatisfiable in $\mathcal{M}$, iff $\mathcal{I}_\varphi(\mathsf{A}) = \mathsf{F}$ for all assignments φ.
- **Example 2.25.** $x \vee x$ is satisfiable and falsifiable.
- **Example 2.26.** $x \vee \neg x$ 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 $\varphi \models \mathsf{A}$ from the preview for $\mathcal{M} \models^\varphi \mathsf{A}$.
- **Definition 2.27 (Entailment).** (aka. logical consequence)
  We say that A entails B (write $\mathsf{A} \models \mathsf{B}$), iff $\mathcal{I}_\varphi(\mathsf{B}) = \mathsf{T}$ for all φ with $\mathcal{I}_\varphi(\mathsf{A}) = \mathsf{T}$ (i.e. all assignments that make A true also make B true)

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

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.
     - 2.1 Persons with red or green hair do not study AI.
     - 2.2 Neither the Physics nor the Chinese students have black hair.
     - 2.3 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

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.
     - 2.1 Persons with red or green hair do not study AI.
     - 2.2 Neither the Physics nor the Chinese students have black hair.
     - 2.3 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 $ai(S) \vee ai(N) \vee ai(J)$
     - 2.1 $ai(x) \Rightarrow \neg red(x) \wedge \neg gre(x)$.
     - 2.2 $phy(x) \Rightarrow \neg bla(x)$ and $chi(x) \Rightarrow \neg bla(x)$.
     - 2.3 $phy(S) \wedge chi(J) \vee phy(J) \wedge chi(S)$.

  Now, we obtain new knowledge via entailment steps:
  3. 1. together with 2.1 entails that $ai(x) \Rightarrow bla(x)$ for every $x \in \{S, N, J\}$,
  4. thus $\neg bla(S) \wedge \neg bla(J)$ by 2.3 and 2.2 and
  5. so $\neg ai(S) \wedge \neg ai(J)$ by 3. and 4.
  6. With 2. the latter entails $ai(N)$.

10.3 Inference in Propositional Logics

Agents that Think Rationally

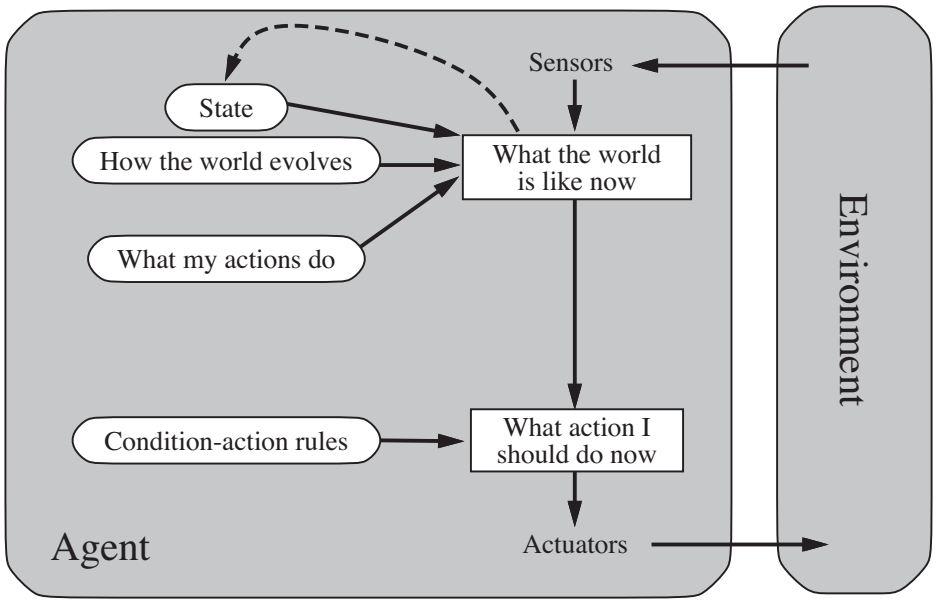
- **Problem:** But how can we build an agent that can do the necessary inferences?

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.

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 3.3.** 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.

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 3.4.** 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 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: $P, Q, R \ldots$ and implication: $\Rightarrow$
- **Semantics:** $\mathcal{I}_\varphi(P) = \varphi(P)$ and $\mathcal{I}_\varphi(\mathsf{A} \Rightarrow \mathsf{B}) = \mathsf{T}$, iff $\mathcal{I}_\varphi(\mathsf{A}) = \mathsf{F}$ or $\mathcal{I}_\varphi(\mathsf{B}) = \mathsf{T}$.
- **Definition 3.5.** The Hilbert calculus $\mathcal{H}^0$ consists of the inference rules:

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

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

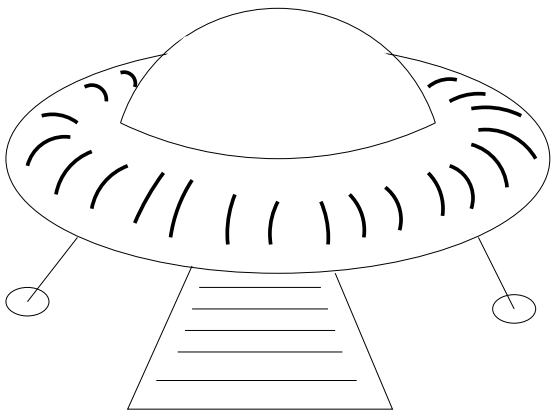
- **Example 3.6.** A $\mathcal{H}^0$ theorem $\mathsf{C} \Rightarrow \mathsf{C}$ and its proof

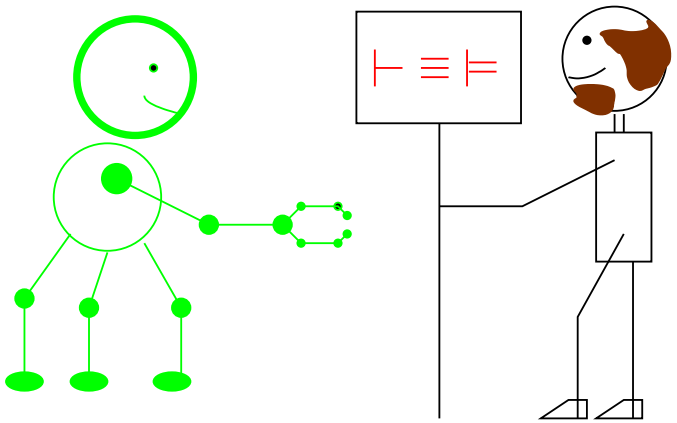
Proof: We show that $\emptyset \vdash_{\mathcal{H}^0} \mathsf{C} \Rightarrow \mathsf{C}$

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

Soundness and Completeness

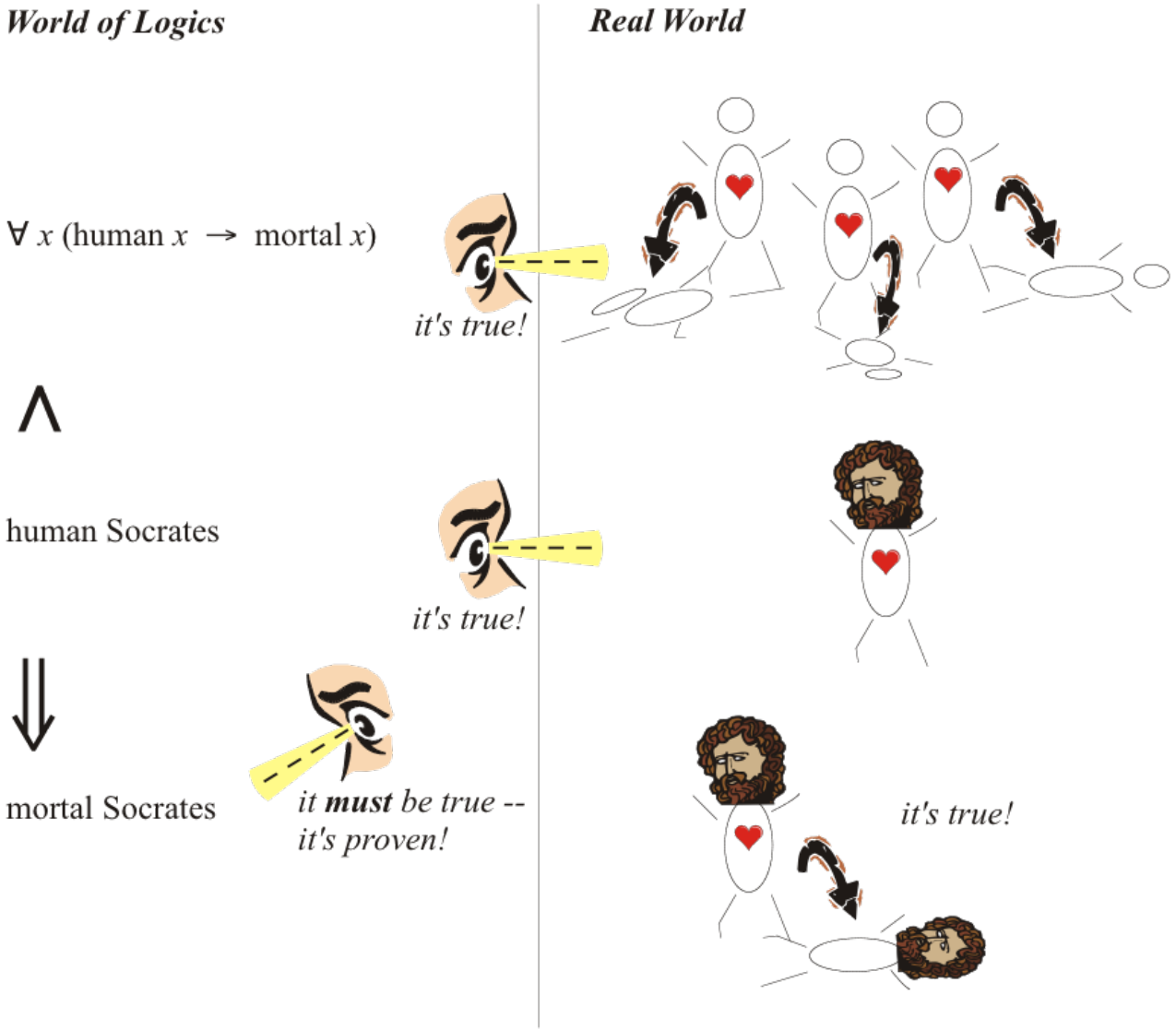
- **Definition 3.7.** Let $\mathcal{L} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system, then we call a calculus $\mathcal{C}$ for $\mathcal{L}$,
  - sound (or correct), iff $\mathcal{H} \models A$, whenever $\mathcal{H} \vdash_{\mathcal{C}} A$, and
  - complete, iff $\mathcal{H} \vdash_{\mathcal{C}} A$, whenever $\mathcal{H} \models A$.
- **Goal:** Find calculi C, such that $\vdash_C A$ iff $\models A$ (provability and validity coincide)
  - To TRUTH through PROOF (CALCULEMUS [Leibniz ∼1680])





The Miracle of Logic

- Purely formal derivations are true in the real world!



10.4 Propositional Natural Deduction Calculus

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

- **Idea:** $\mathcal{ND}_0$ tries to mimic human argumentation for theorem proving.
- **Definition 4.1.** The propositional natural deduction calculus $\mathcal{ND}_0$ has inference rules for the introduction and elimination of connectives:

Introduction | Elimination | Axiom

$$\frac{A \quad B}{A \wedge B}\,\mathcal{ND}_0\wedge I \qquad \frac{A \wedge B}{A}\,\mathcal{ND}_0\wedge E_l \qquad \frac{A \wedge B}{B}\,\mathcal{ND}_0\wedge E_r$$

$$\frac{}{A \vee \neg A}\,\mathcal{ND}_0\mathrm{TND}$$

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

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

- **Definition 4.2.** Given a set $\mathcal{H} \subseteq wff_0(\mathcal{V}_0)$ of assumptions and a conclusion C, we write $\mathcal{H} \vdash_{\mathcal{ND}_0} C$, 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)

Natural Deduction: Examples

- **Example 4.3 (Inference with Local Hypotheses).**

$$\dfrac{\dfrac{\dfrac{[\mathsf{A} \wedge \mathsf{B}]^1}{\mathsf{B}}\mathcal{ND}_0\wedge E_r \quad \dfrac{[\mathsf{A} \wedge \mathsf{B}]^1}{\mathsf{A}}\mathcal{ND}_0\wedge E_l}{\mathsf{B} \wedge \mathsf{A}}\mathcal{ND}_0\wedge I}{\mathsf{A} \wedge \mathsf{B} \Rightarrow \mathsf{B} \wedge \mathsf{A}}\mathcal{ND}_0 \Rightarrow I^1$$

$$\dfrac{\dfrac{\dfrac{[A]^1 \quad [B]^2}{A}\mathcal{ND}_0 \Rightarrow I^2}{B \Rightarrow A}\mathcal{ND}_0 \Rightarrow I^1}{A \Rightarrow B \Rightarrow A}$$

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

- **Theorem 4.4.** $\mathcal{H}, A \vdash_{\mathcal{ND}_0} B$, *iff* $\mathcal{H} \vdash_{\mathcal{ND}_0} A \Rightarrow B$.
- *Proof:* We show the two directions separately
  1. If $\mathcal{H}, A \vdash_{\mathcal{ND}_0} B$, then $\mathcal{H} \vdash_{\mathcal{ND}_0} A \Rightarrow B$ by $\mathcal{ND}_0 \Rightarrow I$, and
  2. If $\mathcal{H} \vdash_{\mathcal{ND}_0} A \Rightarrow B$, then $\mathcal{H}, A \vdash_{\mathcal{ND}_0} A \Rightarrow B$ by weakening and $\mathcal{H}, A \vdash_{\mathcal{ND}_0} B$ by $\mathcal{ND}_0 \Rightarrow E$.

More Rules for Natural Deduction

- **Note:** $\mathcal{ND}_0$ does not try to be minimal, but comfortable to work in!
- **Definition 4.5.** $\mathcal{ND}_0$ has the following additional inference rules for the remaining connectives.

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

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

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

- **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 4.6.** A judgment is a meta-statement about the provability of propositions.
- **Definition 4.7.** A sequent is a judgment of the form $\mathcal{H}\vdash A$ about the provability of the formula A from the set $\mathcal{H}$ of hypotheses. We write $\vdash A$ for $\emptyset\vdash A$.
- **Idea:** Reformulate $\mathcal{ND}_0$ inference rules so that they act on sequents.
- **Example 4.8.** We give the sequent style version of **??**:

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

$$\dfrac{\dfrac{\dfrac{}{A,B\vdash A}\,\mathcal{ND}^0_\vdash Ax}{A\vdash B\Rightarrow A}\,\mathcal{ND}^0_\vdash \Rightarrow I}{\vdash A\Rightarrow B\Rightarrow A}\,\mathcal{ND}^0_\vdash \Rightarrow I$$

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

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

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

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

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

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

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

Linearized Notation for (Sequent-Style) ND Proofs

- **Definition 4.10.** Linearized notation for sequent-style ND proofs

1.	$\mathcal{H}_1$	$\vdash$	A_1	$(\mathcal{J}_1)$	
2.	$\mathcal{H}_2$	$\vdash$	A_2	$(\mathcal{J}_2)$	corresponds to $\dfrac{\mathcal{H}_1\vdash A_1 \quad \mathcal{H}_2\vdash A_2}{\mathcal{H}_3\vdash A_3}\,\mathcal{R}$
3.	$\mathcal{H}_3$	$\vdash$	A_3	$(\mathcal{J}_3 1,2)$	

- **Example 4.11.** We show a linearized version of the $\mathcal{ND}_0$ examples **??**

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

$$\dfrac{\dfrac{\dfrac{}{A,B\vdash A}\,\mathcal{ND}^0_\vdash A}{A\vdash B\Rightarrow A}\,\mathcal{ND}^0_\vdash}{\vdash A\Rightarrow B\Rightarrow A}\,\mathcal{ND}$$

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

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

10.5 Predicate Logic Without Quantifiers

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)

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 5.2.** 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 5.3.** 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	A	$::=$	$p^k(t_1, \ldots, t_k)$	atomic
		$\mid$	$\neg \mathsf{A}$	negation
		$\mid$	$\mathsf{A}_1 \wedge \mathsf{A}_2$	conjunction

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

- **Definition 5.4.** Domains $\mathcal{D}_0 = \{\mathsf{T}, \mathsf{F}\}$ of truth values and $\mathcal{D}_\iota \neq \emptyset$ of individuals.
- **Definition 5.5.** Interpretation $\mathcal{I}$ assigns values to constants, e.g.
  - $\mathcal{I}(\neg): \mathcal{D}_0 \to \mathcal{D}_0$; $\mathsf{T} \mapsto \mathsf{F}$; $\mathsf{F} \mapsto \mathsf{T}$ and $\mathcal{I}(\wedge) = \ldots$ (as in PL^0)
  - $\mathcal{I}: \Sigma^f_0 \to \mathcal{D}_\iota$ (interpret individual constants as individuals)
  - $\mathcal{I}: \Sigma^f_k \to {\mathcal{D}_\iota}^k \to \mathcal{D}_\iota$ (interpret function constants as functions)
  - $\mathcal{I}: {\Sigma^p}_k \to \mathcal{P}({\mathcal{D}_\iota}^k)$ (interpret predicate constants as relations)
- **Definition 5.6.** The value function $\mathcal{I}$ assigns values to formulae: (recursively)
  - $\mathcal{I}(f(\mathsf{A}^1, \ldots, \mathsf{A}^k)) := \mathcal{I}(f)(\mathcal{I}(\mathsf{A}^1), \ldots, \mathcal{I}(\mathsf{A}^k))$
  - $\mathcal{I}(p(\mathsf{A}^1, \ldots, \mathsf{A}^k)) := \mathsf{T}$, iff $\langle \mathcal{I}(\mathsf{A}^1), \ldots, \mathcal{I}(\mathsf{A}^k)\rangle \in \mathcal{I}(p)$
  - $\mathcal{I}(\neg\mathsf{A}) = \mathcal{I}(\neg)(\mathcal{I}(\mathsf{A}))$ and $\mathcal{I}(\mathsf{A} \wedge \mathsf{B}) = \mathcal{I}(\wedge)(\mathcal{I}(\mathsf{A}), \mathcal{I}(\mathsf{G}))$ (just as in PL^0)
- **Definition 5.7.** Model: $\mathcal{M} = \langle \mathcal{D}_\iota, \mathcal{I}\rangle$ varies in $\mathcal{D}_\iota$ and $\mathcal{I}$.
- **Theorem 5.8.** $\mathrm{PL}^{\mathrm{nq}}$ *is isomorphic to* PL^0 *(interpret atoms as prop. variables)*

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

- **Example 5.9.** 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 5.10 (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, $\mathcal{I}(R(a, b) \wedge P(c)) = \mathsf{F}$.

$\mathrm{PL}^{\mathrm{nq}}$ 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 5.11.** *For every model $\mathcal{M} = \langle \mathcal{D}_\iota, \mathcal{I} \rangle$, there is a variable assignment $\varphi_\mathcal{M}$, such that $\mathcal{I}_{\varphi_\mathcal{M}}(\mathsf{A}) = \mathcal{I}(\mathsf{A})$.*
- *Proof sketch:* We just define $\varphi_\mathcal{M}(X) := \mathcal{I}(\theta_\Sigma^{-1}(X))$
- **Lemma 5.12.** *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 $\mathcal{I}_\psi(\mathsf{A}) = \mathcal{I}^\psi(\mathsf{A})$.*
- *Proof sketch:* see next slide
- **Corollary 5.13.** $\mathrm{PL}^{\mathrm{nq}}$ *is isomorphic to* PL^0*, i.e. the following diagram commutes:*

$$\begin{array}{ccc} \langle \mathcal{D}^\psi, \mathcal{I}^\psi \rangle & \xleftarrow{\psi \mapsto \mathcal{M}^\psi} & \mathcal{V}_\Sigma \to \{\mathsf{T}, \mathsf{F}\} \\ \mathcal{I}^\psi() \uparrow & & \uparrow \mathcal{I}_{\varphi_\mathcal{M}}() \\ \mathrm{PL}^{\mathrm{nq}}(\Sigma) & \xrightarrow{\theta_\Sigma} & \mathrm{PL}^0(\mathcal{A}_\Sigma) \end{array}$$

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

Valuation and Satisfiability

- **Lemma 5.14.** *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* $\mathcal{I}_\psi(\mathsf{A}) = \mathcal{I}^\psi(\mathsf{A})$.
- *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 $\mathrm{PL}^{\mathrm{mq}}$ terms over Σ, and
     - $\mathcal{I}^\psi(f) : {\mathcal{D}_\iota}^k \to {\mathcal{D}^\psi}^k ; \langle \mathsf{A}_1,\ldots,\mathsf{A}_k\rangle \mapsto f(\mathsf{A}_1,\ldots,\mathsf{A}_k)$ for $f \in \Sigma^f_k$
     - $\mathcal{I}^\psi(p) := \{\langle \mathsf{A}_1,\ldots,\mathsf{A}_k\rangle \mid \psi(\theta_\psi^{-1} p(\mathsf{A}_1,\ldots,\mathsf{A}_k)) = \mathsf{T}\}$ for $p \in \Sigma^p$.
  2. We show $\mathcal{I}^\psi(\mathsf{A}) = \mathsf{A}$ for terms A by induction on A
     2.1. If $\mathsf{A} = c$, then $\mathcal{I}^\psi(\mathsf{A}) = \mathcal{I}^\psi(c) = c = \mathsf{A}$
     2.2. If $\mathsf{A} = f(\mathsf{A}_1,\ldots,\mathsf{A}_n)$ then
     $\mathcal{I}^\psi(\mathsf{A}) = \mathcal{I}^\psi(f)(\mathcal{I}(\mathsf{A}_1),\ldots,\mathcal{I}(\mathsf{A}_n)) = \mathcal{I}^\psi(f)(\mathsf{A}_1,\ldots,\mathsf{A}_k) = \mathsf{A}$.
  3. For a $\mathrm{PL}^{\mathrm{mq}}$ formula A we show that $\mathcal{I}^\psi(\mathsf{A}) = \mathcal{I}_\psi(\mathsf{A})$ by induction on A.
     3.1. If $\mathsf{A} = p(\mathsf{A}_1,\ldots,\mathsf{A}_k)$, then $\mathcal{I}^\psi(\mathsf{A}) = \mathcal{I}^\psi(p)(\mathcal{I}(\mathsf{A}_1),\ldots,\mathcal{I}(\mathsf{A}_n)) = \mathsf{T}$, iff $\langle \mathsf{A}_1,\ldots,\mathsf{A}_k\rangle \in \mathcal{I}^\psi(p)$, iff $\psi(\theta_\psi^{-1}\mathsf{A}) = \mathsf{T}$, so $\mathcal{I}^\psi(\mathsf{A}) = \mathcal{I}_\psi(\mathsf{A})$ as desired.
     3.2. If $\mathsf{A} = \neg\mathsf{B}$, then $\mathcal{I}^\psi(\mathsf{A}) = \mathsf{T}$, iff $\mathcal{I}^\psi(\mathsf{B}) = \mathsf{F}$, iff $\mathcal{I}^\psi(\mathsf{B}) = \mathcal{I}_\psi(\mathsf{B})$, iff $\mathcal{I}^\psi(\mathsf{A}) = \mathcal{I}_\psi(\mathsf{A})$.
     3.3. If $\mathsf{A} = \mathsf{B} \wedge \mathsf{C}$ then we argue similarly
  4. Hence $\mathcal{I}^\psi(\mathsf{A}) = \mathcal{I}_\psi(\mathsf{A})$ for all $\mathrm{PL}^{\mathrm{mq}}$ formulae and we have concluded the proof.

10.6 Conclusion

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. A entails B if all interpretations that satisfy A also satisfy 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.

Chapter 11
Formal Systems: Syntax, Semantics, Entailment, and Derivation in General

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 $wff_0(\mathcal{V}_0)$ 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 $\mathcal{M} \models A$, and
  - entailment i.e whether $A \models B$.
- **Observation:** In particular, the inductive/compositional nature of $wff_0(\mathcal{V}_0)$ and $\mathcal{I}_\varphi : wff_0(\mathcal{V}_0) \to \mathcal{D}_0$ are secondary.
- **Idea:** Concentrate on language, models $(\mathcal{M}, \varphi)$, and satisfiability.

Logical Systems

- **Definition 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 0.2 (Propositional Logic).** $\langle wff(\Sigma_{PL^0}, \mathcal{V}_{PL^0}), \mathcal{K}_o, \models \rangle$ is a logical system, if we define $\mathcal{K}_o := \mathcal{V}_0 \rightharpoonup \mathcal{D}_0$ (the set of variable assignments) and $\varphi \models A$ iff $\mathcal{I}_\varphi(A) = T$.
- **Definition 0.3.** Let $\langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system, $M \in \mathcal{K}$ a model and $A \in \mathcal{L}$ a formula. Then we say that A is
  - satisfied by M iff $M \models A$.
  - satisfiable iff A is satisfied by some model.
  - unsatisfiable iff A is not satisfiable.
  - falsified by M iff $M \not\models A$.
  - valid or unfalsifiable (write $\models A$) iff A is satisfied by every model.
  - invalid or falsifiable (write $\not\models A$) iff A is not valid.

