AI-KI-B

Logics for Representing Knowledge

Ute Schmid & Diedrich Wolter
Practice: Johannes Rabold

Cognitive Systems and Smart Environments
Applied Computer Science, University of Bamberg

last change: 12. Juni 2019, 13:05

Quiz!

1. What is the difference between information and knowledge?
2. Is it necessary for a knowledge representation to discard some pieces of information?
3. Why can't we expect to identify a universal knowledge representation that suits all tasks perfectly?

Agenda

Topics for today

- Architectures of Logics: How logics are built
- Logic I: propositional logic
- Logic II: first order logic
- Reasoning with logics

Educational objectives: being able to...

- understand and develop logic statements as representation
- characterise and compare logics according to their design principles
- identify effects of design choices on how a logic can be used for representing knowledge

Disclaimer

A common **misunderstanding**

> *Logics are only true and false, but the world isn't black-and-white. Thus, logics are not relevant for real applications. (homo stupididous)*

1. Some logics are built upon 'true' and 'false', others around notions of 'maybe', of 'passt 'scho', of numerical values, or estimates.
   $\rightsquigarrow$ Clearly, we cannot start with the fancy logics and won't have time to go much beyond the basics.
2. Logics are very important to many applications, inside computer science and outside
   - Fundamental skill of (computer) science graduates: Being able to write down facts in a formal language, e.g., for specification
   - Logics as basis for software engineering and tools
   - Logics as overarching frameworks for AI algorithms

Logics

Two ingredients constitute a logic:

1. **Syntax**, i.e., the formal language of the logic; explains what a **well-formed formula**
2. **Validity semantics** defines the meaning of expressions and formulae

Logics are equipped with reasoning techniques to form a **calculus**:

3. Set of **manipulation rules** for manipulating logic statements

Usual design considerations apply

- easy to read
- orthogonal

Besides, we need to prove properties of arbitrary formulae

- recursive syntax rules enable induction proofs over structure of formulae (**structural induction**)

- Let a countably infinite set $A = \{a_1, a_2, \ldots\}$ of symbols, called **atoms**, be given
- Let a set of binary operators $O = \{\wedge, \vee\}$ be given
  $\rightsquigarrow$ note: we *could* use a different set of operators
- Let a set of unary operators $U = \{\neg\}$ be given

Definition

We call ϕ a **well-formed formula (WFF)** in propositional logics if it suits one of these rules

1. $\phi \in A$, i.e., ϕ is an atom
2. $\phi = \ominus\psi$, with $\ominus \in U$ and ψ being a WFF
3. $\phi = (\psi_1 \oplus \psi_2)$, with $\oplus \in O$ and ψ_1, ψ_2 both being WFFs

The set of all WFFs is sometimes written $\mathcal{F}$.

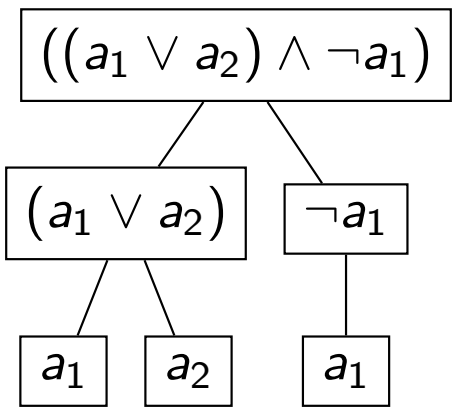
The definition is orthogonal:

Theorem

For every $\phi \in \mathcal{F}$ there exists exactly one way to construct the formula according to the syntax rules.

Proof Idea.

For every formula exactly one of the rules can be applied. □



Our definition of WFFs is isomorphic to circuit algebra which often uses $\overline{a_1}$ for negation, $\cdot$ for $\wedge$, and $+$ for $\vee$.

Some conventions are commonly used when communicating WFFs

- Set of atoms is often not mentioned explicitly, $a \wedge b$ and $x \vee y'$ are both WFFs.
- Parentheses may be omitted if formula structure is clear.
  - depends on validity semantics
  - for propositional logics: $(a_1 \wedge a_2) \vee a_3 \rightsquigarrow a_1 \wedge a_2 \vee a_3$ precedence of $\wedge$ over $\vee$ ("Punkt- vor Strichrechnung")
- Some rewrite rules are commonly assumed:

$$\begin{aligned} \phi \rightarrow \psi &:\equiv (\neg\phi \vee \psi) \\ \phi \leftarrow \psi &:\equiv (\phi \vee \neg\psi) \\ \phi \leftrightarrow \psi &:\equiv ((\phi \leftarrow \psi) \wedge (\phi \rightarrow \psi)) \end{aligned}$$

Rewrite rules for other operators can be introduced.