Derivation Relations and Inference Rules

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

Derivation Relations and Inference Rules

- **Definition 0.8.** 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
  - $\mathcal{H} \vdash A$, if $A \in \mathcal{H}$ ($\vdash$ is proof reflexive),
  - $\mathcal{H} \vdash A$ and $(\mathcal{H}' \cup \{A\}) \vdash B$ imply $(\mathcal{H} \cup \mathcal{H}') \vdash B$ ($\vdash$ is proof transitive),
  - $\mathcal{H} \vdash A$ and $\mathcal{H} \subseteq \mathcal{H}'$ imply $\mathcal{H}' \vdash A$ ($\vdash$ is monotonic or admits weakening).
- **Definition 0.9.** 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

$$\frac{A_1 \quad \ldots \quad A_n}{C}\,\mathcal{N}$$

  where $A_1, \ldots, A_n$ and C are schemata for words in $\mathcal{L}$ and $\mathcal{N}$ is a name. The A_i are called assumptions of $\mathcal{N}$, and C is called its conclusion.
  Any $n+1$-tuple

$$\frac{a_1 \quad \ldots \quad a_n}{c}$$

  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 $a_1, \ldots, a_n \in M$ to obtain c.
- **Definition 0.10.** An inference rule without assumptions is called an axiom.
- **Definition 0.11.** A calculus (or inference system) is a formal language $\mathcal{L}$ equipped with a set $\mathcal{C}$ of inference rules over $\mathcal{L}$.

Derivations

- **Definition 0.12.** 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 $C \in \mathcal{L}$ from a set $\mathcal{H} \subseteq \mathcal{L}$ of hypotheses (write $\mathcal{H} \vdash_{\mathcal{C}} C$) is a sequence $A_1, \ldots, A_m$ of $\mathcal{L}$-formulae, such that
  - $A_m = C$, (derivation culminates in C)
  - for all $1 \leq i \leq m$, either $A_i \in \mathcal{H}$, or (hypothesis)
  - there is an inference rule $\frac{A_{l_1} \quad \ldots \quad A_{l_k}}{A_i}$ in $\mathcal{C}$ with $l_j < i$ for all $j \leq k$. (rule application)

  We can also see a derivation as a derivation tree, where the A_{l_j} are the children of the node A_i.
- **Example 0.13.**

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.

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

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 0.14.** Let $\mathcal{C}$ be a calculus, then a $\mathcal{C}$-derivation $\emptyset \vdash_{\mathcal{C}} A$ is called a proof of A and if one exists (write $\vdash_{\mathcal{C}} A$) then A is called a $\mathcal{C}$-theorem.
  Definition 0.15. The act of finding a proof for A is called proving A.
- **Definition 0.16.** 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 0.17.** An inference rule

$$\frac{A_1 \quad \ldots \quad A_n}{C}$$

  is called derivable (or a derived rule) in a calculus $\mathcal{C}$, if there is a $\mathcal{C}$-derivation $A_1, \ldots, A_n \vdash_{\mathcal{C}} C$.
- **Observation 0.18.** *Derivable inference rules are admissible, but not the other way around.*

Chapter 12
Machine-Oriented Calculi for Propositional Logic

12.1 Test Calculi

Automated Deduction as an Agent Inference Procedure

- **Recall:** Our knowledge of the cave entails a definite Wumpus position! (slide 318)
- **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.

Unsatisfiability Theorem

- **Theorem 1.1 (Unsatisfiability Theorem).** $\mathcal{H} \models \mathsf{A}$ *iff* $\mathcal{H} \cup \{\neg \mathsf{A}\}$ *is unsatisfiable.*
- *Proof:* We prove both directions separately
  1. "$\Rightarrow$": Say $\mathcal{H} \models \mathsf{A}$
     1.1. For any φ with $\varphi \models \mathcal{H}$ we have $\varphi \models \mathsf{A}$ and thus $\varphi \not\models (\neg \mathsf{A})$.
  2. "$\Leftarrow$": Say $\mathcal{H} \cup \{\neg \mathsf{A}\}$ is unsatisfiable.
     2.1. For any φ with $\varphi \models \mathcal{H}$ we have $\varphi \not\models (\neg \mathsf{A})$ and thus $\varphi \models \mathsf{A}$.
- **Observation 1.2.** *Entailment can be tested via satisfiability.*

Test Calculi: A Paradigm for Automating Inference

- **Definition 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 1.4.** Automated theorem proving (ATP) is the automation of theorem proving

Test Calculi: A Paradigm for Automating Inference

- **Definition 1.6.** 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 1.7.** Automated theorem proving (ATP) is the automation of theorem proving
- **Idea:** A set $\mathcal{H}$ of hypotheses and a conjecture A induce a search problem $\Pi_{\mathcal{C}}^{\mathcal{H} \models \mathsf{A}} := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$, 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 $\mathsf{A} \in \mathcal{S}$.
- **Problem:** ATP as a search problem does not admit good heuristics, since these need to take the conjecture $\mathcal{A}$ into account.

Test Calculi: A Paradigm for Automating Inference

- **Definition 1.9.** 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 1.10.** Automated theorem proving (ATP) is the automation of theorem proving
- **Idea:** A set $\mathcal{H}$ of hypotheses and a conjecture A induce a search problem $\Pi_{\mathcal{C}}^{\mathcal{H} \models \mathsf{A}} := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$, 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 $\mathsf{A} \in \mathcal{S}$.
- **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 1.11.** 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.

Test Calculi: A Paradigm for Automating Inference

- **Definition 1.12.** 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 1.13.** Automated theorem proving (ATP) is the automation of theorem proving
- **Idea:** A set $\mathcal{H}$ of hypotheses and a conjecture A induce a search problem $\Pi_{\mathcal{C}}^{\mathcal{H} \models \mathsf{A}} := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$, 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 $\mathsf{A} \in \mathcal{S}$.
- **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 1.14.** 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 $\mathcal{H} \cup \{\neg \mathsf{A}\}$ 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)

12.1.1 Normal Forms

Recap: Atoms and Literals

- **Definition 1.15.** A formula is called atomic (or an atom) if it does not contain logical constants, else it is called complex.
- **Definition 1.16.** Let $\langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system and $A \in \mathcal{L}$, then we call a pair A^α of a formula and a truth value $\alpha \in \{T, F\}$ a labeled formula. For a set Φ of formulae we use $\Phi^\alpha := \{A^\alpha \mid A \in \Phi\}$.
  We call a labeled formula A^T positive and A^F negative.
  Definition 1.17. Let $\langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system and A^α a labeled formula. Then we say that $\mathcal{M} \in \mathcal{K}$ satisfies A^α (written $\mathcal{M} \models A$), iff $\alpha = T$ and $\mathcal{M} \models A$ or $\alpha = F$ and $\mathcal{M} \not\models A$.
- **Definition 1.18.** Let $\langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system, $A \in \mathcal{L}$ atomic, and $\alpha \in \{T, F\}$, then we call a A^α a literal.
- **Intuition:** To satisfy a formula, we make it "true". To satisfy a labeled formula A^α, it must have the truth value α.
- **Definition 1.19.** For a literal A^α, we call the literal A^β with $\alpha \neq \beta$ the opposite literal (or partner literal).

Alternative Definition: Literals

- **Note:** Literals are often defined without recurring to labeled formulae:
- **Definition 1.20.** A literal is an atom A (positive literal) or negated atom $\neg$A (negative literal). A and $\neg$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 1.21.** 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 1.22.** 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 1.23.** *Every formula has equivalent formulae in CNF and DNF.*

12.2 Analytical Tableaux

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 ($\rightsquigarrow$ models).
- **Example 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} $P^{\mathsf{F}} \mid Q^{\mathsf{F}}$ $\bot \mid \bot$	$(P \wedge (Q \vee \neg R) \wedge \neg Q)^{\mathsf{T}}$ $(P \wedge (Q \vee \neg R))^{\mathsf{T}}$ $\neg Q^{\mathsf{T}}$ Q^{F} P^{T} $(Q \vee \neg R)^{\mathsf{T}}$ $Q^{\mathsf{T}} \mid \neg R^{\mathsf{T}}$ $\bot \mid R^{\mathsf{F}}$
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)

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

$$\frac{(A \wedge B)^T}{\begin{array}{c}A^T\\B^T\end{array}}\mathcal{T}_0\wedge \qquad \frac{(A \wedge B)^F}{A^F \mid B^F}\mathcal{T}_0\vee \qquad \frac{\neg A^T}{A^F}\mathcal{T}_0\neg^T \qquad \frac{\neg A^F}{A^T}\mathcal{T}_0\neg^F \qquad \frac{\begin{array}{cc}A^\alpha & \\ A^\beta & \alpha \neq \beta\end{array}}{\bot}\mathcal{T}_0\bot$$

  Use rules exhaustively as long as they contribute new material ($\rightsquigarrow$ termination)
- **Definition 2.3.** We call any tree ($\mid$ introduces branches) produced by the $\mathcal{T}_0$ inference rules from a set Φ of labeled formulae a tableau for Φ.
- **Definition 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 $\Box$ symbol.

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

- **Definition 2.6 ($\mathcal{T}_0$-Theorem/Derivability).** A is a $\mathcal{T}_0$-theorem ($\vdash_{\mathcal{T}_0} A$), iff there is a closed tableau with A^F at the root.
  $\Phi \subseteq \mathit{wff}_0(\mathcal{V}_0)$ derives A in $\mathcal{T}_0$ ($\Phi \vdash_{\mathcal{T}_0} A$), iff there is a closed tableau starting with A^F and Φ^T. The tableau with only a branch of A^F and Φ^T is called initial for $\Phi \vdash_{\mathcal{T}_0} A$.

A Valid Real-World Example

- **Example 2.8.** *If Mary loves Bill and John loves Mary, then John loves Mary*

$$(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john},\mathrm{mary}) \Rightarrow \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}}$$
$$\neg(\neg\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john},\mathrm{mary})) \wedge \neg\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}}$$
$$(\neg\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john},\mathrm{mary})) \wedge \neg\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}}$$
$$\neg\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}}$$
$$\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}}$$
$$(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}}$$
$$\neg\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}}$$
$$\mathrm{loves}(\mathrm{mary},\mathrm{bill})^{\mathsf{T}}$$
$$\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}}$$
$$\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{F}}$$
$$\bot$$

This is a closed tableau, so the $\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \wedge \mathrm{loves}(\mathrm{john},\mathrm{mary}) \Rightarrow \mathrm{loves}(\mathrm{john},\mathrm{mary})$ is a $\mathcal{T}_0$-theorem.
As we will see, $\mathcal{T}_0$ is sound and complete, so

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

is valid.

Deriving Entailment in $\mathcal{T}_0$

- **Example 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

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

A Falsifiable Real-World Example

- **Example 2.10.** * *If Mary loves Bill or John loves Mary, then John loves Mary*
  Try proving the implication (this fails)

$$((\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \vee \mathrm{loves}(\mathrm{john},\mathrm{mary})) \Rightarrow \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}}$$
$$\neg(\neg\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \vee \mathrm{loves}(\mathrm{john},\mathrm{mary})) \wedge \neg\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}}$$
$$(\neg\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \vee \mathrm{loves}(\mathrm{john},\mathrm{mary})) \wedge \neg\mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}}$$
$$\neg\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}}$$
$$\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{F}}$$
$$\neg\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \vee \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}}$$
$$\neg(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \vee \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{F}}$$
$$(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \vee \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}}$$
$$\mathrm{loves}(\mathrm{mary},\mathrm{bill})^{\mathsf{T}} \mid \mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}}$$
$$\mid \quad \bot$$

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

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

- **Example 2.11.** Does *Mary loves Bill or John loves Mary* entail that *John loves Mary*?

$$\frac{\frac{(\mathrm{loves}(\mathrm{mary},\mathrm{bill}) \vee \mathrm{loves}(\mathrm{john},\mathrm{mary}))^{\mathsf{T}}}{\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{F}}}}{\mathrm{loves}(\mathrm{mary},\mathrm{bill})^{\mathsf{T}} \mid \frac{\mathrm{loves}(\mathrm{john},\mathrm{mary})^{\mathsf{T}}}{\bot}}$$

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.

12.2.2 Practical Enhancements for Tableaux

Derived Rules of Inference

- **Definition 2.12.** An inference rule

$$\frac{A_1 \quad \ldots \quad A_n}{C}$$

  is called derivable (or a derived rule) in a calculus $\mathcal{C}$, if there is a $\mathcal{C}$-derivation $A_1, \ldots, A_n \vdash_{\mathcal{C}} C$.

- **Definition 2.13.** We have the following derivable inference rules in $\mathcal{T}_0$:

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

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

$$\begin{array}{c}A^T\\(A \Rightarrow B)^T\\(\neg A \vee B)^T\\\neg(\neg\neg A \wedge \neg B)^T\\(\neg\neg A \wedge \neg B)^F\end{array} \quad \frac{\neg\neg A^F \mid \neg B^F}{\begin{array}{c}\neg A^T\\A^F\\\bot\end{array} \mid B^T}$$

Tableaux with derived Rules (example)

Example 2.14.

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

12.2.3 Soundness and Termination of Tableaux

Soundness (Tableau)

- **Idea:** A test calculus is refutation sound, iff its inference rules preserve satisfiability and the goal formulae are unsatisfiable.
- **Definition 2.15.** A labeled formula A^α is valid under φ, iff $\mathcal{I}_\varphi(A) = \alpha$.
- **Definition 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 2.17.** *$\mathcal{T}_0$ rules transform satisfiable tableaux into satisfiable ones.*
- **Theorem 2.18 (Soundness).** *$\mathcal{T}_0$ is sound, i.e. $\Phi \subseteq wff_0(\mathcal{V}_0)$ 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 2.19 (Completeness).** *$\mathcal{T}_0$ is complete, i.e. if $\Phi \subseteq wff_0(\mathcal{V}_0)$ is valid, then there is a closed tableau $\mathcal{T}$ for Φ^F.*

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

Termination for Tableaux

- **Lemma 2.20.** *$\mathcal{T}_0$ terminates, i.e. every $\mathcal{T}_0$ tableau becomes saturated after finitely many rule applications.*

Termination for Tableaux

- **Lemma 2.22.** *$\mathcal{T}_0$ 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 A^α 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.

Termination for Tableaux

- **Lemma 2.24.** *$\mathcal{T}_0$ 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 A^α 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 2.25.** *$\mathcal{T}_0$ induces a decision procedure for validity in PL^0.*

Proof: We combine the results so far

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

12.3 Resolution for Propositional Logic

12.3.1 Resolution for Propositional Logic

Another Test Calculus: Resolution

- **Definition 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 3.2 (Resolution Calculus).** The resolution calculus $\mathcal{R}_0$ operates a clause sets via a single inference rule:

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

  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 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 $\mathcal{D} : S \vdash_{\mathcal{R}_0} \Box$.

Clause Normal Form Transformation (A calculus)

- **Definition 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 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.

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

- **Definition 3.6.** We write $CNF_0(A^\alpha)$ for the set of all clauses derivable from A^α via the rules above.

Derived Rules of Inference

- **Definition 3.7.** An inference rule

$$\frac{A_1 \quad \ldots \quad A_n}{C}$$

  is called derivable (or a derived rule) in a calculus $\mathcal{C}$, if there is a $\mathcal{C}$-derivation $A_1, \ldots, A_n \vdash_{\mathcal{C}} C$.
- **Idea:** Derived rules make derivations shorter.
- **Example 3.8.**

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

- **Other Derived CNF Rules:**

$$\frac{C \vee (A \Rightarrow B)^T}{C \vee A^F \vee B^T} \qquad \frac{C \vee (A \Rightarrow B)^F}{C \vee A^T \,;\, C \vee B^F}$$

$$\frac{C \vee (A \wedge B)^T}{C \vee A^T \,;\, C \vee B^T} \qquad \frac{C \vee (A \wedge B)^F}{C \vee A^F \vee B^F}$$

Example: Proving Axiom S with Resolution

- **Example 3.9.** Clause Normal Form transformation

$$\frac{((P \Rightarrow Q \Rightarrow R) \Rightarrow (P \Rightarrow Q) \Rightarrow P \Rightarrow R)^{\mathsf{F}}}{(P \Rightarrow Q \Rightarrow R)^{\mathsf{T}} ; ((P \Rightarrow Q) \Rightarrow P \Rightarrow R)^{\mathsf{F}}}$$
$$\frac{P^{\mathsf{F}} \vee (Q \Rightarrow R)^{\mathsf{T}} ; (P \Rightarrow Q)^{\mathsf{T}} ; (P \Rightarrow R)^{\mathsf{F}}}{P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}} ; P^{\mathsf{F}} \vee Q^{\mathsf{T}} ; P^{\mathsf{T}} ; R^{\mathsf{F}}}$$

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

- **Example 3.10.** Resolution Proof

1	$P^{\mathsf{F}} \vee Q^{\mathsf{F}} \vee R^{\mathsf{T}}$	initial
2	$P^{\mathsf{F}} \vee Q^{\mathsf{T}}$	initial
3	P^{T}	initial
4	R^{F}	initial
5	$P^{\mathsf{F}} \vee Q^{\mathsf{F}}$	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

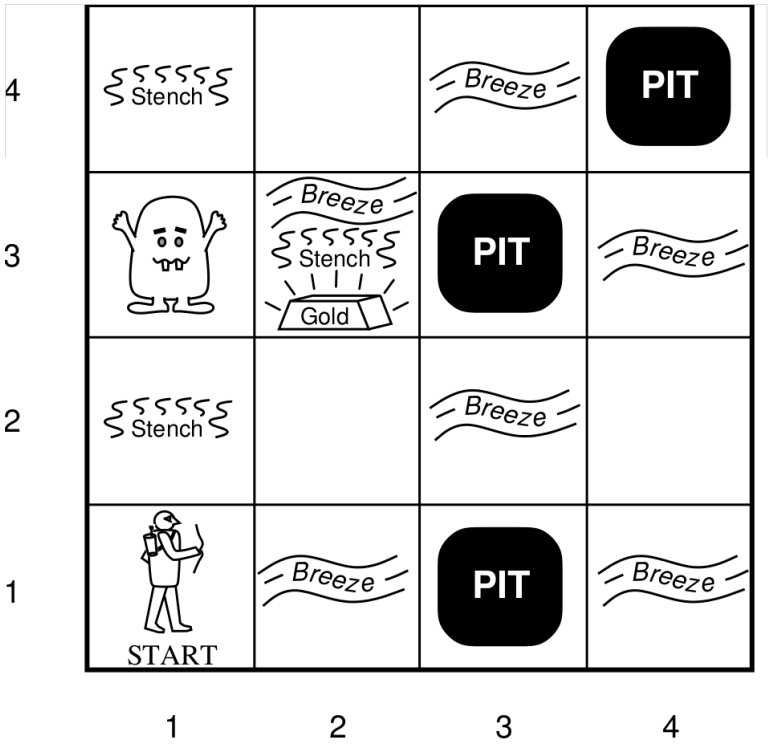
- **Observation:** Let Δ be a clause set, l a literal with $l \in \Delta$ (unit clause), and Δ' be Δ where
  - all clauses $l \vee C$ have been removed and
  - and all clauses $\bar{l} \vee C$ have been shortened to C.

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

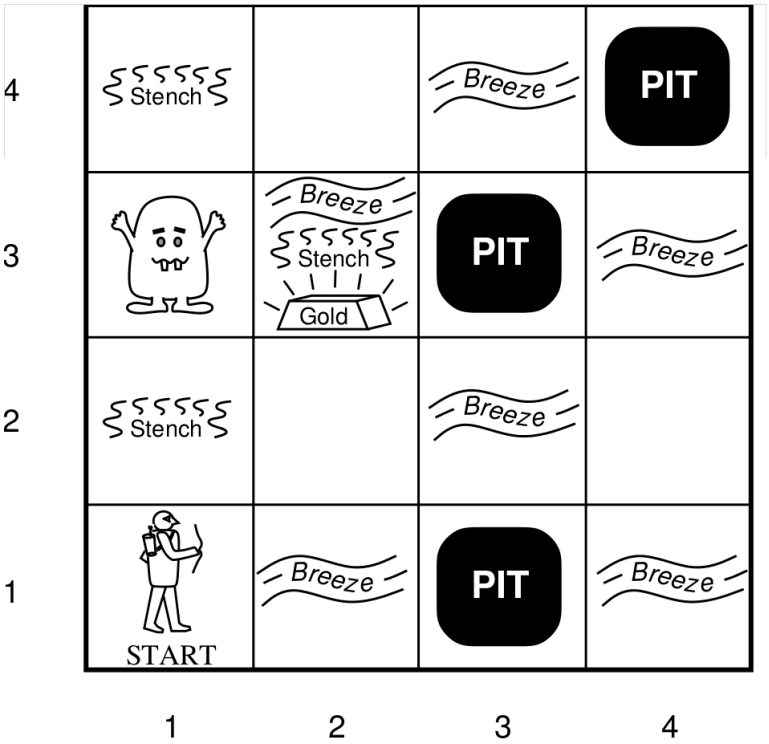
Applying Propositional Inference: Where is the Wumpus?

- **Example 3.12 (Finding the Wumpus).** The situation



Applying Propositional Inference: Where is the Wumpus?

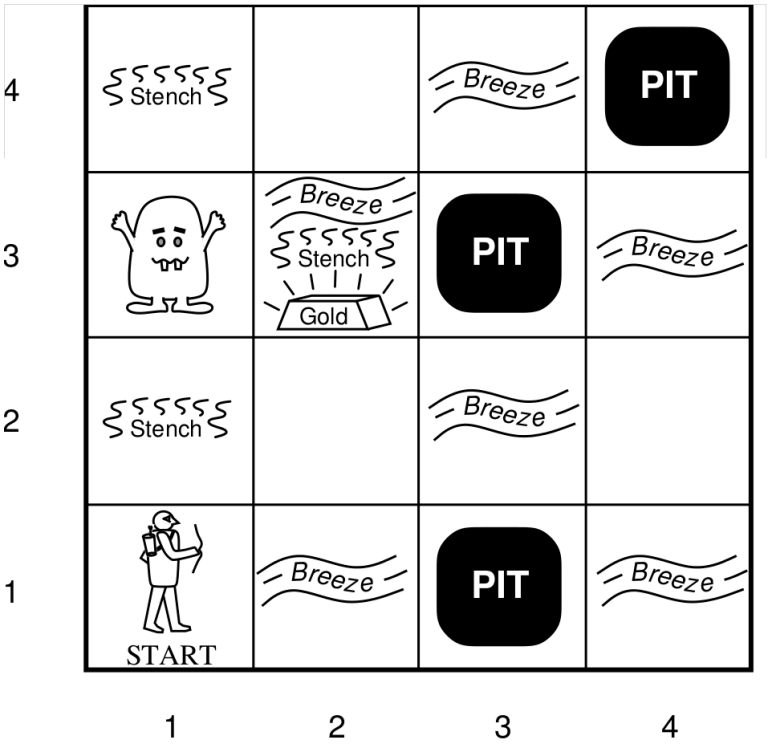
- **Example 3.13 (Finding the Wumpus).** The situation and what the agent knows



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

Applying Propositional Inference: Where is the Wumpus?

- **Example 3.14 (Finding the Wumpus).** The situation and what the agent knows

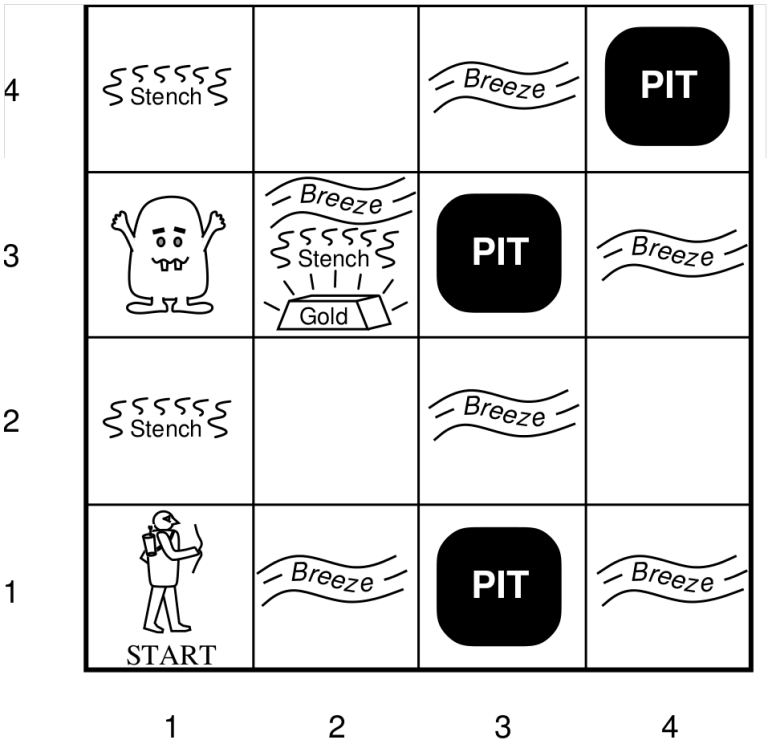


1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 A S OK	2,2	3,2	4,2
1,1 V OK	2,1 B V OK	3,1	4,1

- What should the agent do next and why?

Applying Propositional Inference: Where is the Wumpus?

- **Example 3.15 (Finding the Wumpus).** The situation and what the agent knows

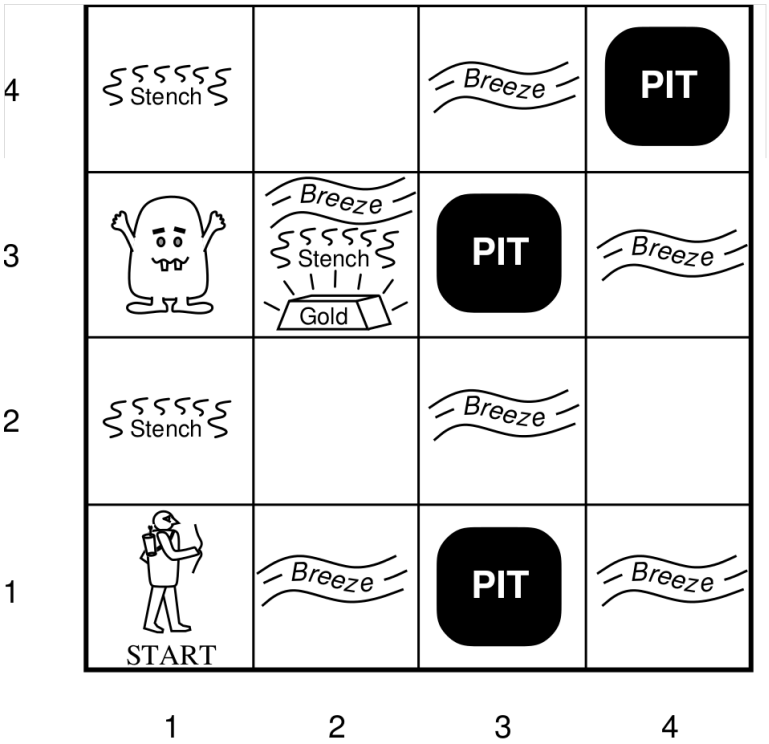


1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 A S OK	2,2	3,2	4,2
1,1 V OK	2,1 B V OK	3,1	4,1

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

Applying Propositional Inference: Where is the Wumpus?

- **Example 3.16 (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)

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: $S_{i,j} \mathrel{\widehat{=}}$ *stench in* $[i,j]$, $W_{i,j} \mathrel{\widehat{=}}$ 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
  $R_1 := \neg S_{1,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,1}$
  $R_2 := \neg S_{2,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{2,1} \wedge \neg W_{2,2} \wedge \neg W_{3,1}$
  $R_3 := \neg S_{1,2} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,2} \wedge \neg W_{1,3}$
  $R_4 := S_{1,2} \Rightarrow (W_{1,3} \vee W_{2,2} \vee W_{1,1})$
  $\vdots$
- **To show:**
  $R_1, R_2, R_3, R_4 \models W_{1,3}$ (we will use resolution)

And Now Using Resolution Conventions

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

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

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

Resolution Proof Killing the Wumpus!

- **Example 3.17 (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: $\square$.

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

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

- **Question:** Where did the ${W_{1,3}}^{\mathsf{F}}$ come from?
- **Observation 3.18.** *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 3.19 (Back to the Wumpus).** In **??**, the agent is in $[1,2]$, it has perceived stench, and the possible actions include shoot, and $\mathrm{goForward}$. 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.

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 $\mathrm{KB} \models \mathrm{A}$ by showing that $\mathrm{KB} \cup \{\neg \mathrm{A}\}$ is unsatisfiable.

Chapter 13
Propositional Reasoning: SAT Solvers

13.1 Introduction

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.

SAT: The Propositional Satisfiability Problem

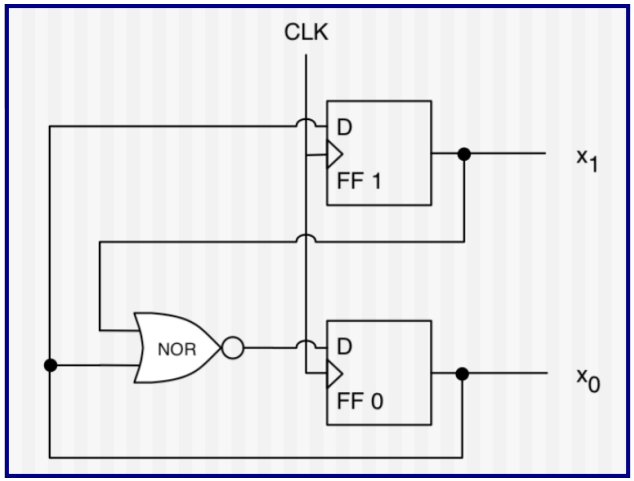
- **Definition 1.1.** The SAT problem (SAT): Given a propositional formula A, decide whether or not A is satisfiable. We denote the class of all SAT problems with SAT
- The SAT problem was the first problem proved to be NP-complete!
- A is commonly assumed to be in CNF. This is without loss of generality, because any 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 1.2.** Tools addressing SAT are commonly referred to as SAT solvers.
- **Recall:** To decide whether $\mathrm{KB} \models \mathrm{A}$, decide satisfiability of $\theta := \mathrm{KB} \cup \{\neg \mathrm{A}\}$: θ is unsatisfiable iff $\mathrm{KB} \models \mathrm{A}$.
- **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 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 1.4 (Encoding CSP as SAT).** *Given any constraint network $\mathcal{C}$, we can in low order polynomial time construct a CNF formula $\mathsf{A}(\mathcal{C})$ 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 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**: ${y_1}^F \vee {x_0}^T$, ${y_1}^T \vee {x_0}^F$, ${y_0}^T \vee {x_1}^T \vee {x_0}^T$, ${y_0}^F \vee {x_1}^F$, ${y_0}^F \vee {x_0}^F$, ${x_1}^F \vee {x_0}^F$, ${y_1}^T$, ${y_0}^T$.
- **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

The DPLL Procedure

- **Definition 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(Δ,I) **returns** a partial assignment I, or "unsatisfiable"
  /* Unit Propagation (UP) Rule: */
  $\Delta' :=$ a copy of Δ; $I' := I$
  while Δ' contains a unit clause $C = P^\alpha$ **do**
    extend I' with $[\alpha/P]$, clause–set simplify Δ'
   /* Termination Test: */
   if $\square \in \Delta'$ **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 Δ'; simplify Δ''
  if $I''' :=$ DPLL(Δ'',I'') $\neq$ "unsatisfiable" **then return** I'''
  $I'' := I'$ extended with the other truth value **for** P; $\Delta'' := \Delta'$; simplify Δ''
  return DPLL(Δ'',I'')

- In practice, of course one uses flags etc. instead of "copy".

DPLL: Example (Vanilla1)

- **Example 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. $P \mapsto \mathsf{F}$
     $Q^{\mathsf{T}} \,;\, Q^{\mathsf{F}}$

     3a. UP Rule: $Q \mapsto \mathsf{T}$
     $\square$
     returning "unsatisfiable"

     2b. $P \mapsto \mathsf{T}$
     Q^{F}

     3b. UP Rule: $Q \mapsto \mathsf{F}$
     clause set empty
     returning "$R \mapsto \mathsf{T}, P \mapsto \mathsf{T}, Q \mapsto \mathsf{F}$

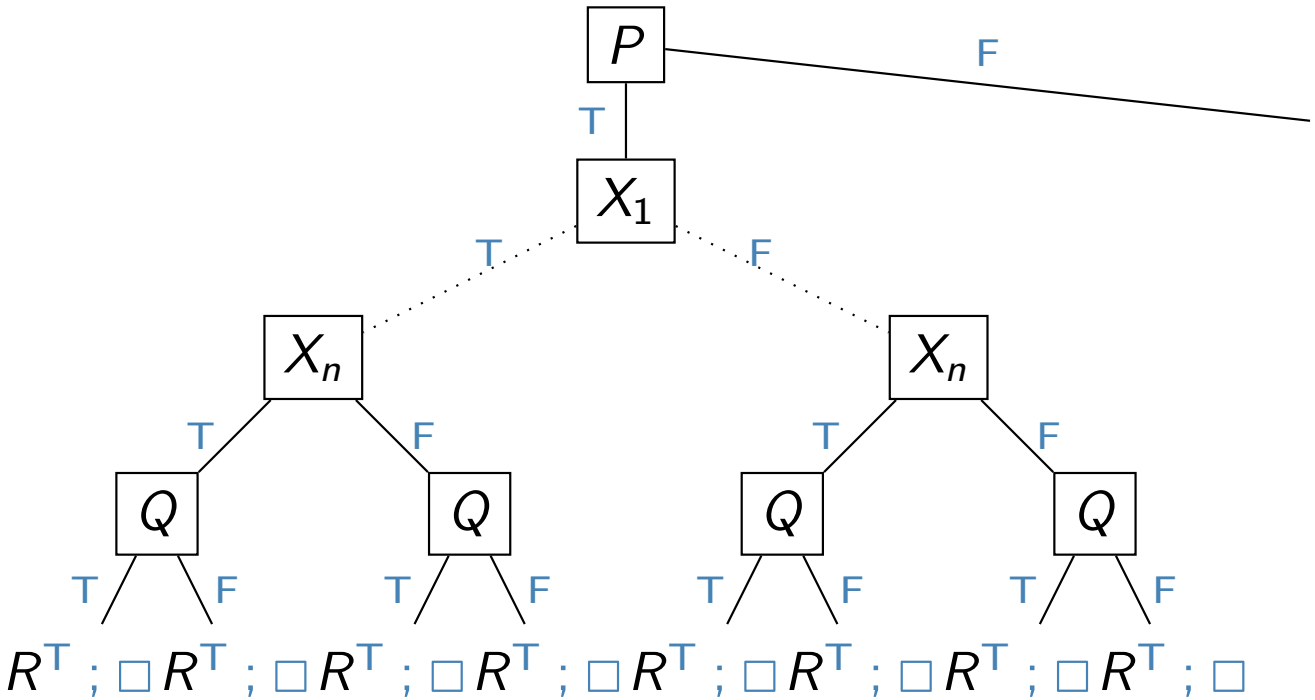
DPLL: Example (Vanilla2)

- **Observation:** Sometimes UP is all we need.
- **Example 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: $P \mapsto \mathsf{T}$
     $\square$

DPLL: Example (Redundance1)

- **Example 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 $\Theta := {X_1}^{\mathsf{T}} \vee \ldots \vee {X_n}^{\mathsf{T}} \,;\, {X_1}^{\mathsf{F}} \vee \ldots \vee {X_n}^{\mathsf{F}}$



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.

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.)

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?

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

UP $\widehat{=}$ Unit Resolution

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

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

- **Definition 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 3.2 (Soundness).** UR *is refutation sound.* *(since resolution is)*
- **Observation 3.3 (Completeness).** UR *is not refutation complete (alone).*
- **Example 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.

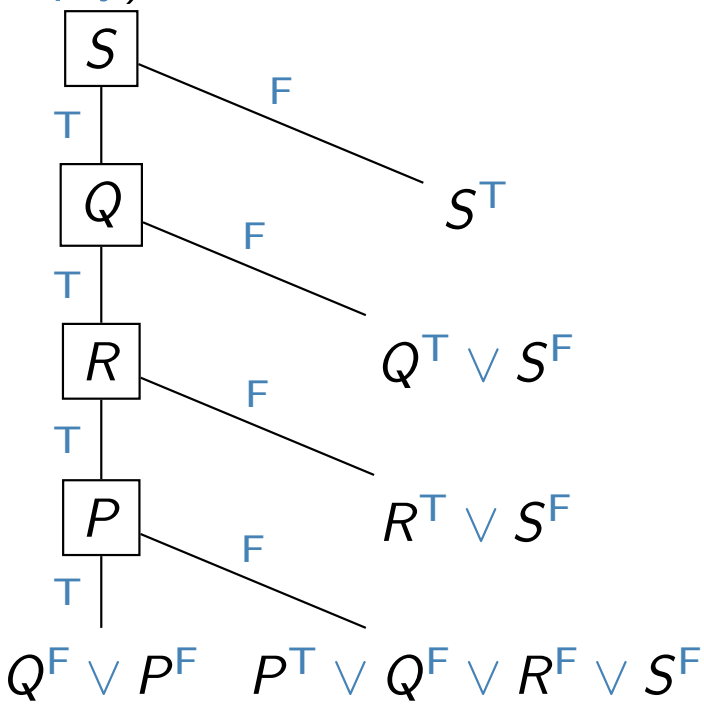
DPLL vs. Resolution

- **Definition 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 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.

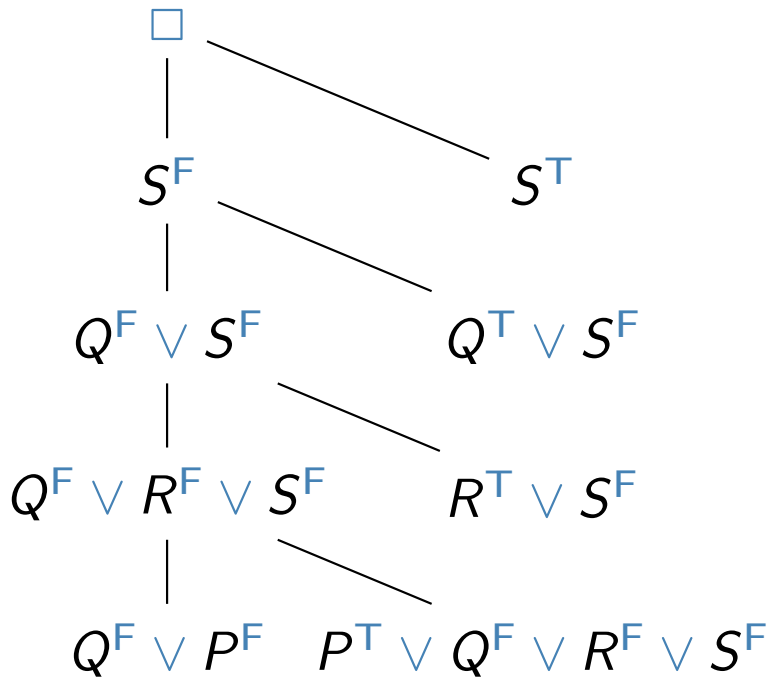
DPLL vs. Resolution: Example (Vanilla2)

- **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 3.7.** We follow the steps in the proof of **??** for
  $\Delta := Q^F \vee P^F ; P^T \vee Q^F \vee R^F \vee S^F ; Q^T \vee S^F ; R^T \vee S^F ; S^T$

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



Resolution proof from that DPLL tree:



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

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 3.8.** 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)

13.4 Conclusion

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 4.1.** The GSAT algorithm **OUTPUT**: a satisfying truth assignment of Δ, if found

```
function GSAT (Δ, MaxFlips MaxTries
  for i :=1 to MaxTries
    I := a randomly−generated truth assignment
    for j :=1 to MaxFlips
    if I satisfies Δ 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

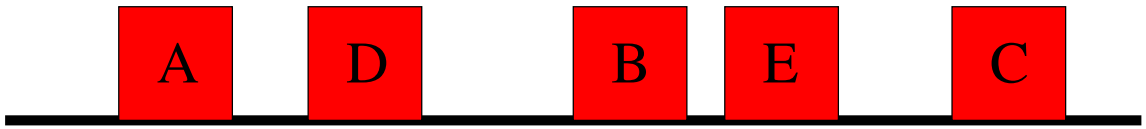
- **Variable/value selection heuristics**: A whole zoo is out there.
- **Implementation techniques**: One of the most intensely researched subjects. Famous "watched literals" technique for UP had huge practical impact.
- **Local search**: In space of all truth value assignments. GSAT (slide 405) had 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)?

Chapter 14
First-Order Predicate Logic

14.1 Motivation: A more Expressive Language

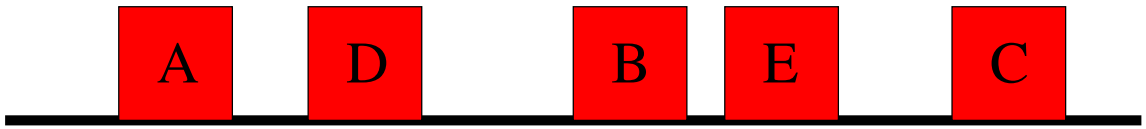
Let's Talk About Blocks, Baby ...

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



Let's Talk About Blocks, Baby ...

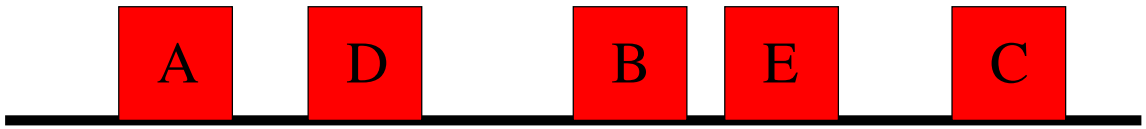
- **Question:** What do you see here?



- **You say:** "All blocks are red"; "All blocks are on the table"; "A is a block".
- **And now:** Say it in propositional logic!

Let's Talk About Blocks, Baby ...

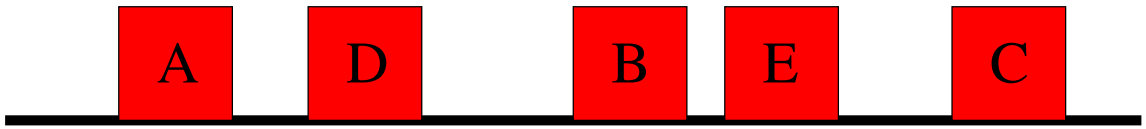
- **Question:** What do you see here?



- **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.

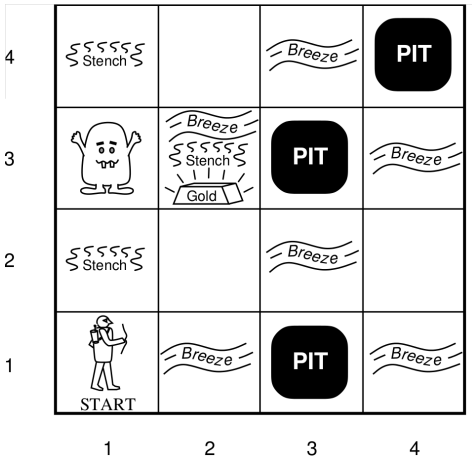
Let's Talk About Blocks, Baby ...

- **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 1.4.** "$\forall x.\mathrm{block}(x) \Rightarrow \mathrm{red}(x)$"; "$\mathrm{block}(\mathrm{A})$"

Let's Talk About the Wumpus Instead?



Percepts: [*Stench*, *Breeze*, *Glitter*, *Bump*, *Scream*]

- Cell adjacent 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*).

- 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 1.5 (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 1.6.**

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

  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:**

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."*

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."*

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.

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.

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)

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.*

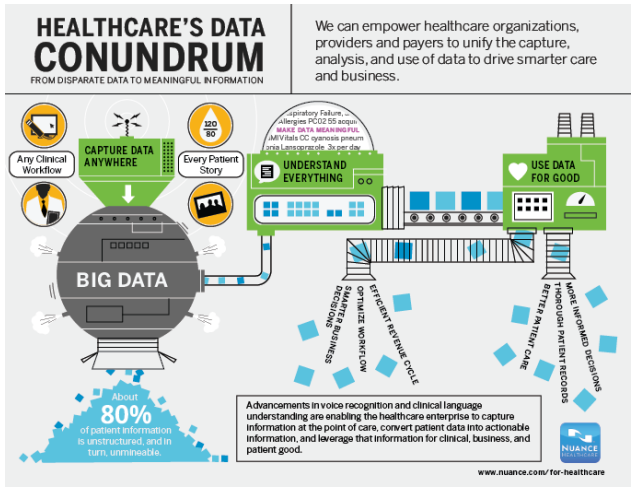
(A Few) Semantic Technology Applications

Web Queries

Jeopardy (IBM Watson)

Context-Aware Apps

Healthcare



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

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,...*)

14.2.1 First-Order Logic: Syntax and Semantics

PL^1 Syntax (Signature and Variables)

- **Definition 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 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 2.3.** We assume a set of individual variables: $\mathcal{V}_\iota := \{X, Y, Z, \ldots\}$. (countably ∞)

PL^1 Syntax (Formulae)

- **Definition 2.4.** Terms: $A \in wff_\iota(\Sigma_1, \mathcal{V}_\iota)$ (denote individuals)
  - $\mathcal{V}_\iota \subseteq wff_\iota(\Sigma_1, \mathcal{V}_\iota)$,
  - if $f \in \Sigma^f_k$ and $A^i \in wff_\iota(\Sigma_1, \mathcal{V}_\iota)$ for $i \leq k$, then $f(A^1, \ldots, A^k) \in wff_\iota(\Sigma_1, \mathcal{V}_\iota)$.
- **Definition 2.5.** First-order propositions: $A \in wff_o(\Sigma_1, \mathcal{V}_\iota)$: (denote truth values)
  - if $p \in \Sigma^p{}_k$ and $A^i \in wff_\iota(\Sigma_1, \mathcal{V}_\iota)$ for $i \leq k$, then $p(A^1, \ldots, A^k) \in wff_o(\Sigma_1, \mathcal{V}_\iota)$,
  - if $A, B \in wff_o(\Sigma_1, \mathcal{V}_\iota)$ and $X \in \mathcal{V}_\iota$, then $T, A \wedge B, \neg A, \forall X.A \in wff_o(\Sigma_1, \mathcal{V}_\iota)$.
    $\forall$ is a binding operator called the universal quantifier.
- **Definition 2.6.** We define the connectives $F, \vee, \Rightarrow, \Leftrightarrow$ via the abbreviations $A \vee B := \neg(\neg A \wedge \neg B)$, $A \Rightarrow B := \neg A \vee B$, $A \Leftrightarrow B := (A \Rightarrow B) \wedge (B \Rightarrow A)$, and $F := \neg T$. We will use them like the primary connectives $\wedge$ and $\neg$
- **Definition 2.7.** We use $\exists X.A$ as an abbreviation for $\neg(\forall X.\neg A)$. $\exists$ is a binding operator called the existential quantifier.
- **Definition 2.8.** Call formulae without connectives or quantifiers atomic else complex.

Alternative Notations for Quantifiers

Here	Elsewhere	
$\forall x.A$	$\bigwedge x.A$	$(x)A$
$\exists x.A$	$\bigvee x.A$	

Free and Bound Variables

- **Definition 2.9.** We call an occurrence of a variable X bound in a formula A (otherwise free), iff it occurs in a sub-formula $\forall X.\mathsf{B}$ of A.
  For a formula A, we will use $\mathrm{BVar}(\mathsf{A})$ (and $\mathrm{free}(\mathsf{A})$) for the set of bound (free) variables of A, i.e. variables that have a free/bound occurrence in A.
- **Definition 2.10.** We define the set $\mathrm{free}(\mathsf{A})$ of free variables of a formula A:

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

- **Definition 2.11.** We call a formula A closed or ground, iff $\mathrm{free}(\mathsf{A})=\emptyset$. 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 2.12.** *Bound variables can be renamed, i.e. any subterm $\forall X.\mathsf{B}$ of a formula A can be replaced by $\mathsf{A}' := (\forall Y.\mathsf{B}')$, where B' arises from B by replacing all $X\in\mathrm{free}(\mathsf{B})$ with a new variable Y that does not occur in A. We call A' an alphabetical variant of A – and the other way around too.*

Semantics of PL^1 (Models)

- **Definition 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 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)
  - $\mathcal{I}\colon \Sigma^f_k \to {\mathcal{D}_\iota}^k \to \mathcal{D}_\iota$ (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 2.15.** A variable assignment $\varphi\colon \mathcal{V}_\iota \to \mathcal{D}_\iota$ maps variables into the domain.
- **Definition 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 $\mathcal{I}$.

Semantics of PL^1 (Evaluation)

- **Definition 2.17.** Given a model $\langle \mathcal{D}, \mathcal{I} \rangle$, the value function $\mathcal{I}_\varphi$ is recursively defined: (two parts: terms & propositions)
  - $\mathcal{I}_\varphi : \mathit{wff}_\iota(\Sigma_1, \mathcal{V}_\iota) \to \mathcal{D}_\iota$ assigns values to terms.
    - $\mathcal{I}_\varphi(X) := \varphi(X)$ and
    - $\mathcal{I}_\varphi(f(A_1, \ldots, A_k)) := \mathcal{I}(f)(\mathcal{I}_\varphi(A_1), \ldots, \mathcal{I}_\varphi(A_k))$
  - $\mathcal{I}_\varphi : \mathit{wff}_o(\Sigma_1, \mathcal{V}_\iota) \to \mathcal{D}_0$ assigns values to formulae:
    - $\mathcal{I}_\varphi(T) = \mathcal{I}(T) = \mathsf{T}$,
    - $\mathcal{I}_\varphi(\neg A) = \mathcal{I}(\neg)(\mathcal{I}_\varphi(A))$
    - $\mathcal{I}_\varphi(A \wedge B) = \mathcal{I}(\wedge)(\mathcal{I}_\varphi(A), \mathcal{I}_\varphi(B))$ (just as in PL^0)
    - $\mathcal{I}_\varphi(p(A_1, \ldots, A_k)) := \mathsf{T}$, iff $\langle \mathcal{I}_\varphi(A_1), \ldots, \mathcal{I}_\varphi(A_k) \rangle \in \mathcal{I}(p)$
    - $\mathcal{I}_\varphi(\forall X.A) := \mathsf{T}$, iff $\mathcal{I}_{\varphi,[a/X]}(A) = \mathsf{T}$ for all $a \in \mathcal{D}_\iota$.
- **Definition 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 $\{(Y,a) \in \varphi \mid Y \neq X\} \cup \{(X,a)\}$: $\varphi,[a/X]$ coincides with φ off X, and gives the result a there.

Semantics Computation: Example

- **Example 2.19.** We define an instance of first-order logic:
  - Signature: Let $\Sigma^f_0 := \{j, m\}$, $\Sigma^f_1 := \{f\}$, 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 $\psi := \varphi,[a/X]$ for $a \in \mathcal{D}_\iota$ we have

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

  But $a \neq J$ for $a = M$, 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

Substitutions on Terms

- **Intuition:** If B is a term and X is a variable, then we denote the result of systematically replacing all occurrences of X in a term A by B with $[B/X](A)$.
- **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 2.20.** Let $wfe(\Sigma,\mathcal{V})$ be an expression language, then we call $\sigma : \mathcal{V} \to wfe(\Sigma,\mathcal{V})$ a substitution, iff the support $\mathrm{supp}(\sigma) := \{X \mid (X,A) \in \sigma, X \neq A\}$ of σ is finite. We denote the empty substitution with ϵ.
- **Definition 2.21 (Substitution Application).** We define substitution application by
  - $\sigma(c) = c$ for $c \in \Sigma$
  - $\sigma(X) = A$, iff $X \in \mathcal{V}$ and $(X,A) \in \sigma$.
  - $\sigma(f(A_1,\ldots,A_n)) = f(\sigma(A_1),\ldots,\sigma(A_n))$,
  - $\sigma(\forall X.A) = \forall X.\sigma_{-X}(A)$. ($\exists$ analogous)
- **Example 2.22.** $[a/x],[f(b)/y],[a/z]$ instantiates $g(x,y,h(z))$ to $g(a,f(b),h(a))$.

Substitution Extension

- **Definition 2.23 (Substitution Extension).** Let σ be a substitution, then we denote the extension of σ with $[A/X]$ by $\sigma,[A/X]$ and define it as $\{(Y,B) \in \sigma \mid Y \neq X\} \cup \{(X,A)\}$: $\sigma,[A/X]$ coincides with σ off X, and gives the result A there.
- **Note:** If σ is a substitution, then $\sigma,[A/X]$ is also a substitution.
- We also need the dual operation: removing a variable from the support:
- **Definition 2.24.** We can discharge a variable X from a substitution σ by setting $\sigma_{-X} := \sigma,[X/X]$.

Substitutions on Propositions

- **Problem:** We want to extend substitutions to propositions, in particular to quantified formulae: What is $\sigma(\forall X.\mathsf{A})$?
- **Idea:** σ should not instantiate bound variables. ($[\mathsf{A}/X](\forall X.\mathsf{B}) = \forall \mathsf{A}.\mathsf{B}'$ ill-formed)
- **Definition 2.25.** $\sigma(\forall X.\mathsf{A}) := (\forall X.\sigma_{-X}(\mathsf{A}))$.
- **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 instantiation, whereas it was free before. **Solution:** Rename away the bound variable X in $\forall X.p(X,Y)$ before applying the substitution.
- **Definition 2.26 (Capture-Avoiding Substitution Application).** Let σ be a substitution, A a formula, and A' an alphabetic variant of A, such that $\mathrm{intro}(\sigma) \cap \mathrm{BVar}(\mathsf{A}) = \emptyset$. Then we define capture-avoiding substitution application via $\sigma(\mathsf{A}) := \sigma(\mathsf{A}')$.

Substitution Value Lemma for Terms

- **Lemma 2.27.** *Let* A *and* B *be terms, then* $\mathcal{I}_\varphi([B/X]A) = \mathcal{I}_\psi(A)$, *where* $\psi = \varphi, [\mathcal{I}_\varphi(B)/X]$.
- *Proof:* by induction on the depth of A:
  1. depth=0 *Then* A *is a variable (say* Y*), or constant, so we have three cases*
     1.1. $A = Y = X$
     1.1.1. then
     $\mathcal{I}_\varphi([B/X](A)) = \mathcal{I}_\varphi([B/X](X)) = \mathcal{I}_\varphi(B) = \psi(X) = \mathcal{I}_\psi(X) = \mathcal{I}_\psi(A)$.
     1.2. $A = Y \neq X$
     1.2.1. then $\mathcal{I}_\varphi([B/X](A)) = \mathcal{I}_\varphi([B/X](Y)) = \mathcal{I}_\varphi(Y) = \varphi(Y) = \psi(Y) = \mathcal{I}_\psi(Y) = \mathcal{I}_\psi(A)$.
     1.3. 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 $A = f(A_1, \ldots, A_n)$ and we have

$$\begin{aligned}\mathcal{I}_\varphi([B/X](A)) &= \mathcal{I}(f)(\mathcal{I}_\varphi([B/X](A_1)), \ldots, \mathcal{I}_\varphi([B/X](A_n))) \\ &= \mathcal{I}(f)(\mathcal{I}_\psi(A_1), \ldots, \mathcal{I}_\psi(A_n)) \\ &= \mathcal{I}_\psi(A).\end{aligned}$$

     by induction hypothesis

Substitution Value Lemma for Propositions

- **Lemma 2.28.** $\mathcal{I}_\varphi([B/X](A)) = \mathcal{I}_\psi(A)$, *where* $\psi = \varphi,[\mathcal{I}_\varphi(B)/X]$.
- *Proof:* by induction on the number n of connectives and quantifiers in A:
  1. $n = 0$
     1.1. then 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 $A = \neg B$ or $A = C \circ D$
     2.1. Here we argue like in the induction step of the term lemma as well.
  3. $n > 0$ and $A = \forall Y.C$ where (WLOG) $X \neq Y$ *(otherwise rename)*
     3.1. then $\mathcal{I}_\psi(A) = \mathcal{I}_\psi(\forall Y.C) = T$, iff $\mathcal{I}_{\psi,[a/Y]}(C) = T$ for all $a \in \mathcal{D}_\iota$.
     3.2. But $\mathcal{I}_{\psi,[a/Y]}(C) = \mathcal{I}_{\varphi,[a/Y]}([B/X](C)) = T$, by induction hypothesis.
     3.3. So $\mathcal{I}_\psi(A) = \mathcal{I}_\varphi(\forall Y.[B/X](C)) = \mathcal{I}_\varphi([B/X](\forall Y.C)) = \mathcal{I}_\varphi([B/X](A))$

14.3 First-Order Natural Deduction

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

- Rules for connectives just as always
- **Definition 3.1 (New Quantifier Rules).** The first-order natural deduction calculus $\mathcal{ND}^1$ extends $\mathcal{ND}_0$ by the following four rules:

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

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

* means that A does not depend on any hypothesis in which X is free.

First-Order Natural Deduction in Sequent Formulation

- Rules for connectives from $\mathcal{ND}^0_\vdash$
- **Definition 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:

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

$$\frac{\Gamma\vdash[B/X](A)}{\Gamma\vdash\exists X.A}\,\mathcal{ND}^1_\vdash\exists I \qquad \frac{\Gamma\vdash\exists X.A \quad \Gamma,[c/X](A)\vdash C \quad c\in\Sigma^{sk}_0 \text{ new}}{\Gamma\vdash C}\,\mathcal{ND}^1_\vdash\exists E$$

Natural Deduction with Equality

- **Definition 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 $\mathcal{I}(=) := \{(x,x) \mid x \in \mathcal{D}_\iota\}$. We call the extended logic first-order logic with equality ($\mathrm{PL}^1_=$)
- We now extend natural deduction as well.
- **Definition 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:

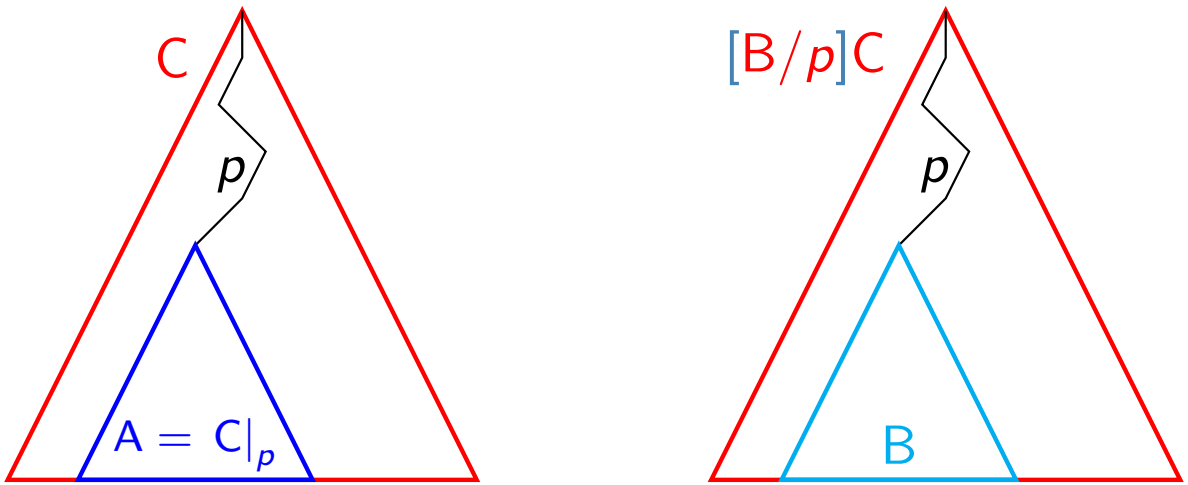
$$\frac{}{A = A}\,{=}I \qquad \frac{A = B \quad C\,[A]_p}{[B/p]C}\,{=}E$$

  where $C\,[A]_p$ if the formula C has a subterm A at position p and $[B/p]C$ is the result of replacing that subterm with B.
- In many ways equivalence behaves like equality, we will use the following rules in $\mathcal{ND}^1$
- **Definition 3.5.** $\Leftrightarrow I$ is derivable and $\Leftrightarrow E$ is admissible in $\mathcal{ND}^1$:

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

Positions in Formulae

- **Idea:** Formulae are (naturally) trees, so we can use tree positions to talk about subformulae
- **Definition 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 A we denote the subexpression at p with $A|_p$.
  We will sometimes write an expression C as $C[A]_p$ to indicate that C the subexpression A at position p.
  If $C[A]_p$ and A is atomic, then we speak of an occurrence of A in C.
- **Definition 3.8.** Let p be a position, then $[A/p]C$ is the expression obtained from C by replacing the subexpression at p by A.
- **Example 3.9 (Schematically).**



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

- We can do real mathematics with $\mathcal{ND}^1_=$:
- **Theorem 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

$\mathcal{ND}^1_=$ 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	$\mathrm{prime}(2) \Rightarrow \#(2q^2) = (\#(q^2)+1)$	$\mathcal{ND}^1_\vdash \forall E^2(1)$

$\mathcal{ND}^1_=$ 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		$\mathrm{irr}(\sqrt{2})$	$\mathcal{ND}_0 \neg E^2(23)$

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.

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?

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 4.3.** 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 $\rightsquigarrow$ proof checkers run in linear time.

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 4.5.** 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 $\rightsquigarrow$ 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?

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 4.7.** 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 $\rightsquigarrow$ 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.

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 4.9.** 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 $\rightsquigarrow$ 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 4.10.** 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.

Chapter 15
Automated Theorem Proving in First-Order Logic

15.1 First-Order Inference with Tableaux

15.1.1 First-Order Tableau Calculi

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 ($\rightsquigarrow$ models).
- **Example 1.1.** 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} $P^{\mathsf{F}} \mid Q^{\mathsf{F}}$ $\bot \mid \bot$	$(P \wedge (Q \vee \neg R) \wedge \neg Q)^{\mathsf{T}}$ $(P \wedge (Q \vee \neg R))^{\mathsf{T}}$ $\neg Q^{\mathsf{T}}$ Q^{F} P^{T} $(Q \vee \neg R)^{\mathsf{T}}$ $Q^{\mathsf{T}} \mid \neg R^{\mathsf{T}}$ $\bot \mid R^{\mathsf{F}}$
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)

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

$$\frac{(A \wedge B)^T}{\begin{array}{c}A^T\\B^T\end{array}}\mathcal{T}_0\wedge \qquad \frac{(A \wedge B)^F}{A^F \mid B^F}\mathcal{T}_0\vee \qquad \frac{\neg A^T}{A^F}\mathcal{T}_0\neg^T \qquad \frac{\neg A^F}{A^T}\mathcal{T}_0\neg^F \qquad \frac{\begin{array}{cc}A^\alpha & \\ A^\beta & \alpha \neq \beta\end{array}}{\bot}\mathcal{T}_0\bot$$

  Use rules exhaustively as long as they contribute new material ($\rightsquigarrow$ termination)
- **Definition 1.3.** We call any tree ($\mid$ introduces branches) produced by the $\mathcal{T}_0$ inference rules from a set Φ of labeled formulae a tableau for Φ.
- **Definition 1.4.** 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 $\Box$ symbol.

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

- **Definition 1.6 ($\mathcal{T}_0$-Theorem/Derivability).** A is a $\mathcal{T}_0$-theorem ($\vdash_{\mathcal{T}_0} A$), iff there is a closed tableau with A^F at the root.
  $\Phi \subseteq \mathit{wff}_0(\mathcal{V}_0)$ derives A in $\mathcal{T}_0$ ($\Phi \vdash_{\mathcal{T}_0} A$), iff there is a closed tableau starting with A^F and Φ^T. The tableau with only a branch of A^F and Φ^T is called initial for $\Phi \vdash_{\mathcal{T}_0} A$.

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

- **Definition 1.8.** The standard tableau calculus ($\mathcal{T}_1$) extends $\mathcal{T}_0$ (propositional tableau calculus) with the following quantifier rules:

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

- **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 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.

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

- **Definition 1.9.** The free variable tableau calculus ($\mathcal{T}_1^f$) extends $\mathcal{T}_0$ (propositional tableau calculus) with the quantifier rules:

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

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

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

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

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

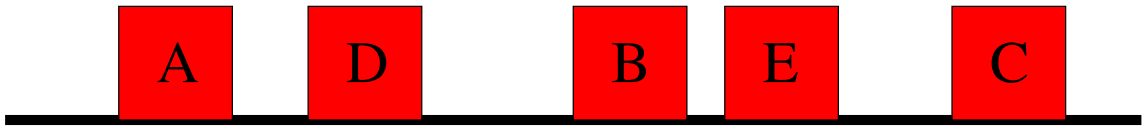
- **Definition 1.10.** Derivable quantifier rules in $\mathcal{T}_1^f$:

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

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

Tableau Reasons about Blocks

- **Example 1.11 (Reasoning about Blocks).** Returing to slide 407



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

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

15.1.2 First-Order Unification

Unification (Definitions)

- **Definition 1.12.** For given terms A and B, unification is the problem of finding a substitution σ, such that $\sigma(\mathsf{A}) = \sigma(\mathsf{B})$.
- **Notation:** We write term pairs as $\mathsf{A} =^? \mathsf{B}$ e.g. $f(X) =^? f(g(Y))$.
- **Definition 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, $\mathsf{U}(\mathsf{A} =^? \mathsf{B}) := \{\sigma \mid \sigma(\mathsf{A}) = \sigma(\mathsf{B})\}$.
- **Idea:** Find representatives in $\mathsf{U}(\mathsf{A} =^? \mathsf{B})$, that generate the set of solutions.
- **Definition 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 1.15.** σ is called a most general unifier (mgu) of A and B, iff it is minimal in $\mathsf{U}(\mathsf{A} =^? \mathsf{B})$ wrt. $\leq[(\mathrm{free}(\mathsf{A}) \cup \mathrm{free}(\mathsf{B}))]$.

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

- **Idea:** Unification is equation solving.
- **Definition 1.16.** We call a formula $A^1 =^? B^1 \wedge \ldots \wedge A^n =^? B^n$ an unification problem iff $A^i, B^i \in \mathit{wff}_\iota(\Sigma_\iota, \mathcal{V}_\iota)$.
- **Note:** We consider unification problems as sets of equations ($\wedge$ is ACI), and equations as two-element multisets ($=^?$ is C).
- **Definition 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 $\mathcal{E}$.

Solved forms and Most General Unifiers

- **Definition 1.18.** We call a pair $A=^?B$ solved in a unification problem $\mathcal{E}$, iff $A = X$, $\mathcal{E} = X=^?A \wedge \mathcal{E}'$, and $X \notin (\mathrm{free}(A) \cup \mathrm{free}(\mathcal{E}'))$. We call an unification problem $\mathcal{E}$ a solved form, iff all its pairs are solved.
- **Lemma 1.19.** *Solved forms are of the form* $X^1=^?B^1 \wedge \ldots \wedge X^n=^?B^n$ *where the* X^i *are distinct and* $X^i \notin \mathrm{free}(B^j)$.
- **Definition 1.20.** Any substitution $\sigma = [B^1/X^1], \ldots, [B^n/X^n]$ induces a solved unification problem $\mathcal{E}_\sigma := (X^1=^?B^1 \wedge \ldots \wedge X^n=^?B^n)$.
- **Lemma 1.21.** *If* $\mathcal{E} = X^1=^?B^1 \wedge \ldots \wedge X^n=^?B^n$ *is a solved form, then* $\mathcal{E}$ *has the unique most general unifier* $\sigma_\mathcal{E} := [B^1/X^1], \ldots, [B^n/X^n]$.
- *Proof:* Let $\theta \in \mathrm{U}(\mathcal{E})$
  1. then $\theta(X^i) = \theta(B^i) = \theta \circ \sigma_\mathcal{E}(X^i)$
  2. and thus $\theta = \theta \circ \sigma_\mathcal{E}[\mathrm{supp}(\sigma)]$.
- **Note:** We can rename the introduced variables in most general unifiers!

Unification Algorithm

- **Definition 1.22.** The inference system $\mathcal{U}$ consists of the following rules:

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

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

- **Lemma 1.23.** $\mathcal{U}$ *is correct:* $\mathcal{E} \vdash_{\mathcal{U}} \mathcal{F}$ *implies* $\mathsf{U}(\mathcal{F}) \subseteq \mathsf{U}(\mathcal{E})$.
- **Lemma 1.24.** $\mathcal{U}$ *is complete:* $\mathcal{E} \vdash_{\mathcal{U}} \mathcal{F}$ *implies* $\mathsf{U}(\mathcal{E}) \subseteq \mathsf{U}(\mathcal{F})$.
- **Lemma 1.25.** $\mathcal{U}$ *is confluent: the order of derivations does not matter.*
- **Corollary 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.

Unification Examples

- **Example 1.27.** Two similar unification problems:

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

Unification (Termination)

- **Definition 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 1.29.** We call a variable X solved in an unification problem $\mathcal{E}$, iff $\mathcal{E}$ contains a solved pair $X =^? A$.
- **Lemma 1.30.** *If $\prec$ is linear/terminating On S, then $\prec^m$ is linear/terminating on $\mathcal{P}(S)$.*
- **Lemma 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.

First-Order Unification is Decidable

- **Definition 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 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 $A=^?B$ in $\mathcal{E} = \mathcal{E} \wedge A=^?B'$.

  we have two cases

  2. $A, B \notin \mathcal{V}_\iota$
     2.1. then $A = f(A^1 \ldots A^n)$ and $B = f(B^1 \ldots B^n)$, and thus $\mathcal{U}\mathrm{dec}$ is applicable
  3. $A = X \in \mathrm{free}(\mathcal{E})$
     3.1. then $\mathcal{U}\mathrm{elim}$ (if $B \neq X$) or $\mathcal{U}\mathrm{triv}$ (if $B = X$) is applicable.
- **Corollary 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

Complexity of Unification

- **Observation:** Naive implementations of unification are exponential in time and space.
- **Example 1.35.** Consider the terms

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

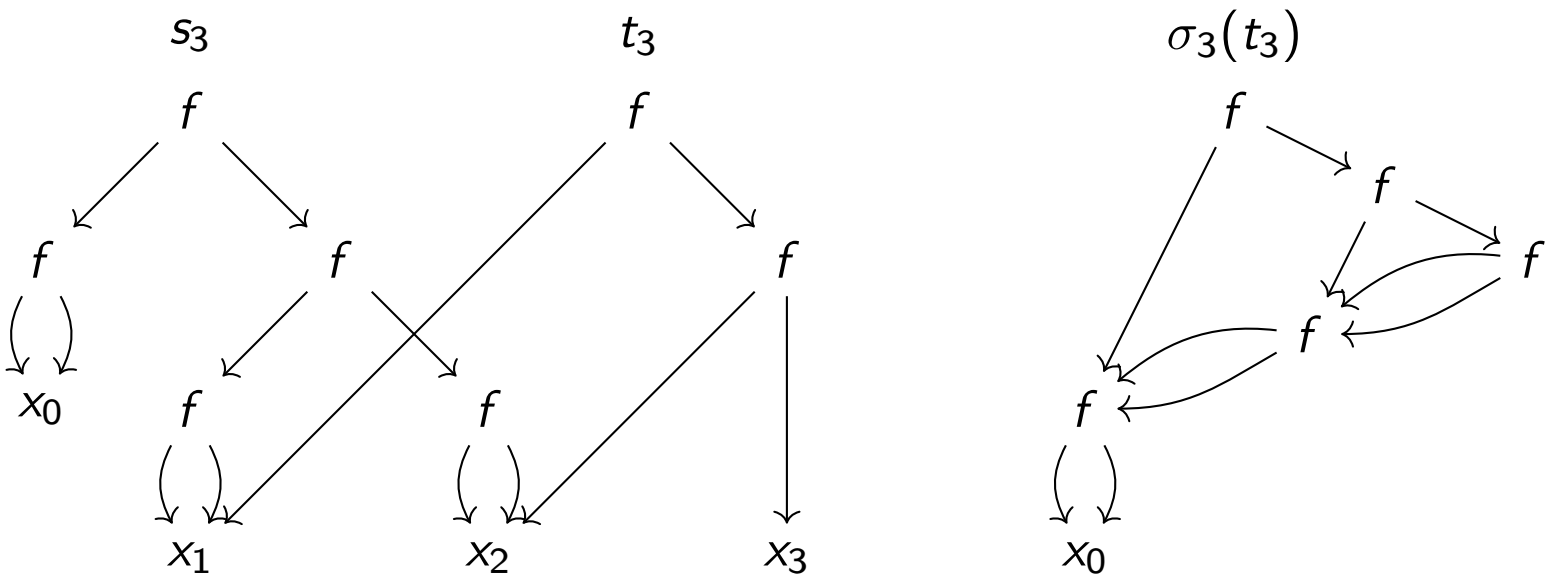
  - The most general unifier of s_n and t_n is

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

  - 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.

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 1.36.** *Terms can be transformed into DAGs in linear time.*
- **Example 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)

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 1.38.** We say that $X^1 =^? \mathsf{B}^1 \wedge \ldots \wedge X^n =^? \mathsf{B}^n$ is a DAG solved form, iff the X^i are distinct and $X^i \notin \mathrm{free}(\mathsf{B}^j)$ for $i \leq j$.
- **Definition 1.39.** The inference system $\mathcal{DU}$ contains rules $\mathcal{U}\mathrm{dec}$ and $\mathcal{U}\mathrm{triv}$ from $\mathcal{U}$ plus the following:

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

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

where $|\mathsf{A}|$ is the number of symbols in A.

- The analysis for $\mathcal{U}$ applies mutatis mutandis.

Unification by DAG-chase

- **Idea:** Extend the Input-DAGs by edges that represent unifiers.
- **Definition 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 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 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

Termination and Multiplicity in Tableaux

- **Recall:** In $\mathcal{T}_0$, all rules only needed to be applied once.
  $\rightsquigarrow$ $\mathcal{T}_0$ terminates and thus induces a decision procedure for PL^0.
- **Observation 1.43.** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f\forall$ only need to be applied once.*

Termination and Multiplicity in Tableaux

- **Recall:** In $\mathcal{T}_0$, all rules only needed to be applied once.
  $\rightsquigarrow$ $\mathcal{T}_0$ terminates and thus induces a decision procedure for PL^0.
- **Observation 1.48.** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f\forall$ only need to be applied once.*
- **Example 1.49.** A tableau proof for $(p(a) \vee p(b)) \Rightarrow (\exists.p())$.

Start, close left branch	use $\mathcal{T}_1^f\forall$ again (right branch)
$((p(a) \vee p(b)) \Rightarrow (\exists.p()))^{\mathsf{F}}$ $(p(a) \vee p(b))^{\mathsf{T}}$ $(\exists x.p(x))^{\mathsf{F}}$ $(\forall x.\neg p(x))^{\mathsf{T}}$ $\neg p(y)^{\mathsf{T}}$ $p(y)^{\mathsf{F}}$ $p(a)^{\mathsf{T}}$ | $p(b)^{\mathsf{T}}$ $\bot : [a/y]$ |	$((p(a) \vee p(b)) \Rightarrow (\exists.p()))^{\mathsf{F}}$ $(p(a) \vee p(b))^{\mathsf{T}}$ $(\exists x.p(x))^{\mathsf{F}}$ $(\forall x.\neg p(x))^{\mathsf{T}}$ $\neg p(a)^{\mathsf{T}}$ $p(a)^{\mathsf{F}}$ $p(a)^{\mathsf{T}}$ | $p(b)^{\mathsf{T}}$ $\bot : [a/y]$ | $\neg p(z)^{\mathsf{T}}$ $p(z)^{\mathsf{F}}$ $\bot : [b/z]$

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 $\mathcal{T}_1^f\forall$.

Termination and Multiplicity in Tableaux

- **Recall:** In $\mathcal{T}_0$, all rules only needed to be applied once.
  $\rightsquigarrow$ $\mathcal{T}_0$ terminates and thus induces a decision procedure for PL^0.
- **Observation 1.53.** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f\forall$ only need to be applied once.*
- **Example 1.54.** A tableau proof for $(p(a) \vee p(b)) \Rightarrow (\exists.p())$.
- **Definition 1.55.** Let $\mathcal{T}$ be a tableau for A, and a positive occurrence of $\forall x.$B in A, then we call the number of applications of $\mathcal{T}_1^f\forall$ to $\forall x.$B its multiplicity.
- **Observation 1.56.** *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 $\forall$.

Termination and Multiplicity in Tableaux

- **Recall:** In $\mathcal{T}_0$, all rules only needed to be applied once.
  $\rightsquigarrow$ $\mathcal{T}_0$ terminates and thus induces a decision procedure for PL^0.
- **Observation 1.58.** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f\forall$ only need to be applied once.*
- **Example 1.59.** A tableau proof for $(p(a) \vee p(b)) \Rightarrow (\exists.p())$.
- **Definition 1.60.** Let $\mathcal{T}$ be a tableau for A, and a positive occurrence of $\forall x.\mathrm{B}$ in A, then we call the number of applications of $\mathcal{T}_1^f\forall$ to $\forall x.\mathrm{B}$ its multiplicity.
- **Observation 1.61.** *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 1.62.** *$\mathcal{T}_1^f$ 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 **??**.

Termination and Multiplicity in Tableaux

- **Recall:** In $\mathcal{T}_0$, all rules only needed to be applied once.
  $\rightsquigarrow$ $\mathcal{T}_0$ terminates and thus induces a decision procedure for PL^0.
- **Observation 1.63.** *All $\mathcal{T}_1^f$ rules except $\mathcal{T}_1^f\forall$ only need to be applied once.*
- **Example 1.64.** A tableau proof for $(p(a) \vee p(b)) \Rightarrow (\exists.p())$.
- **Definition 1.65.** Let $\mathcal{T}$ be a tableau for A, and a positive occurrence of $\forall x.$B in A, then we call the number of applications of $\mathcal{T}_1^f\forall$ to $\forall x.$B its multiplicity.
- **Observation 1.66.** *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 1.67.** *$\mathcal{T}_1^f$ 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.

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 1.68.** Choosing which matters – this tableau does not close!

$$
\begin{array}{c}
(\exists x.(p(a) \wedge p(b) \Rightarrow p()) \wedge (q(b) \Rightarrow q(x)))^{\mathsf{F}} \\
((p(a) \wedge p(b) \Rightarrow p()) \wedge (q(b) \Rightarrow q(y)))^{\mathsf{F}} \\
\hline
(p(a) \wedge p(b) \Rightarrow p())^{\mathsf{F}} \mid (q(b) \Rightarrow q(y))^{\mathsf{F}} \\
p(a)^{\mathsf{T}} \mid q(b)^{\mathsf{T}} \\
p(b)^{\mathsf{T}} \mid q(y)^{\mathsf{F}} \\
p(y)^{\mathsf{F}} \mid \\
\bot : [a/y] \mid
\end{array}
$$

  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)

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

- **Observation 1.69.** *$\mathcal{T}_1^f$ 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 1.70.**
  Let $\mathcal{T}$ be a $\mathcal{T}_1^f$ tableau, then we call a unification problem $\mathcal{E} := A_1 =^? B_1 \wedge \ldots \wedge A_n =^? B_n$ a mating for $\mathcal{T}$, iff ${A_i}^{\mathsf{T}}$ and ${B_i}^{\mathsf{F}}$ 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 ${A_i}^{\mathsf{T}}$ and ${B_i}^{\mathsf{F}}$ for some i.
- **Theorem 1.71.** *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.

15.2 First-Order Resolution

First-Order Resolution (and CNF)

- **Definition 2.1.** The first-order CNF calculus CNF_1 is given by the inference rules of CNF_0 extended by the following quantifier rules:

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

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

  the first-order CNF $CNF_1(\Phi)$ of Φ is the set of all clauses that can be derived from Φ.

- **Definition 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:

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

First-Order CNF – Derived Rules

- **Definition 2.3.** The following inference rules are derivable from the ones above via $(\exists X.A) = \neg(\forall X.\neg A)$:

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

$$\frac{(\exists X.A)^F \vee C \quad Z \notin (\mathrm{free}(A) \cup \mathrm{free}(C))}{([Z/X](A))^F \vee C}$$

15.2.1 Resolution Examples

Col. West, *a Criminal?*

- **Example 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 50ms.
- **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: $\mathrm{ami}(X_1) \wedge \mathrm{weap}(Y_1) \wedge \mathrm{sell}(X_1, Y_1, Z_1) \wedge \mathrm{host}(Z_1) \Rightarrow \mathrm{crook}(X_1)$

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)

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: $\mathrm{mle}(X_2) \wedge \mathrm{own}(\mathrm{NN}, X_2) \Rightarrow \mathrm{sell}(\mathrm{West}, X_2, \mathrm{NN})$

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: $\mathrm{mle}(X_3) \Rightarrow \mathrm{weap}(X_3)$

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: $\mathrm{enmy}(X_4, \mathrm{USA}) \Rightarrow \mathrm{host}(X_4)$

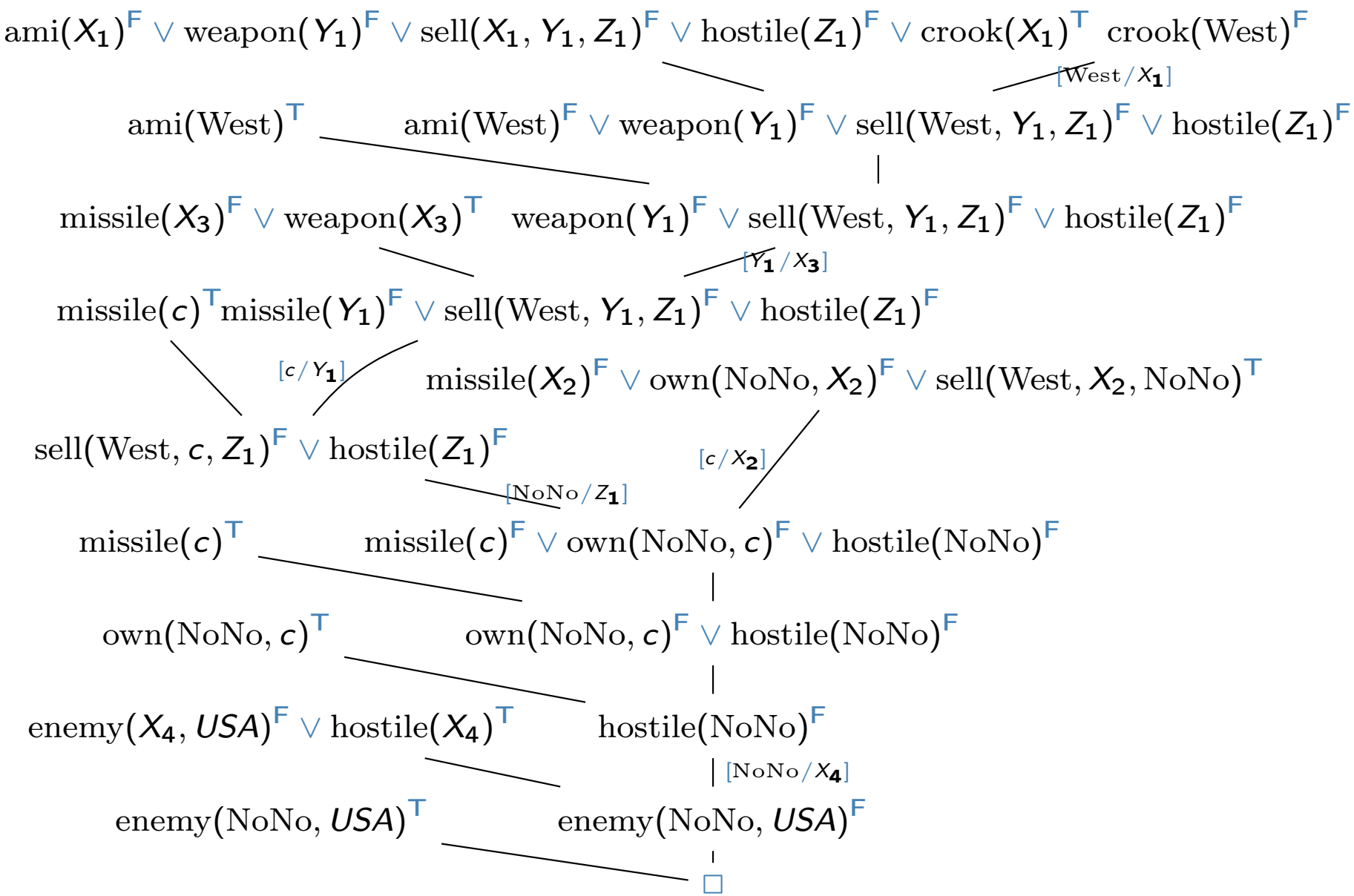
Col. West, *a Criminal?*

- *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: $\mathrm{ami}(\mathrm{West})$

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



Curiosity Killed the Cat?

- **Example 2.5.** From [RN09]

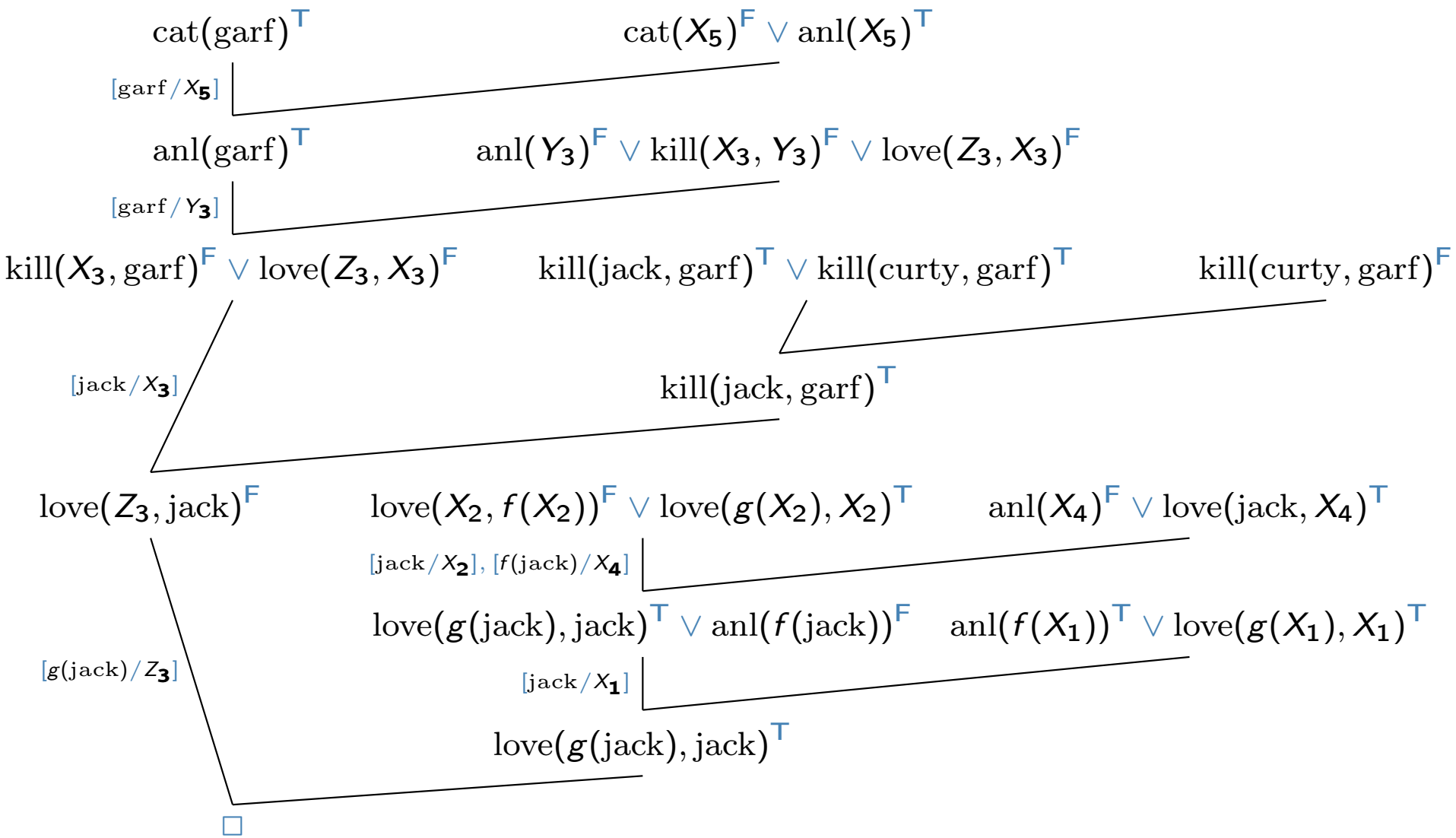
  Everyone who loves all animals is loved by someone.
  Anyone who kills an animal is loved by noone.
  Jack loves all animals.
  Cats are animals.
  Either Jack or curiosity killed the cat (whose name is "Garfield").

  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: $\mathrm{animal}(X_4)^{\mathsf{F}} \vee \mathrm{love}(\mathrm{jack},X_4)^{\mathsf{T}}$
- *Cats are animals*:
  Clause: $\mathrm{cat}(X_5)^{\mathsf{F}} \vee \mathrm{animal}(X_5)^{\mathsf{T}}$
- *Either Jack or curiosity killed the cat (whose name is "Garfield")*:
  Clauses: $\mathrm{kill}(\mathrm{jack},\mathrm{garf})^{\mathsf{T}} \vee \mathrm{kill}(\mathrm{curiosity},\mathrm{garf})^{\mathsf{T}}$ and $\mathrm{cat}(\mathrm{garf})^{\mathsf{T}}$

Curiosity Killed the Cat! PL1 Resolution Proof



15.3 Logic Programming as Resolution Theorem Proving

We know all this already

- Goals, goal sets, rules, and facts are just clauses. (called Horn clauses)
- **Observation 3.1 (Rule).** $H{:-}B_1,\ldots,B_n.$ *corresponds to* ${H^{\mathsf{T}}} \vee {B_1}^{\mathsf{F}} \vee \ldots \vee {B_n}^{\mathsf{F}}$ *(head the only positive literal)*
- **Observation 3.2 (Goal set).** $?{-}\ G_1,\ldots,G_n.$ *corresponds to* ${G_1}^{\mathsf{F}} \vee \ldots \vee {G_n}^{\mathsf{F}}$
- **Observation 3.3 (Fact).** $F.$ *corresponds to the unit clause* $F^{\mathsf{T}}.$
- **Definition 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).
  - $\mathit{next}(\langle G, R_1, \ldots, R_l\rangle) := \langle \sigma(B_1), \ldots, \sigma(B_m), \sigma(R_1), \ldots, \sigma(R_l)\rangle$ if there is a rule $H{:-}B_1,\ldots,\ B_m.$ and a substitution σ with $\sigma(H) = \sigma(G)$.
- **Note:** Backchaining becomes resolution

$$\frac{P^{\mathsf{T}} \vee \mathsf{A} \quad P^{\mathsf{F}} \vee \mathsf{B}}{\mathsf{A} \vee \mathsf{B}}$$

positive, unit-resulting hyperresolution (PURR)

PROLOG (Horn Logic)

- **Definition 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 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 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 3.8 (Query).** ?− fallible(X),greek(X).
- **Answer substitution:** $[\text{sokrates}/X]$

Knowledge Base (Example)

- **Example 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).
```

$$\dfrac{\dfrac{m(c) \quad w(c,4)}{m(c) \wedge w(c,4)}\wedge I \quad \dfrac{m(x) \wedge w(x,4) \Rightarrow car()}{m(c) \wedge w(c,4) \Rightarrow car()}\text{Subst}}{car(c)}\text{MP}$$

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 3.10.** Deduction $\widehat{=}$ knowledge extension
- **Example 3.11.** $\dfrac{rains \Rightarrow wet\_street \quad rains}{wet\_street}\, D$

Three Principal Modes of Inference

- **Definition 3.16.** Deduction $\widehat{=}$ knowledge extension
- **Example 3.17.** $\dfrac{rains \Rightarrow wet\_street \quad rains}{wet\_street}\, D$
- **Definition 3.18.** Abduction $\widehat{=}$ explanation
- **Example 3.19.** $\dfrac{rains \Rightarrow wet\_street \quad wet\_street}{rains}\, A$

Three Principal Modes of Inference

- **Definition 3.22.** Deduction $\widehat{=}$ knowledge extension
- **Example 3.23.** $\dfrac{rains \Rightarrow wet\_street \quad rains}{wet\_street}\, D$
- **Definition 3.24.** Abduction $\widehat{=}$ explanation
- **Example 3.25.** $\dfrac{rains \Rightarrow wet\_street \quad wet\_street}{rains}\, A$
- **Definition 3.26.** Induction $\widehat{=}$ learning general rules from examples
- **Example 3.27.** $\dfrac{wet\_street \quad rains}{rains \Rightarrow wet\_street}\, I$

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.

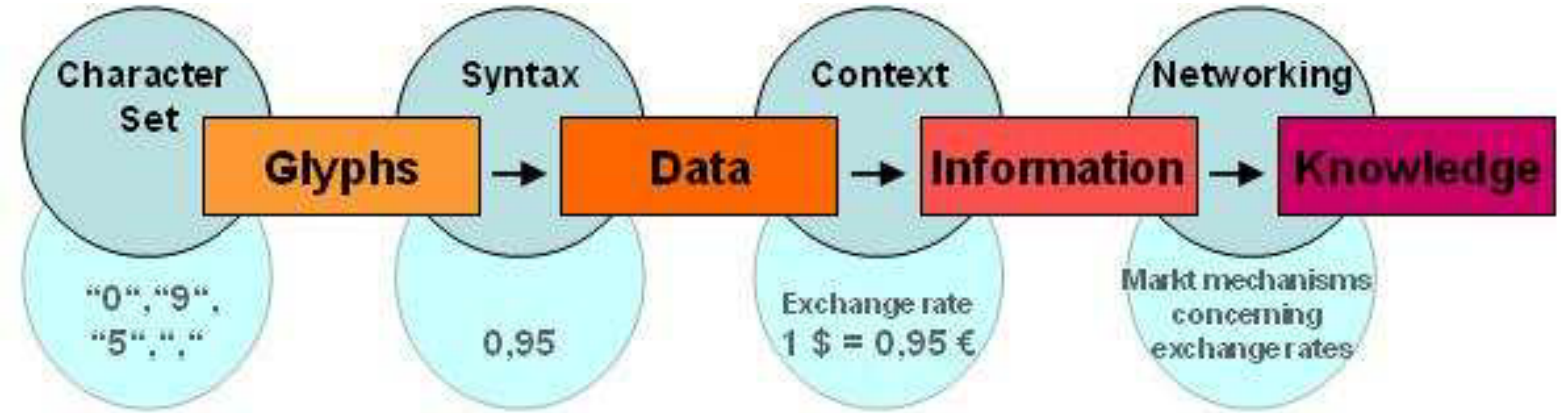
Chapter 16
Knowledge Representation and the Semantic Web

16.1 Introduction to Knowledge Representation

16.1.1 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

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)

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.

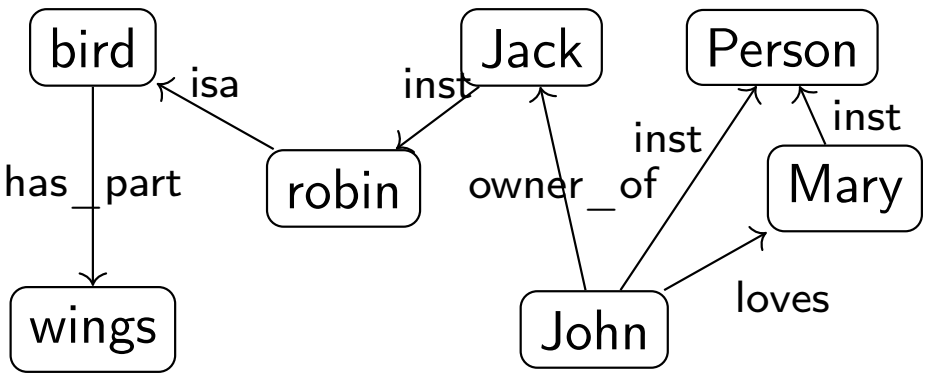
KR Approaches/Evaluation Criteria

- **Definition 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.

16.1.2 Semantic Networks

Semantic Networks [CQ69]

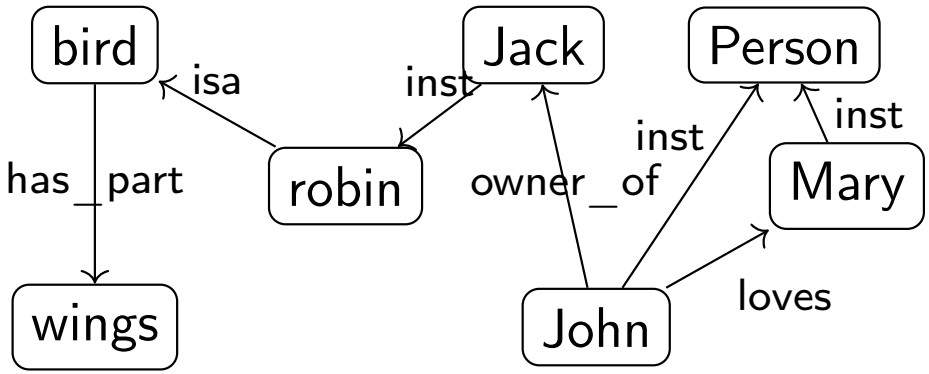
- **Definition 1.2.** A semantic network is a directed graph for representing knowledge:
  - nodes represent objects and concepts (classes of objects) (e.g. John (object) and bird (concept))
  - edges (called links) represent relations between these (isa, father_of, belongs_to)
- **Example 1.3.** A semantic 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.

Deriving Knowledge Implicit in Semantic Networks

- **Observation 1.4.** *There is more knowledge in a semantic network than is explicitly written down.*
- **Example 1.5.** In the network below, we "know" that *robins have wings* and in particular, *Jack has wings*.



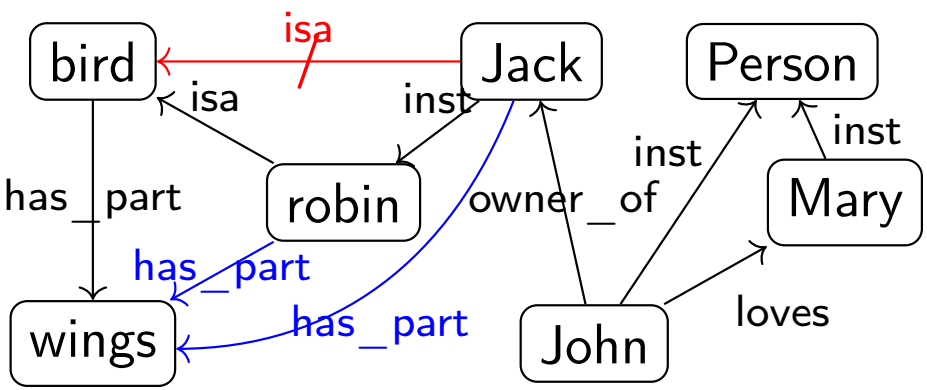
- **Idea:** Links labeled with "isa" and "inst" are special: they propagate properties encoded by other links.
- **Definition 1.6.** We call links labeled by
  - "isa" an inclusion or isa link (inclusion of concepts)
  - "inst" instance or inst link (concept membership)

Deriving Knowledge Semantic Networks

- **Definition 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.

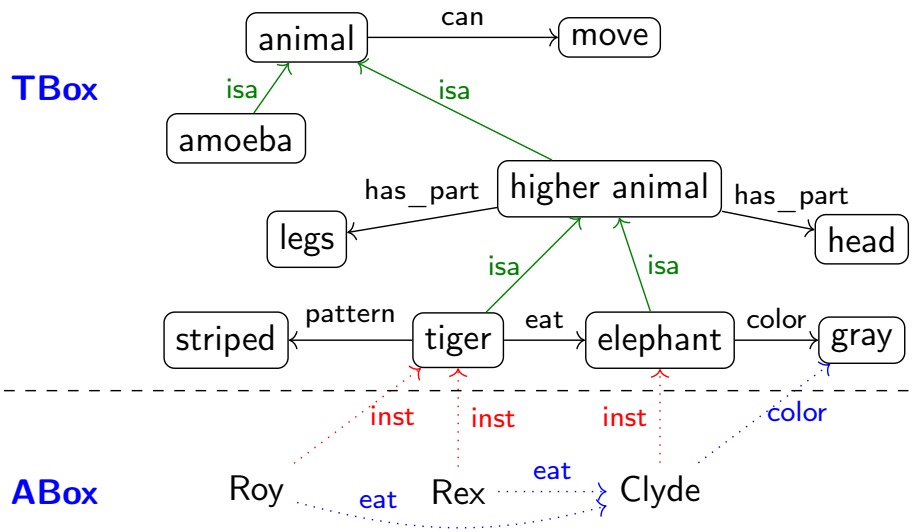
  The process of deriving new concepts and relations from existing ones is called inference and concepts/relations that are only available via inference implicit (in a semantic network).
- **Intuition:** Derived relations represent knowledge that is implicit in the network; they could be added, but usually are not to avoid clutter.
- **Example 1.8.** Derived relations in **??**



- **Slogan:** Get out more knowledge from a semantic networks than you put in.

Terminologies and Assertions

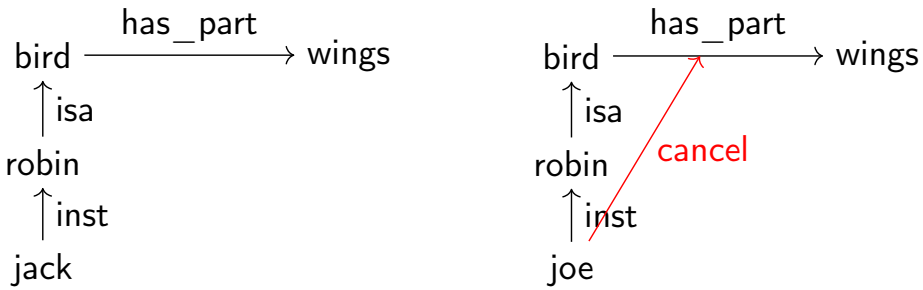
- *Remark 1.9.* We should distinguish concepts from objects.
- **Definition 1.10.** We call the subgraph of a semantic network N spanned by the isa links and relations between concepts the terminology (or TBox, or the famous Isa Hierarchy) and the subgraph spanned by the inst links and relations between objects, the assertions (together the ABox) of N.
- **Example 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*).

Limitations of Semantic Networks

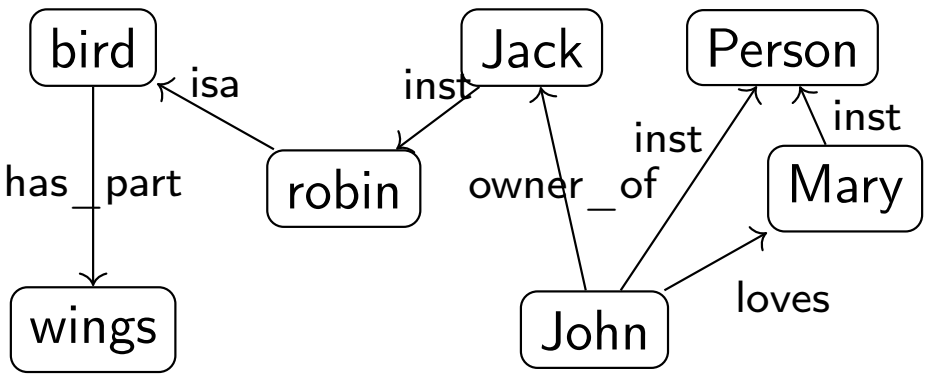
- What is the meaning of a link?
  - link labels are very suggestive (misleading for humans)
  - meaning of link types defined in the process model (no denotational semantics)
- **Problem:** No distinction of optional and defining traits!
- **Example 1.12.** Consider a robin that has lost its wings in an accident:



"Cancel-links" have been proposed, but their status and process model are debatable.

Another Notation for Semantic Networks

- **Definition 1.13.** Function/argument notation for semantic networks
  - interprets nodes as arguments (reification to individuals)
  - interprets links as functions (predicates actually)
- **Example 1.14.**

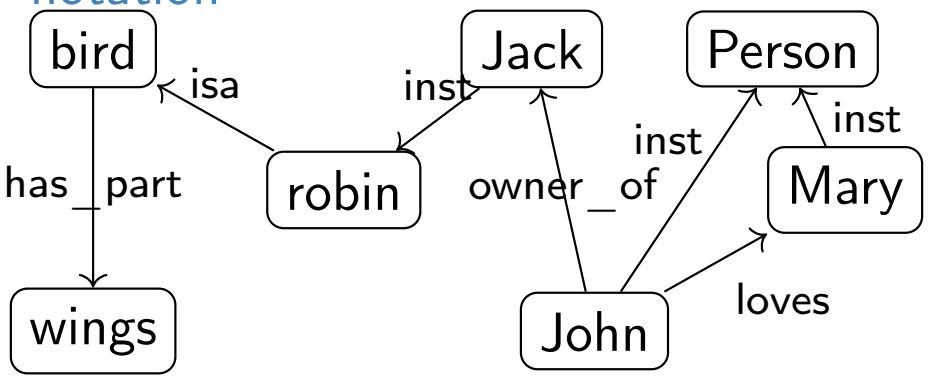


isa(robin,bird)
haspart(bird,wings)
inst(Jack,robin)
owner_of(John, robin)
loves(John,Mary)

- **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)

A Denotational Semantics for Semantic Networks

- **Observation:** If we handle isa and inst links specially in function/argument notation



$\mathrm{robin} \subseteq \mathrm{bird}$
haspart(bird,wings)
$\mathrm{Jack} \in \mathrm{robin}$
owner_of(John, Jack)
loves(John,Mary)

it looks like first-order logic, if we take

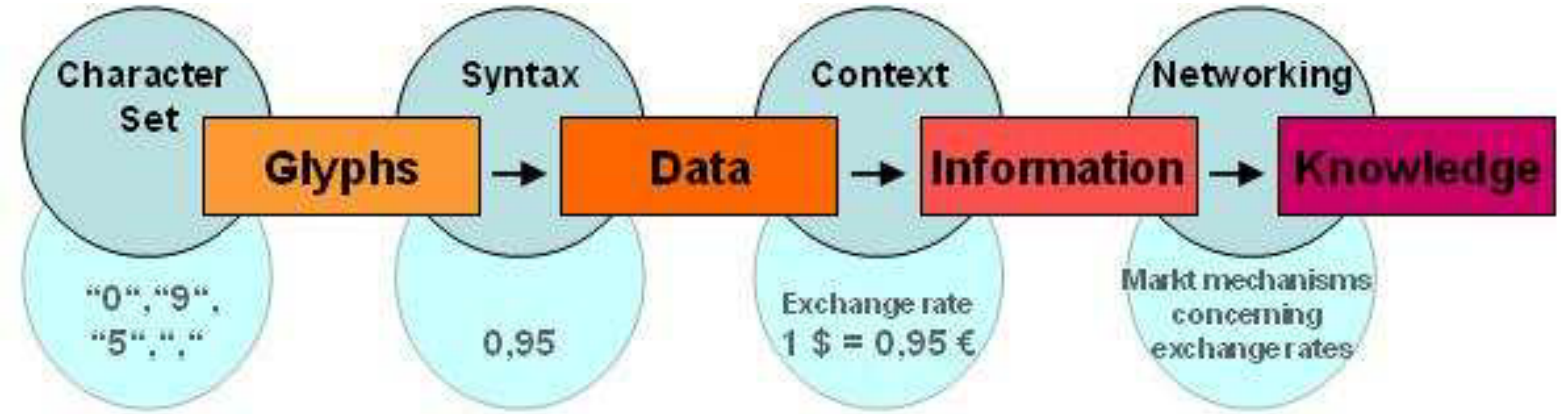
- $a \in S$ to mean $S(a)$ for an object a and a concept S.
- $A \subseteq B$ to mean $\forall X.A(X) \Rightarrow B(X)$ and concepts A and B
- $R(A, B)$ to mean $\forall X.A(X) \Rightarrow (\exists Y.B(Y) \wedge R(X, Y))$ for a relation R.

- **Idea:** Take first-order deduction as process model (gives inheritance for free)

16.1.3 The Semantic Web

The Semantic Web

- **Definition 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 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 $\rightsquigarrow$ semantic web technology.

What is the Information a User sees?

- **Example 1.17.** Take the following web-site with a conference announcement

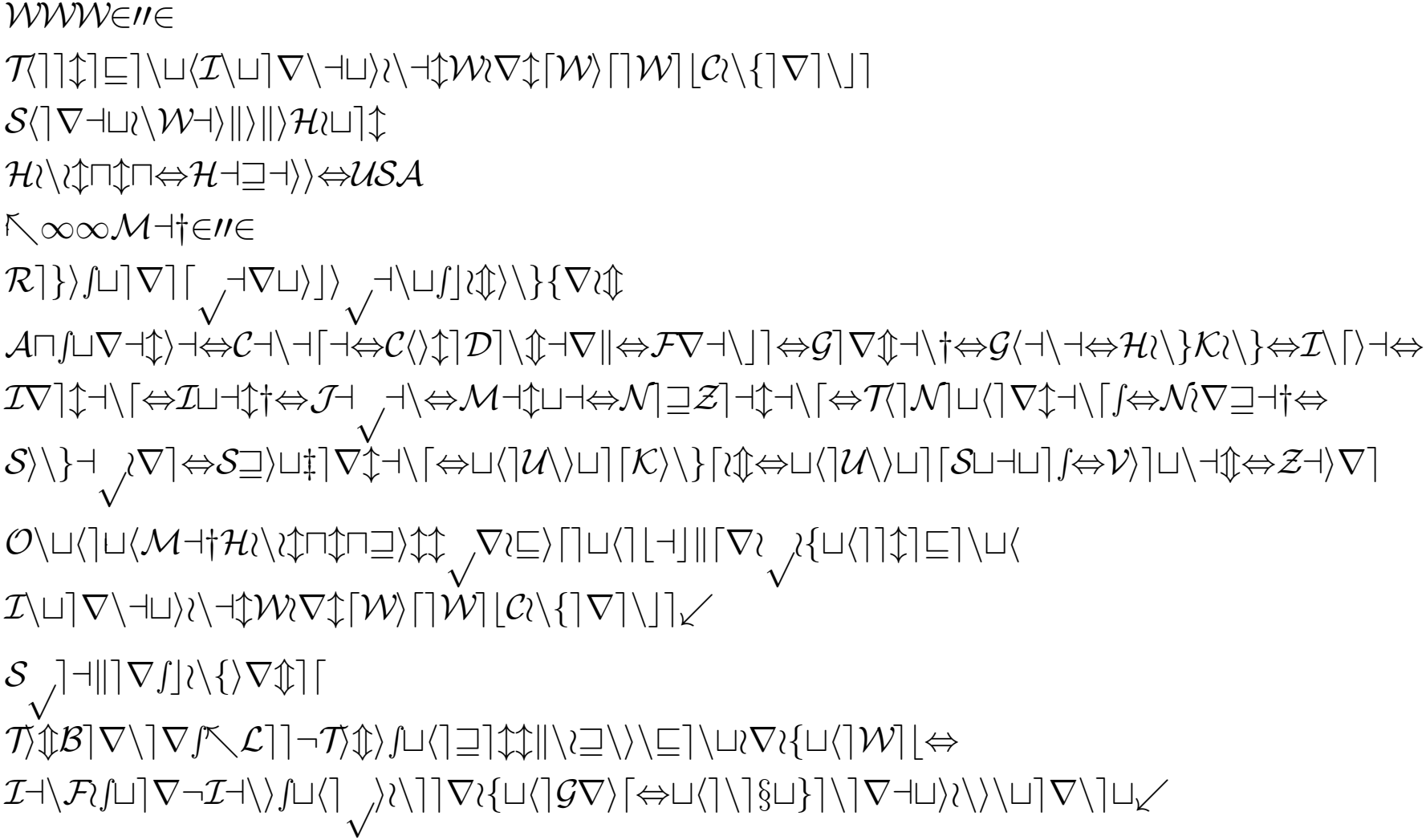
  WWW2002
  The eleventh International World Wide Web Conference
  Sheraton Waikiki Hotel
  Honolulu, Hawaii, USA
  7-11 May 2002
  Registered participants coming from
  Australia, Canada, Chile Denmark, France, Germany, Ghana, Hong Kong, India, Ireland, Italy, Japan, Malta, New Zealand, The Netherlands, 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.

What the machine sees

- **Example 1.18.** Here is what the machine "sees" from the conference announcement:



Solution: XML markup with "meaningful" Tags

- **Example 1.19.** Let's annotate (parts of) the meaning via XML markup

<title>WWW∈//∈
T⟨⌉⌉↕⌉⊑⌉\⊔⟨I\⊔⌉∇\⊣⊔⟩ι\⊣↕Wι∇↕⌈W⟩⌈⌉W⌉⌊Cι\{⌉∇⌉\⌋⌉</title>
<place>S⟨⌉∇⊣⊔ι\W⊣⟩‖⟩‖⟩Hι⊔⌉↕Hι\ι↕⊓↕⊓⇔H⊣⊒⊣⟩⟩⇔USA</place>
<date>↖∞∞M⊣†∈//∈</date>
<participants>R⌉}⟩∫⊔⌉∇⌉⌈√⊣∇⊔⟩⌋⟩√⊣\⊔∫⌋ι⇳⟩\}{∇ι⇳
A⊓∫⊔∇⊣↕⟩⊣⇔C⊣\⊣⌈⊣⇔C⟨⟩↕⌉D⌉\⇳⊣∇‖⇔F∇⊣\⌋⌉⇔G⌉∇⇳⊣\†⇔G⟨⊣\⊣⇔Hι\}Kι\}⇔I\⌈⟩⊣⇔
I∇⌉↕⊣\⌈⇔I⊔⊣↕†⇔J⊣√⊣\⇔M⊣↕⊔⊣⇔N⌉⊒Z⌉⊣↕⊣\⌈⇔T⟨⌉N⌉⊔⟨⌉∇↕⊣\⌈∫⇔Nι∇⊒⊣†⇔
S⟩\}⊣√ι∇⌉⇔S⊒⟩⊔‡⌉∇↕⊣\⌈⇔⊔⟨⌉U\⟩⊔⌉⌈K⟩\}⌈ι⇳⇔⊔⟨⌉U\⟩⊔⌉⌈S⊔⊣⊔⌉∫⇔V⟩⌉⊔\⊣⇳⇔Z⊣⟩∇⌉
</participants>
<introduction>O\⊔⟨⌉⊔⟨M⊣†Hι\ι↕⊓↕⊓⊒⟩↕↕√∇ι⊑⟩⌈⌉⊔⟨⌉⌊⊣⌋‖⌈∇ι√ι{⊔⟨⌉⌉↕⌉⊑⌉\⊔⟨I\↖
⊔⌉∇\⊣⊔⟩ι\⊣↕Wι∇↕⌈W⟩⌈⌉W⌉⌊Cι\{⌉∇⌉\⌋⌉↙</introduction>
<program>S√⌉⊣‖⌉∇∫⌋ι\{⟩∇⇳⌉⌈
<speaker>T⇳⇕B⌉∇\⌉∇∫↖L⌉⌉¬T⇳⇕⟩∫⊔⟨⌉⊒⌉↕↕‖\ι⊒\⟩\⊑⌉\⊔ι∇ι{⊔⟨⌉W⌉⌊</speaker>
<speaker>I⊣\Fι∫⊔⌉∇¬I⊣\⟩∫⊔⟨⌉√⟩ι\⌉⌉∇ι{⊔⟨⌉G∇⟩⌈⇔⊔⟨⌉\⌉§⊔}⌉\⌉∇⊣⊔⟩ι\⟩\⊔⌉∇↖
\⌉⊔<speaker>
</program>

What can we do with this?

- **Example 1.20.** Consider the following fragments:

  $\Re\sqcup\rangle\sqcup\updownarrow\rceil\top\mathcal{W}\mathcal{W}\mathcal{W}\in\!\!/\!\!/\in$

  $\mathcal{T}\langle\rceil\rceil\updownarrow\rceil\sqsubseteq\rceil\backslash\sqcup\langle\mathcal{I}\backslash\sqcup\rceil\nabla\backslash\dashv\sqcup\rangle\wr\backslash\dashv\updownarrow\mathcal{W}\wr\nabla\updownarrow\lceil\mathcal{W}\rangle\lceil\rceil\mathcal{W}\rceil\lfloor\mathcal{C}\wr\backslash\{\rceil\nabla\rceil\backslash\rfloor\rceil\Re\propto\sqcup\rangle\sqcup\updownarrow\rceil\top$

  $\Re\sqrt{\updownarrow}\dashv\rfloor\rceil\top\mathcal{S}\langle\rceil\nabla\dashv\sqcup\wr\backslash\mathcal{W}\dashv\rangle\|\rangle\|\rangle\mathcal{H}\wr\sqcup\rceil\updownarrow\mathcal{H}\wr\backslash\wr\updownarrow\sqcap\updownarrow\sqcap\Leftrightarrow\mathcal{H}\dashv\sqsupseteq\dashv\rangle\rangle\Leftrightarrow\mathcal{U}\mathcal{S}\mathcal{A}\Re\propto\sqrt{\updownarrow}\dashv\rfloor\rceil\top$

  $\Re\lceil\dashv\sqcup\rceil\top\nwarrow\infty\infty\mathcal{M}\dashv\dagger\in\!\!/\!\!/\in\Re\propto\lceil\dashv\sqcup\rceil\top$

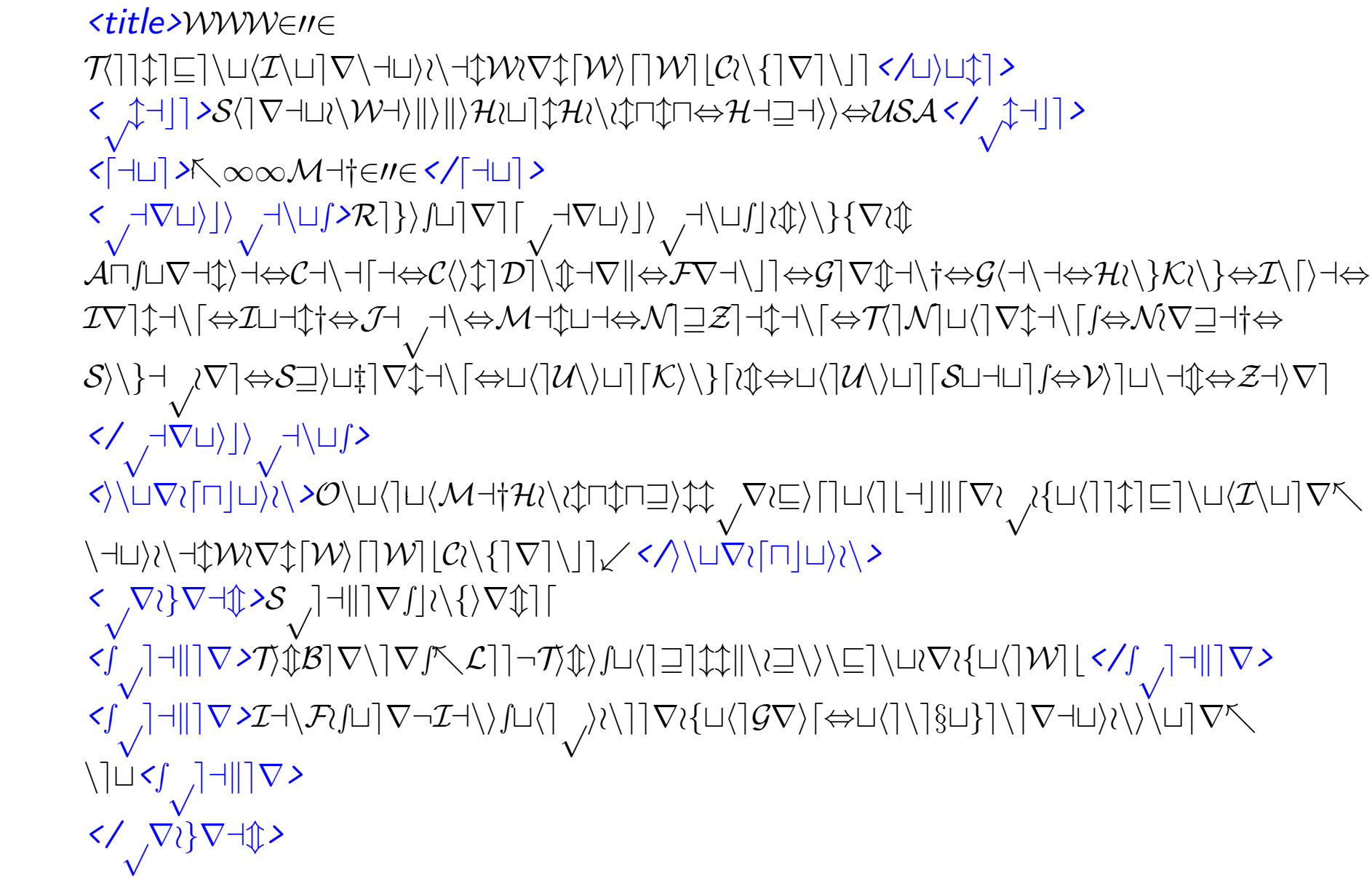
  Given the markup above, a machine agent can

  - parse $\infty\infty\mathcal{M}\dashv\dagger\in\!\!/\!\!/\in$ as the date May 7 11 2002 and add this to the user's calendar,
  - parse $\mathcal{S}\langle\rceil\nabla\dashv\sqcup\wr\backslash\mathcal{W}\dashv\rangle\|\rangle\|\rangle\mathcal{H}\wr\sqcup\rceil\updownarrow\mathcal{H}\wr\backslash\wr\updownarrow\sqcap\updownarrow\sqcap\Leftrightarrow\mathcal{H}\dashv\sqsupseteq\dashv\rangle\rangle\Leftrightarrow\mathcal{U}\mathcal{S}\mathcal{A}$ as a destination and find flights.

- **But:** do not be deceived by your ability to understand English!

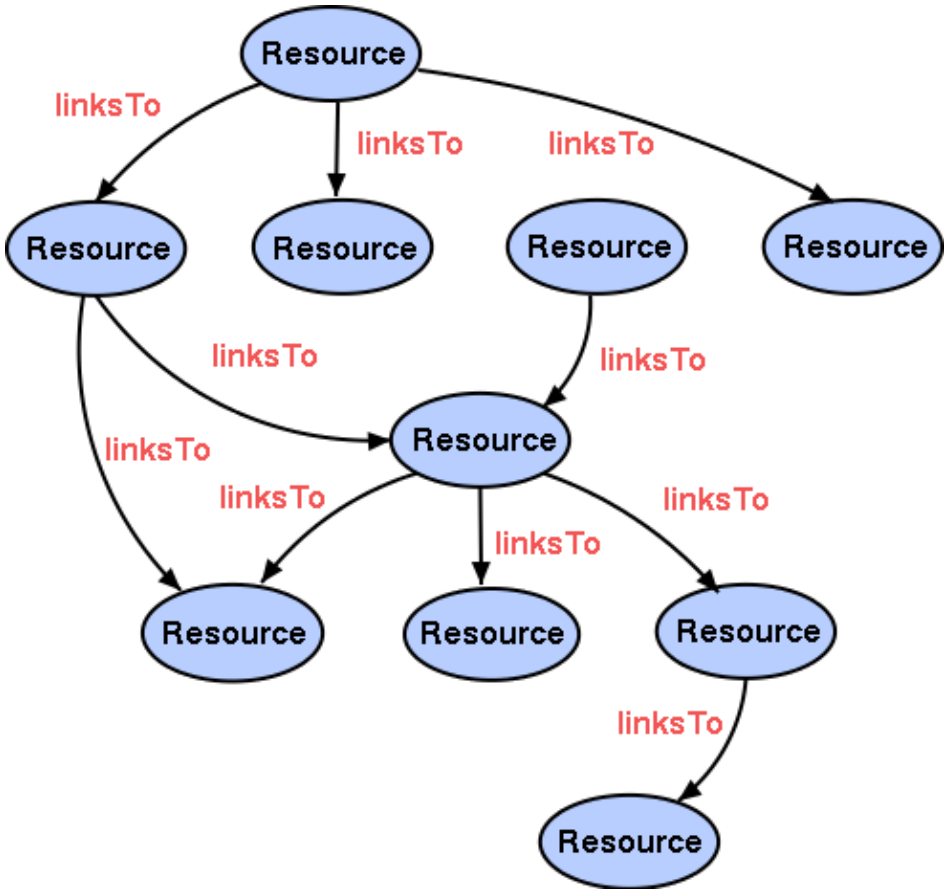
What the machine sees of the XML

- **Example 1.21.** Here is what the machine sees of the XML



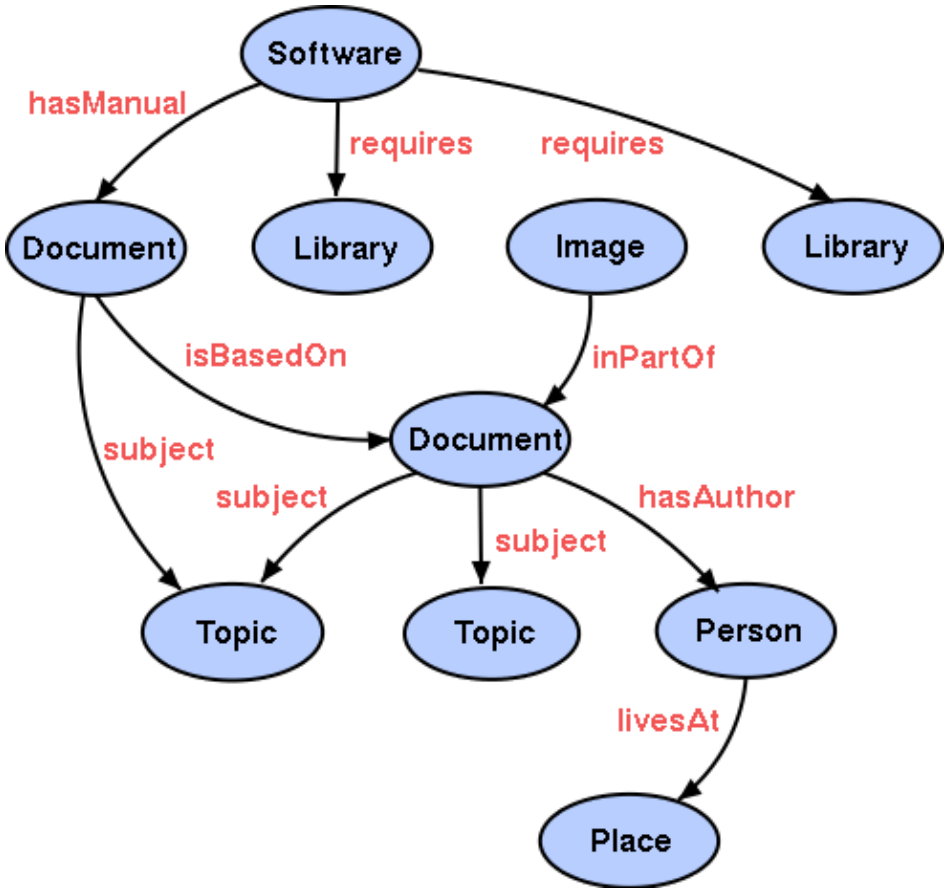
The Current Web

- **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.



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.



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

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

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+13] 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

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

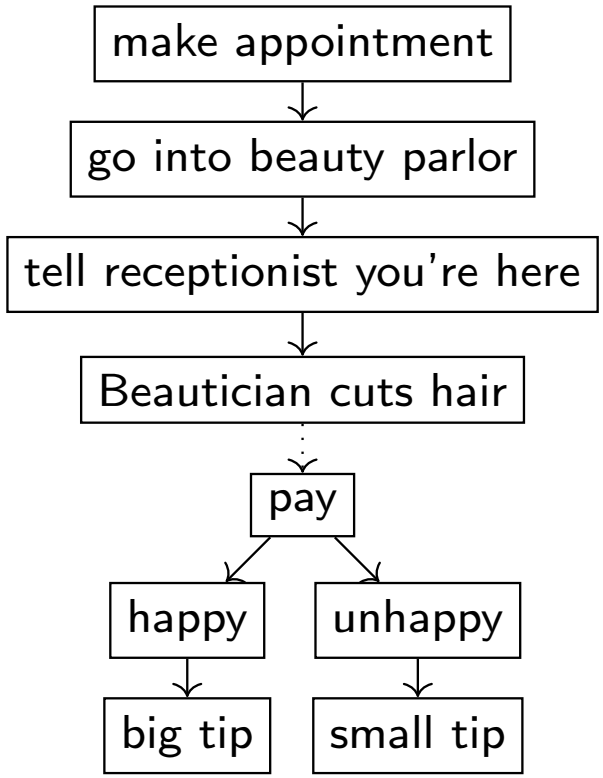
- **Definition 1.22.** Frames (group everything around the object)

```
(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 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 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

16.2 Logic-Based Knowledge Representation

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)

16.2.1 Propositional Logic as a Set Description Language

Propositional Logic as Set Description Language

- **Idea:** Use propositional logic as a set description language: (variant syntax/semantics)
- **Definition 2.1.** Let $\mathrm{PL}^0_{\mathrm{DL}}$ be given by the following grammar for the $\mathrm{PL}^0_{\mathrm{DL}}$ concepts. (formulae)

$$\mathcal{L} ::= C \mid \top \mid \bot \mid \overline{\mathcal{L}} \mid \mathcal{L} \sqcap \mathcal{L} \mid \mathcal{L} \sqcup \mathcal{L} \mid \mathcal{L} \sqsubseteq \mathcal{L} \mid \mathcal{L} \equiv \mathcal{L}$$

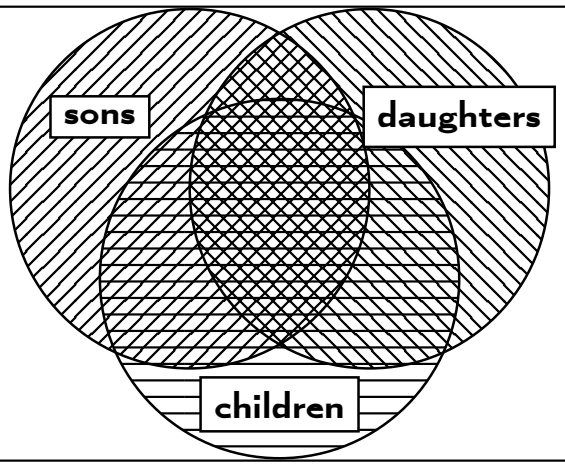
  i.e. $\mathrm{PL}^0_{\mathrm{DL}}$ formed from
  - atomic formulae ($\widehat{=}$ propositional variables)
  - concept intersection ($\sqcap$) ($\widehat{=}$ conjunction $\wedge$)
  - concept complement ($\overline{\cdot}$) ($\widehat{=}$ negation $\neg$)
  - concept union ($\sqcup$), subsumption ($\sqsubseteq$), and equivalence ($\equiv$) defined from these.($\widehat{=}$ $\vee$, $\Rightarrow$, and $\Leftrightarrow$)
- **Definition 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 $\varphi(P) \subseteq \mathcal{D}$).
  - $[\![\mathsf{A} \sqcap \mathsf{B}]\!] := [\![\mathsf{A}]\!] \cap [\![\mathsf{B}]\!]$ and $[\![\overline{\mathsf{A}}]\!] := \mathcal{D} \setminus [\![\mathsf{A}]\!]$ ...

  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.

Concept Axioms

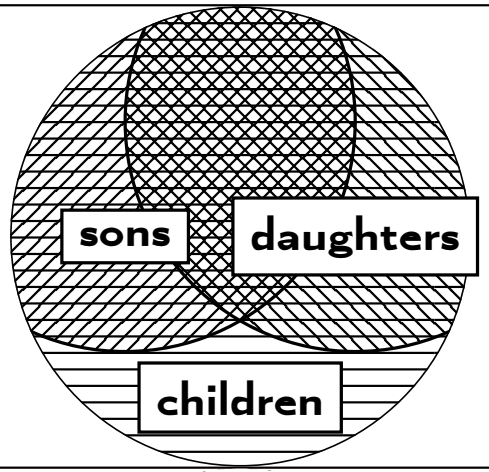
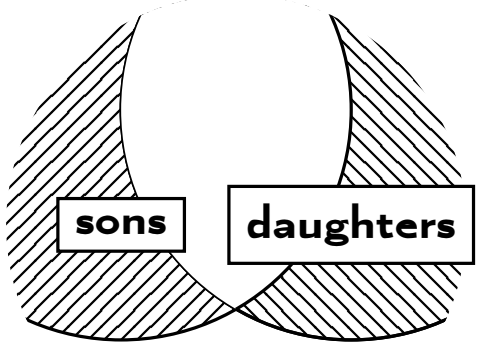
- **Observation:** Set-theoretic semantics of 'true' and 'false' ($\top := \varphi \sqcup \overline{\varphi}$ $\bot := \varphi \sqcap \overline{\varphi}$)

  $[\![\top]\!] = [\![p]\!] \cup [\![\overline{p}]\!] = [\![p]\!] \cup \mathcal{D} \setminus [\![p]\!] = \mathcal{D}$ Analogously: $[\![\bot]\!] = \emptyset$
- **Idea:** Use logical axioms to describe the world (Axioms restrict the class of admissible domain structures)
- **Definition 2.3.** A concept axiom is a $\mathrm{PL}^0_{\mathrm{DL}}$ formula A that is assumed to be true in the world.
- **Definition 2.4 (Set-Theoretic Semantics of Axioms).** A is true in domain of discourse $\mathcal{D}$ iff $[\![\mathsf{A}]\!] = \mathcal{D}$.
- **Example 2.5.** A world with three concepts and no concept axioms

concepts	Set Semantics
child daughter son	

Effects of Axioms to Siblings

- **Example 2.6.** We can use concept axioms to describe the world from **??**.

Axioms	Semantics
$\text{son} \sqsubseteq \text{child}$ iff $[\![\overline{\text{son}}]\!] \cup [\![\text{child}]\!] = \mathcal{D}$ iff $[\![\text{son}]\!] \subseteq [\![\text{child}]\!]$ $\text{daughter} \sqsubseteq \text{child}$ iff $[\![\overline{\overline{\text{daughter}}}]\!] \cup [\![\text{child}]\!] = \mathcal{D}$ iff $[\![\text{daughter}]\!] \subseteq [\![\text{child}]\!]$	
$\overline{\text{son} \sqcap \text{daughter}}$ $\text{child} \sqsubseteq \text{son} \sqcup \text{daughter}$	

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	$\overline{\overline{\varphi}} = \varphi$	

Set-Theoretic Semantics and Predicate Logic

- **Definition 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
$\overline{p}^{fo(x)} := p(x)$	
$\overline{\overline{\mathsf{A}}}^{fo(x)} := \neg\overline{\mathsf{A}}^{fo(x)}$	
$\overline{\mathsf{A}\sqcap\mathsf{B}}^{fo(x)} := \overline{\mathsf{A}}^{fo(x)} \wedge \overline{\mathsf{B}}^{fo(x)}$	$\wedge$ vs. $\sqcap$
$\overline{\mathsf{A}\sqcup\mathsf{B}}^{fo(x)} := \overline{\mathsf{A}}^{fo(x)} \vee \overline{\mathsf{B}}^{fo(x)}$	$\vee$ vs. $\sqcup$
$\overline{\mathsf{A}\sqsubseteq\mathsf{B}}^{fo(x)} := \overline{\mathsf{A}}^{fo(x)} \Rightarrow \overline{\mathsf{B}}^{fo(x)}$	$\Rightarrow$ vs. $\sqsubseteq$
$\overline{\mathsf{A}=\mathsf{B}}^{fo(x)} := \overline{\mathsf{A}}^{fo(x)} \Leftrightarrow \overline{\mathsf{B}}^{fo(x)}$	$\Leftrightarrow$ vs. $=$
$\overline{\mathsf{A}}^{fo} := (\forall x.\overline{\mathsf{A}}^{fo(x)})$	for formulae

Translation Examples

- **Example 2.8.** We translate the concept axioms from **??** to fortify our intuition:

$$\overline{\mathsf{son} \sqsubseteq \mathsf{child}}^{fo} = \forall x.\mathsf{son}(x) \Rightarrow \mathsf{child}(x)$$

$$\overline{\mathsf{daughter} \sqsubseteq \mathsf{child}}^{fo} = \forall x.\mathsf{daughter}(x) \Rightarrow \mathsf{child}(x)$$

$$\overline{\overline{\mathsf{son} \sqcap \mathsf{daughter}}}^{fo} = \forall x.\overline{\mathsf{son}(x) \wedge \mathsf{daughter}(x)}$$

$$\overline{\mathsf{child} \sqsubseteq \mathsf{son} \sqcup \mathsf{daughter}}^{fo} = \forall x.\mathsf{child}(x) \Rightarrow (\mathsf{son}(x) \vee \mathsf{daughter}(x))$$

- 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?)

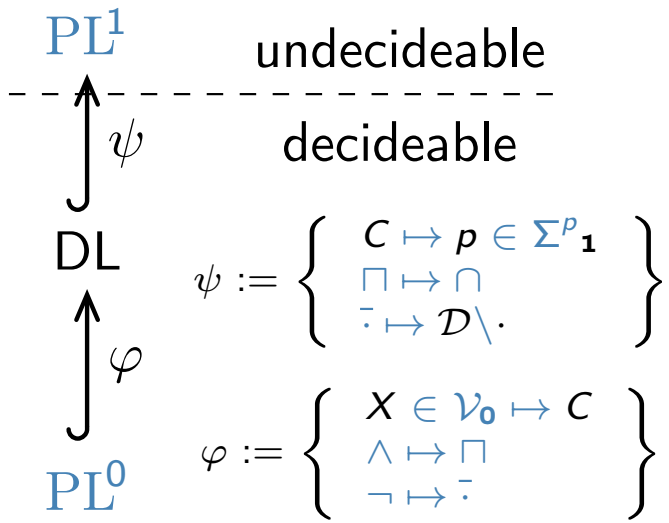
16.2.2 Ontologies and Description Logics

Ontologies aka. "World Descriptions"

- **Definition 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 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 2.11.** Semantic networks are ontologies. (relatively informal)
- **Example 2.12.** $\mathrm{PL}^0_{\mathrm{DL}}$ is an ontology format. (formal, but relatively weak)
- **Example 2.13.** PL^1 is an ontology format as well. (formal, expressive)

The Description Logic Paradigm

- **Idea:** Build a whole family of logics for describing sets and their relations. (tailor their expressivity and computational properties)
- **Definition 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 $\sqcap$, $\dot{\neg}$, $\sqcup$, $\sqsubseteq$, 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.



- **Definition 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.

TBoxes in Description Logics

- Let $\mathcal{D}$ be a description logic with concepts $\mathcal{C}$.
- **Definition 2.16.** A concept definition is a pair $c{=}C$, where c is a new concept name and $C \in \mathcal{C}$ is a $\mathcal{D}$-formula.
- **Example 2.17.** We can define mother=woman $\sqcap$ has_child.
- **Definition 2.18.** A concept definition $c{=}C$ is called recursive, iff c occurs in C.
- **Definition 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 2.20.** A formula A is called normalized wrt. an TBox $\mathcal{T}$, iff it does not contain concepts defined in $\mathcal{T}$. (convenient)
- **Definition 2.21 (Algorithm).** (for arbitrary DLs)
  Input: A formula A and a TBox $\mathcal{T}$.
  - While [A contains concept c and $\mathcal{T}$ a concept definition $c{=}C$]
    - substitute c by C in A.
- **Lemma 2.22.** *This algorithm terminates for acyclic TBoxes, but results can be exponentially large.*

16.2.3 Description Logics and Inference

Kinds of Inference in Description Logics

- **Definition 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

Consistency Test

- **Definition 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 2.25.** A concept C is called inconsistent, iff $[\![C]\!] = \emptyset$ for all $\mathcal{D}$.
- **Example 2.26 (T-Box in $\mathrm{PL}^0_{\mathrm{DL}}$).**

man	$=$	person $\sqcap$ has_Y	person with y-chromosome
woman	$=$	person $\sqcap \overline{\text{has\_Y}}$	person without y-chromosome
hermaphrodite	$=$	man $\sqcap$ 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 $\mathrm{PL}^0_{\mathrm{DL}}$.

Subsumption Test

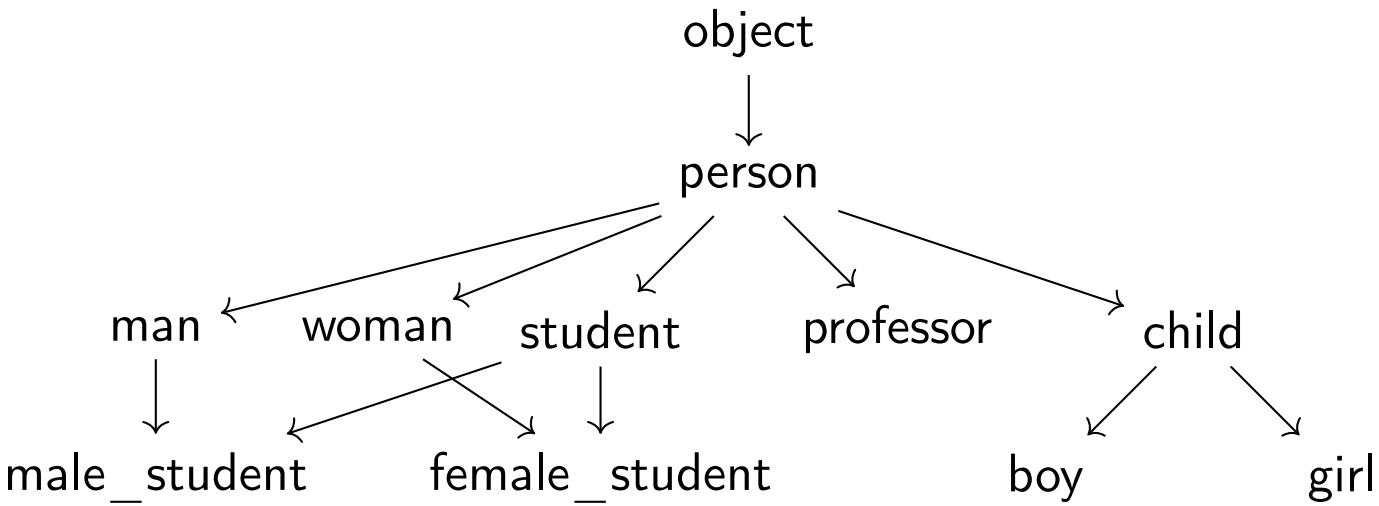
- **Example 2.27.** In this case trivial

axiom	entailed subsumption relation
$\mathsf{man} = \mathsf{person} \sqcap \mathsf{has\_Y}$	$\mathsf{man} \sqsubseteq \mathsf{person}$
$\mathsf{woman} = \mathsf{person} \sqcap \overline{\mathsf{has\_Y}}$	$\mathsf{woman} \sqsubseteq \mathsf{person}$

- **Definition 2.28.** A subsumes B (modulo a set $\mathcal{A}$ of concept axioms), iff $[\![\mathsf{B}]\!] \subseteq [\![\mathsf{A}]\!]$ for all interpretations $\mathcal{D}$ that satisfy $\mathcal{A}$.
- **Observation:** Or equivalently, iff $\mathcal{A} \sqsubseteq \mathsf{B} \sqsubseteq \mathsf{A} = \top$
- **Reduction to consistency test:** (need to implement only one)
  In PL^0, $\mathcal{A} \Rightarrow (\mathsf{A} \Rightarrow \mathsf{B})$ is valid iff $\mathcal{A} \wedge \mathsf{A} \wedge \neg\mathsf{B}$ is inconsistent.
- **In our example:** The concept person subsumes woman and man.

Classification

- The subsumption relation among all concepts (subsumption graph)
- Visualization of the subsumption graph for inspection (plausibility)
- **Definition 2.29.** Classification is the computation of the subsumption graph.
- **Example 2.30.** (not always so trivial)

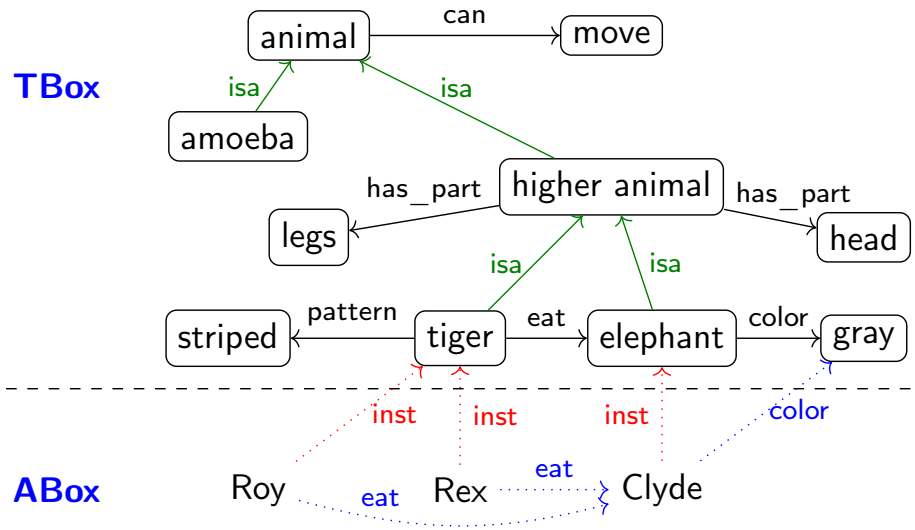


Instance Test: Inferring Concept Membership

- **Definition 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 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:



16.3 A simple Description Logic: ALC

16.3.1 Basic ALC: Concepts, Roles, and Quantification

Motivation for $\mathcal{ALC}$ (Prototype Description Logic)

- Propositional logic (PL^0) is not expressive enough!
- **Example 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)

$$\forall x.\mathit{mother}(x) \Leftrightarrow \mathit{woman}(x) \wedge (\exists y.\mathit{has\_child}(x,y))$$

- **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*.

Syntax of $\mathcal{ALC}$

- **Definition 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 3.3 (Special Concepts).** The top concept $\top$ (for "true" or "all") and the bottom concept $\bot$ (for "false" or "none").
- **Example 3.4.** person, woman, man, mother, professor, student, car, BMW, computer, computer program, heart attack risk, furniture, table, leg of a chair, . . .
- **Definition 3.5.** Roles represent binary relations (like in PL^1)
- **Example 3.6.** has_child, has_son, has_daughter, loves, hates, gives_course, executes_computer_program, has_leg_of_table, has_wheel, has_motor, . . .
- **Definition 3.7 (Grammar).** The formulae of $\mathcal{ALC}$ are given by the following grammar: $F_{\mathcal{ALC}} ::= C \mid \top \mid \bot \mid \overline{F_{\mathcal{ALC}}} \mid F_{\mathcal{ALC}} \sqcap F_{\mathcal{ALC}} \mid F_{\mathcal{ALC}} \sqcup F_{\mathcal{ALC}} \mid \exists \mathrm{R}.F_{\mathcal{ALC}} \mid \forall \mathrm{R}.F_{\mathcal{ALC}}$

Syntax of $\mathcal{ALC}$: Examples

- **Example 3.8.** person $\sqcap \exists$has_child.student
  $\widehat{=}$ The set of persons that have a child which is a student
  $\widehat{=}$ parents of students
- **Example 3.9.** person $\sqcap \exists$has_child.$\exists$has_child.student
  $\widehat{=}$ grandparents of students
- **Example 3.10.** person $\sqcap \exists$has_child.$\exists$has_child.(student $\sqcup$ teacher)
  $\widehat{=}$ grandparents of students or teachers
- **Example 3.11.** person $\sqcap \forall$has_child.student
  $\widehat{=}$ parents whose children are all students
- **Example 3.12.** person $\sqcap \forall \mathrm{haschild}.\exists$has_child.student
  $\widehat{=}$ grandparents, whose children all have at least one child that is a student

More $\mathcal{ALC}$ Examples

- **Example 3.13.** $\text{car} \sqcap \exists \text{has\_part}.\exists \text{made\_in}.\overline{\text{EU}}$
  $\widehat{=}$ cars that have at least one part that has not been made in the EU
- **Example 3.14.** $\text{student} \sqcap \forall \text{audits\_course}.\text{graduatelevelcourse}$
  $\widehat{=}$ students, that only audit graduate level courses
- **Example 3.15.** $\text{house} \sqcap \forall \text{has\_parking}.\text{off\_street} \widehat{=}$ houses with off-street parking
- **Note:** $p \sqsubseteq q$ can still be used as an abbreviation for $\overline{p} \sqcup q$.
- **Example 3.16.** $\text{student} \sqcap \forall \text{audits\_course}.(\exists \text{hastutorial}.\top \sqsubseteq \forall \text{has\_TA}.\text{woman})$
  $\widehat{=}$ students that only audit courses that either have no tutorial or tutorials that are TAed by women

$\mathcal{ALC}$ Concept Definitions

- **Idea:** Define new concepts from known ones.
- **Definition 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 3.18.**

Definition	rec?
$\mathsf{man} = \mathsf{person} \sqcap \exists \mathsf{has\_chrom}.\mathsf{Y\_chrom}$	-
$\mathsf{woman} = \mathsf{person} \sqcap \forall \mathsf{has\_chrom}.\overline{\mathsf{Y\_chrom}}$	-
$\mathsf{mother} = \mathsf{woman} \sqcap \exists \mathsf{has\_child}.\mathsf{person}$	-
$\mathsf{father} = \mathsf{man} \sqcap \exists \mathsf{has\_child}.\mathsf{person}$	-
$\mathsf{grandparent} = \mathsf{person} \sqcap \exists \mathsf{has\_child}.(\mathsf{mother} \sqcup \mathsf{father})$	-
$\mathsf{german} = \mathsf{person} \sqcap \exists \mathsf{has\_parents}.\mathsf{german}$	+
$\mathsf{number\_list} = \mathsf{empty\_list} \sqcup \exists \mathsf{is\_first}.\mathsf{number} \sqcap \exists \mathsf{is\_rest}.\mathsf{number\_list}$	+

TBox Normalization in $\mathcal{ALC}$

- **Definition 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 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 3.21 (Normalizing grandparent).**

$$\begin{aligned} & \mathsf{grandparent} \\ \mapsto\ & \mathsf{person} \sqcap \exists \mathsf{has\_child}.(\mathsf{mother} \sqcup \mathsf{father}) \\ \mapsto\ & \mathsf{person} \sqcap \exists \mathsf{has\_child}.(\mathsf{woman} \sqcap \exists \mathsf{has\_child}.\mathsf{person} \sqcap \mathsf{man} \sqcap \exists \mathsf{has\_child}.\mathsf{person}) \\ \mapsto\ & \mathsf{person} \sqcap \exists \mathsf{has\_child}.(\mathsf{person} \sqcap \exists \mathsf{has\_chrom}.\mathsf{Y\_chrom} \sqcap \exists \mathsf{has\_child}.\mathsf{person} \sqcap \mathsf{person} \sqcap \exists \mathsf{has\_chrom}.\mathsf{Y\_chrom} \sqcap \exists \mathsf{ha} \end{aligned}$$

- **Observation 3.22.** *Normalization results can be exponential.* (*contain redundancies*)
- **Observation 3.23.** *Normalization need not terminate on cyclic TBoxes.*
- **Example 3.24.**

$$\begin{aligned} \mathsf{german} &\mapsto \mathsf{person} \sqcap \exists \mathsf{has\_parents}.\mathsf{german} \\ &\mapsto \mathsf{person} \sqcap \exists \mathsf{has\_parents}.(\mathsf{person} \sqcap \exists \mathsf{has\_parents}.\mathsf{german}) \\ &\mapsto \ldots \end{aligned}$$

Semantics of $\mathcal{ALC}$

- $\mathcal{ALC}$ semantics is an extension of the set-semantics of propositional logic.
- **Definition 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
$[\![c]\!] \subseteq \mathcal{D} = [\![\top]\!]$	$[\![\bot]\!] = \emptyset$ $[\![r]\!] \subseteq \mathcal{D} \times \mathcal{D}$
$\bar{\cdot}$	$[\![\overline{\varphi}]\!] = \overline{[\![\varphi]\!]} = \mathcal{D} \setminus [\![\varphi]\!]$
$\sqcap$	$[\![\varphi \sqcap \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$
$\sqcup$	$[\![\varphi \sqcup \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$
$\exists \mathsf{R}.$	$[\![\exists \mathsf{R}.\varphi]\!] = \{x \in \mathcal{D} \mid \exists y. \langle x, y \rangle \in [\![\mathsf{R}]\!] \text{ and } y \in [\![\varphi]\!]\}$
$\forall \mathsf{R}.$	$[\![\forall \mathsf{R}.\varphi]\!] = \{x \in \mathcal{D} \mid \forall y. \text{if } \langle x, y \rangle \in [\![\mathsf{R}]\!] \text{ then } y \in [\![\varphi]\!]\}$

- Alternatively we can define the semantics of $\mathcal{ALC}$ by translation into PL^1.
- **Definition 3.26.** The translation of $\mathcal{ALC}$ into PL^1 extends the one from **??** by the following quantifier rules:

$$\overline{\forall \mathsf{R}.\varphi}^{fo(x)} := (\forall y. \mathsf{R}(x, y) \Rightarrow \overline{\varphi}^{fo(y)}) \quad \overline{\exists \mathsf{R}.\varphi}^{fo(x)} := (\exists y. \mathsf{R}(x, y) \wedge \overline{\varphi}^{fo(y)})$$

- **Observation 3.27.** *The set-theoretic semantics from* **??** *and the "semantics-by-translation" from* **??** *induce the same notion of satisfiability.*

$\mathcal{ALC}$ 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

$$
\begin{aligned}
[\![\overline{\exists \mathsf{R}.\varphi}]\!] = \mathcal{D} \setminus [\![\exists \mathsf{R}.\varphi]\!] &= \mathcal{D} \setminus \{x \in \mathcal{D} \mid \exists y.(\langle x, y\rangle \in [\![\mathsf{R}]\!] \text{ and } (y \in [\![\varphi]\!])\} \\
&= \{x \in \mathcal{D} \mid \text{not } \exists y.(\langle x, y\rangle \in [\![\mathsf{R}]\!] \text{ and } (y \in [\![\varphi]\!])\} \\
&= \{x \in \mathcal{D} \mid \forall y.\text{if } (\langle x, y\rangle \in [\![\mathsf{R}]\!] \text{ then } (y \notin [\![\varphi]\!])\} \\
&= \{x \in \mathcal{D} \mid \forall y.\text{if } (\langle x, y\rangle \in [\![\mathsf{R}]\!] \text{ then } (y \in (\mathcal{D} \setminus [\![\varphi]\!]))\} \\
&= \{x \in \mathcal{D} \mid \forall y.\text{if } (\langle x, y\rangle \in [\![\mathsf{R}]\!] \text{ then } (y \in [\![\overline{\varphi}]\!])\} \\
&= [\![\forall \mathsf{R}.\overline{\varphi}]\!]
\end{aligned}
$$

Negation Normal Form

- **Definition 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 3.29.**

example	by rule
$\overline{\exists \mathsf{R}.(\forall \mathsf{S}.e \sqcap \overline{\forall \mathsf{S}.d})}$	
$\mapsto \forall \mathsf{R}.\overline{\forall \mathsf{S}.e \sqcap \overline{\forall \mathsf{S}.d}}$	$\overline{\exists \mathsf{R}.\varphi} \mapsto \forall \mathsf{R}.\overline{\varphi}$
$\mapsto \forall \mathsf{R}.(\overline{\forall \mathsf{S}.e} \sqcup \overline{\overline{\forall \mathsf{S}.d}})$	$\overline{\varphi \sqcap \psi} \mapsto \overline{\varphi} \sqcup \overline{\psi}$
$\mapsto \forall \mathsf{R}.(\exists \mathsf{S}.\overline{e} \sqcup \overline{\overline{\forall \mathsf{S}.d}})$	$\overline{\forall \mathsf{R}.\varphi} \mapsto \exists \mathsf{R}.\overline{\varphi}$
$\mapsto \forall \mathsf{R}.(\exists \mathsf{S}.\overline{e} \sqcup \forall \mathsf{S}.d)$	$\overline{\overline{\varphi}} \mapsto \varphi$

$\mathcal{ALC}$ with Assertions about Individuals

- **Definition 3.30.** We define the ABox assertions for $\mathcal{ALC}$:
  - Role assertions $a{:}\varphi$ (a is a φ)
  - a R b (a stands in relation R to b)

  assertions make up the ABox in $\mathcal{ALC}$.
- **Definition 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
  - $[\![a \text{ R } b]\!] = \mathsf{T}$, iff $([\![a]\!], [\![b]\!]) \in [\![\mathsf{R}]\!]$.
- **Definition 3.32.** We extend the PL^1 translation of $\mathcal{ALC}$ to $\mathcal{ALC}$ assertions:
  - $\overline{a{:}\varphi}^{fo} := \overline{\varphi}^{fo(a)}$, and
  - $\overline{a \text{ R } b}^{fo} := \mathsf{R}(a, b)$.

16.3.2 Inference for ALC

$\mathcal{T}_{\mathcal{ALC}}$: 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 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:

$$\frac{x{:}C \quad x{:}\overline{C}}{\bot}\,\mathcal{T}_{\bot} \qquad \frac{x{:}\varphi \sqcap \psi}{x{:}\varphi \quad x{:}\psi}\,\mathcal{T}_{\sqcap} \qquad \frac{x{:}\varphi \sqcup \psi}{x{:}\varphi \mid x{:}\psi}\,\mathcal{T}_{\sqcup} \qquad \frac{x{:}\forall \mathsf{R}.\varphi \quad x\ \mathsf{R}\ y}{y{:}\varphi}\,\mathcal{T}_{\forall} \qquad \frac{x{:}\exists \mathsf{R}.\varphi}{x\ \mathsf{R}\ y \quad y{:}\varphi}\,\mathcal{T}_{\exists}$$

- To test consistency of a concept φ, normalize φ to ψ, initialize the tableau with $x{:}\psi$, saturate. Open branches $\rightsquigarrow$ consistent. (x arbitrary)

- **Example 3.34 (Tableau Proofs).** We have two similar conjectures about children.
  - $x{:}\forall \text{has\_child}.\text{man} \sqcap \exists \text{has\_child}.\overline{\text{man}}$ (all sons, but a daughter)

$x{:}\forall \text{has\_child}.\text{man} \sqcap \exists \text{has\_child}.\overline{\text{man}}$	initial
$x{:}\forall \text{has\_child}.\text{man}$	$\mathcal{T}_\sqcap$
$x{:}\exists \text{has\_child}.\overline{\text{man}}$	$\mathcal{T}_\sqcap$
x has_child y	$\mathcal{T}_\exists$
$y{:}\overline{\text{man}}$	$\mathcal{T}_\exists$
$\bot$	$\mathcal{T}_\bot$
inconsistent	

  - $x{:}\forall \text{has\_child}.\text{man} \sqcap \exists \text{has\_child}.\text{man}$ (only sons, and at least one)

$x{:}\forall \text{has\_child}.\text{man} \sqcap \exists \text{has\_child}.\text{man}$	initial
$x{:}\forall \text{has\_child}.\text{man}$	$\mathcal{T}_\sqcap$
$x{:}\exists \text{has\_child}.\text{man}$	$\mathcal{T}_\sqcap$
x has_child y	$\mathcal{T}_\exists$
$y{:}\text{man}$	$\mathcal{T}_\exists$
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 $\mathcal{TALC}$ Example

- **Example 3.35.** $\forall \text{has\_child}.(\text{ugrad} \sqcup \text{grad}) \sqcap \exists \text{has\_child}.\overline{\text{ugrad}}$ is satisfiable.
  - Let's try it on the board

Another $\mathcal{T}_{\mathcal{ALC}}$ Example

- **Example 3.36.** $\forall \text{has\_child}.(\text{ugrad} \sqcup \text{grad}) \sqcap \exists \text{has\_child}.\overline{\text{ugrad}}$ is satisfiable.
  - Let's try it on the board
  - Tableau proof for the notes

1	$x{:}\forall \text{has\_child}.(\text{ugrad} \sqcup \text{grad}) \sqcap \exists \text{has\_child}.\overline{\text{ugrad}}$	initial
2	$x{:}\forall \text{has\_child}.(\text{ugrad} \sqcup \text{grad})$	$\mathcal{T}_\sqcap$
3	$x{:}\exists \text{has\_child}.\overline{\text{ugrad}}$	$\mathcal{T}_\sqcap$
4	$x\ \text{has\_child}\ y$	$\mathcal{T}_\exists$
5	$y{:}\overline{\text{ugrad}}$	$\mathcal{T}_\exists$
6	$y{:}\text{ugrad} \sqcup \text{grad}$	$\mathcal{T}_\forall$
7	$y{:}\text{ugrad}$ | $y{:}\text{grad}$	$\mathcal{T}_\sqcup$
8	$\bot$ | 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.

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)

Correctness

- **Lemma 3.37.** *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 $w \in [\![\varphi]\!]$.

$$\begin{array}{lll} \mathcal{M} \models (x{:}\psi) & \text{iff} & [\![x]\!] \in [\![\psi]\!] \\ \mathcal{M} \models x \ \mathsf{R} \ y & \text{iff} & \langle x, y \rangle \in [\![\mathsf{R}]\!] \\ \mathcal{M} \models S & \text{iff} & \mathcal{I} \models c \text{ for all } c \in S \end{array}$$

1. We define $[\![x]\!] := w$
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)

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 $\mathcal{M}\models x\ \mathsf{R}\ y$, 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 $\mathcal{M}\models x\ \mathsf{R}\ y$ and $\mathcal{M}\models(y{:}\varphi)$

Completeness of the Tableau Calculus

- **Lemma 3.38.** *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

$$\begin{aligned} \mathcal{D} &:= \{x \mid x{:}\psi \in \mathcal{B} \text{ or } z \mathsf{R}\, x \in \mathcal{B}\} \\ [\![c]\!] &:= \{x \mid x{:}c \in \mathcal{B}\} \\ [\![\mathsf{R}]\!] &:= \{\langle x, y\rangle \mid x \mathsf{R}\, y \in \mathcal{B}\} \end{aligned}$$

     - 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 $k = \mathit{size}(\psi)$ next slide)
  3. $\mathcal{M} \models (x{:}\varphi)$.

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 $y{:}\psi_1 \in \mathsf{B}$ or $y{:}\psi_2 \in \mathsf{B}$ ($\mathcal{T}_\sqcup$, 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 $\{y\ \mathsf{R}\ z, z{:}\theta\} \subseteq \mathsf{B}$ (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 $y\ \mathsf{R}\ z \in \mathsf{B}$ (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 3.39.** $\mathcal{T}_{\mathcal{ALC}}$ *terminates.*
- To prove termination of a tableau algorithm, find a well-founded measure (function) that is decreased by all rules

$$\frac{x{:}C \quad x{:}\overline{C}}{\bot}\,\mathcal{T}_\bot \qquad \frac{x{:}\varphi \sqcap \psi}{x{:}\varphi \quad x{:}\psi}\,\mathcal{T}_\sqcap \qquad \frac{x{:}\varphi \sqcup \psi}{x{:}\varphi \mid x{:}\psi}\,\mathcal{T}_\sqcup \qquad \frac{x{:}\forall \mathrm{R}.\varphi \quad x\ \mathrm{R}\ y}{y{:}\varphi}\,\mathcal{T}_\forall \qquad \frac{x{:}\exists \mathrm{R}.\varphi}{x\ \mathrm{R}\ y \quad y{:}\varphi}\,\mathcal{T}_\exists$$

- *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 $x{:}\forall \mathrm{R}.\psi$ at most as the number of R-successors of x. (those y with x R y above)
  3. The R-successors are generated by $x{:}\exists \mathrm{R}.\theta$ 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.

Complexity of $\mathcal{TALC}$

- **Idea:** Work off tableau branches one after the other. (Branch size $\widehat{=}$ space complexity)
- **Observation 3.40.** *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 3.41.** *The satisfiability problem for $\mathcal{ALC}$ is in PSPACE.*
- **Theorem 3.42.** *The satisfiability problem for $\mathcal{ALC}$ is PSPACE-Complete.*
- *Proof sketch:* Reduce a PSPACE-complete problem to $\mathcal{ALC}$-satisfiability
- **Theorem 3.43 (Time Complexity).** *The $\mathcal{ALC}$ satisfiability problem is in EXPTIME.*
- *Proof sketch:* There can be exponentially many branches (already for PL^0)

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 $x{:}\exists \mathsf{R}.\psi$
  - this triggers the $\mathcal{T}_\forall$-rule for $x{:}\forall \mathsf{R}.\theta_i$, which generate $y{:}\theta_1, \ldots, y{:}\theta_n$
  - for y we have $y{:}\psi$ and $y{:}\theta_1, \ldots, y{:}\theta_n$. (do all of this in a single step)
  - we are only interested in non-emptiness, not in particular witnesses (leave them out)
- **Definition 3.44.** 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($\{\psi\} \cup \{\theta \in \theta \mid `\forall \mathsf{R}.\theta' \in S\}$)
  else true

- Relatively simple to implement. (good implementations optimized)
- **But:** This is restricted to $\mathcal{ALC}$. (extension to other DL difficult)

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 $\exists$: $\dfrac{x{:}\exists \mathsf{R}.\varphi \quad \mathcal{CA} = \{\alpha_1, \ldots, \alpha_n\}}{\begin{array}{c} y{:}\varphi \\ x\ \mathsf{R}\ y \\ y{:}\alpha_1 \\ \vdots \\ y{:}\alpha_n \end{array}}\ \mathcal{T}^{\exists}_{\mathcal{CA}}$ (instead of $\mathcal{T}_\exists$)
- **Problem:** $\mathcal{CA} := \{\exists \mathsf{R}.c\}$ and start tableau with $x{:}d$ (non-termination)

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 3.45.** *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

Instance Test: Concept Membership

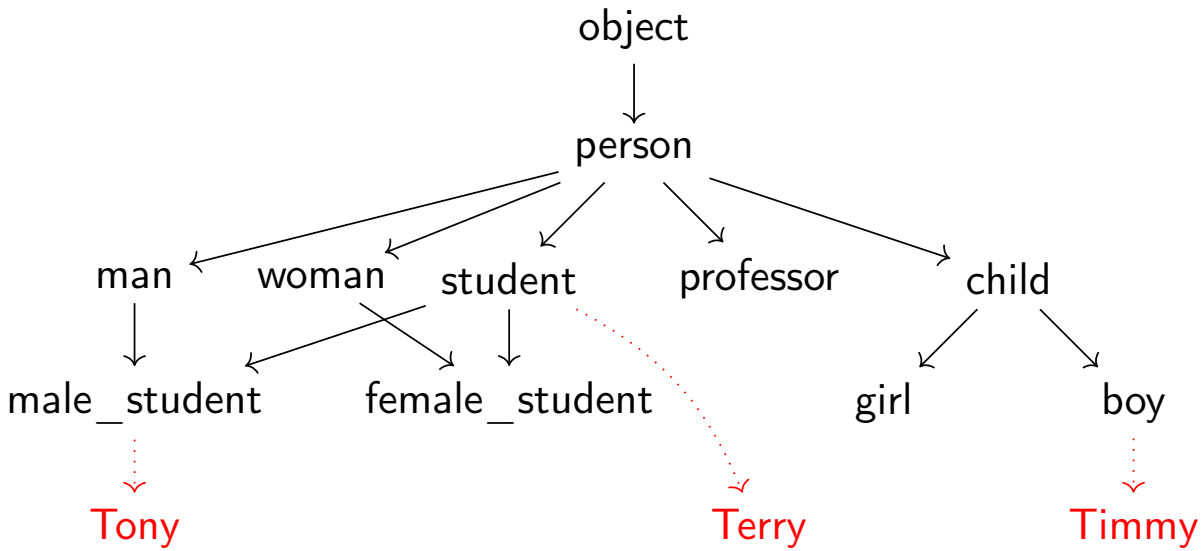
- **Definition 3.46.** An instance test computes whether given an $\mathcal{ALC}$ ontology an individual is a member of a given concept.
- **Example 3.47 (An Ontology).**

TBox (terminological Box)			ABox (assertional Box, data base)	
woman	=	person $\sqcap \overline{\text{has\_Y}}$	tony:person	Tony is a person
man	=	person $\sqcap$ has_Y	tony:has_Y	Tony has a y-chrom

  This entails: tony:man (Tony is a man).
- **Problem:** Can we compute this?

Realization

- **Definition 3.48.** 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 3.49.** If tony:male_student is known, we do not need tony:man.

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 3.50.**

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:$\forall$has_grad.genius tony has_grad mary $\models$ mary:genius
information that can be combined with TBox info	TBox: happy_prof $=$ prof $\sqcap \forall$has_grad.genius ABox: tony:happy_prof, tony has_grad mary $\models$ mary:genius

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 3.51.**

Example: add mary:$\overline{\text{genius}}$ to determine *ABox*, *TBox* $\models$ mary:genius		
TBox	happy_prof = prof $\sqcap$ $\forall$has_grad.genius	tony:prof $\sqcap$ $\forall$has_grad.genius TBox tony has_grad mary ABox mary:$\overline{\text{genius}}$ Query tony:prof $\mathcal{T}_\sqcap$ tony:$\forall$has_grad.genius $\mathcal{T}_\sqcap$ mary:genius $\mathcal{T}_\forall$ $\bot$ $\mathcal{T}_\bot$
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)

16.4 Description Logics and the Semantic Web

Resource Description Framework

- **Definition 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 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

Resources and URIs

- RDF describes resources with properties and property values.
- RDF uses Web identifiers (URIs) to identify resources.
- **Definition 4.3.** A resource is anything that can have a URI, such as `http://www.fau.de`.
- **Definition 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 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 4.6.** Statements: *[This slide]^{subj} has been [author]^{pred}ed by [Michael Kohlhase]^{obj}*

XML Syntax for RDF

- RDF is a concrete XML vocabulary for writing statements
- **Example 4.7.** The following RDF document could describe the slides as a resource

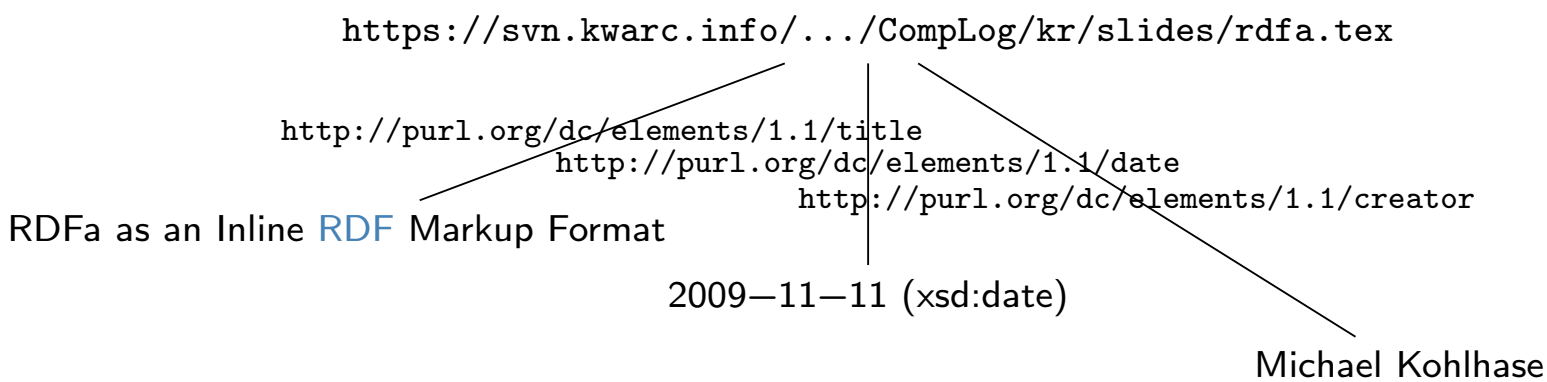
```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc= "http://purl.org/dc/elements/1.1/">
  <rdf:Description about="https://.../CompLog/kr/en/rdf.tex">
    <dc:creator>Michael Kohlhase</dc:creator>
    <dc:source>http://www.w3schools.com/rdf</dc:source>
  </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.

RDFa as an Inline RDF Markup Format

- **Problem:** RDF is a standoff markup format (annotate by URIs pointing into other files)
  Definition 4.8. RDFa (RDF annotations) is a markup scheme for inline annotation (as XML attributes) of RDF triples.
- **Example 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>
```



RDF as an ABox Language for the Semantic Web

- **Idea:** RDF triples are ABox entries h R s or $h{:}\varphi$.
- **Example 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 $\mathcal{ALC}$)

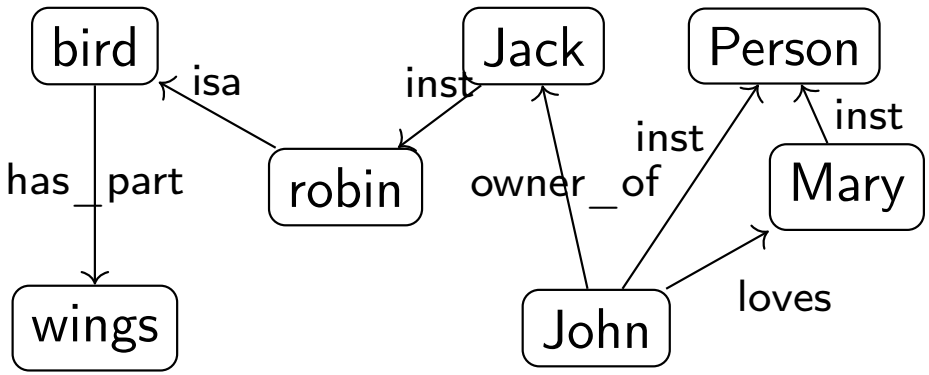
OWL as an Ontology Language for the Semantic Web

- **Task:** Complement RDF (ABox) with a TBox language.
- **Idea:** Make use of resources that are values in rdf:type. (called Classes)
- **Definition 4.11.** OWL (the ontology web language) is a language for encoding TBox information about RDF classes.
- **Example 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))

Extended OWL Example in Functional Syntax

- **Example 4.13.** The semantic network from ?? can be expressed in OWL (in functional syntax)



  - 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*.

Extended OWL Example in Functional Syntax

- **Example 4.14.** 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*.

SPARQL an RDF Query language

- **Definition 4.15.** 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 4.16.** A system is called a SPARQL endpoint, iff it answers SPARQL queries.
- **Example 4.17.** 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 Applications: DBPedia

- **Typical Application:** DBPedia screen-scrapes Wikipedia fact boxes for RDF triples and uses SPARQL for querying the induced triplestore.
- **Example 4.18 (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
```

Results: Browse Go! Reset

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

Triple Stores: the Semantic Web Databases

- **Definition 4.19.** 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 4.20.** 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.

References I

[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.

[Gen34] Gerhard Gentzen. "Untersuchungen über das logische Schließen I". In: *Mathematische Zeitschrift* 39.2 (1934), pp. 176–210.

[Her+13] 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/.

[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/.

[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/.

References II

[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.