For propositional logic, **validity** is binary:

1. $\top$ (read: top, type: \top)
2. $\bot$ (read: bottom, type: \bot)

Validity semantics defines semantics of WFFs by explaining validity of every WFF.
Intuitively, $\top$ resembles 'true' and $\bot$ resembles 'false'. However, $\top, \bot$ make clear that we deal with **symbols that represent meaning**.

Some authors distinguish symbols $\top, \bot$ which represent two validities from actual validities **T**, **F**, we only use symbols $\top, \bot$.

Definition: We call $I : \mathcal{F} \to \{\top, \bot\}$ an **interpretation** of a WFF in propositional logic if it is constructed as follows:

1. $I(a)$ is arbitrary for $a \in A$
2. $I(\phi)$ is defined inductively for $\phi \in \mathcal{F} \setminus A$.

$$
\begin{aligned}
I(\top) &:= \top \\
I(\bot) &:= \bot \\
I(\neg\phi) &:= \begin{cases} \top & \text{if } I(\phi) = \top \\ \bot & \text{otherwise} \end{cases} \\
I(\underbrace{\psi \vee \chi}_{=\phi}) &:= \begin{cases} \top & \text{if } I(\psi) = \top \text{ or } I(\chi) = \top \\ \bot & \text{otherwise} \end{cases} \\
I(\underbrace{\psi \wedge \chi}_{=\phi}) &:= \begin{cases} \top & \text{if } I(\psi) = \top \text{ and } I(\chi) = \top \\ \bot & \text{otherwise} \end{cases}
\end{aligned}
$$

Note: Definition benefits from having only two binary operators!

CNF and DNF

A formula ϕ is in **disjunctive normal form (DNF)** if it matches

$$\underbrace{(l_{1,1} \wedge l_{1,2} \wedge \ldots \wedge l_{1,n_1})}_{C_1} \vee \underbrace{(l_{2,1} \wedge l_{2,2} \wedge \ldots \wedge l_{2,n_2})}_{C_2} \vee \ldots \vee \underbrace{(l_{1,1} \wedge l_{k,2} \wedge \ldots \wedge l_{k,n_k})}_{C_k}$$

- $l_{i,j}$ are called **literals**, which is either an atom or an negated atom, i.e., $l_{i,j} = a, a \in \mathcal{A}$ or $l_{i,j} = \neg a, a \in \mathcal{A}$.
  $\rightsquigarrow$ no negations except in front of atoms!
- C_i are called **clauses**
- In DNF, the formula is written as disjunction $\phi = \bigvee_{i=1}^{k} C_i$

A formula ϕ is in **conjunctive normal form (CNF)** if it matches

$$\underbrace{(l'_{1,1} \vee l'_{1,2} \vee \ldots \vee l'_{1,n_1})}_{C'_1} \wedge \underbrace{(l'_{2,1} \vee l'_{2,2} \vee \ldots \vee l'_{2,n_2})}_{C'_2} \wedge \ldots \wedge \underbrace{(l'_{1,1} \vee l'_{k,2} \vee \ldots \vee l'_{k,n_{k'}})}_{C'_{k'}}$$

- In CNF, the formula is written as conjunction $\phi = \bigwedge_{i=1}^{k} C'_i$.

There are several ways of writing equivalent formulae, e.g., for $\phi = \neg(a_1 \vee a_2)$ and $\psi = \neg a_1 \wedge \neg a_2$ it holds that $I(\phi) = I(\psi)$ for every valuation V. For syntactic manipulation it is often advantageous to reduce syntactic variations, e.g., to avoid negation in front of formulae that are no atoms (like CNF or DNF).

Theorem

Every $\phi \in \mathcal{F}$ can be transformed into an equivalent formulae in CNF and DNF, respectively.

Proof.

Left blank intensionally! □

Satisfiability

For a WFF ϕ, three classes of **satisfiability** are distinguished:

satisfiable There exists a valuation V such that $I(\phi) = \top$

tautology For every valuation V it holds that $I(\phi) = \top$

contradiction For every valuation V it holds that $I(\phi) = \bot$

A natural question arises: Given a WFF ϕ, is ϕ satisfiable, a contradiction, or an tautology?

For a WFF ϕ, three classes of **satisfiability** are distinguished:

satisfiable There exists a valuation V such that $I(\phi) = \top$

tautology For every valuation V it holds that $I(\phi) = \top$

contradiction For every valuation V it holds that $I(\phi) = \bot$

A natural question arises: Given a WFF ϕ, is ϕ satisfiable, a contradiction, or an tautology?

Definition

The satisfiability problem of propositional logic **SAT** is the problem of deciding whether a given WFF ϕ is satisfiable.

- **Decision problem**: answer yes or no
- Instances given (without loss of generality) in CNF
- Fundamental, NP-complete problem

SAT-Solving

SAT-Solving allows us to answer several logic problems, e.g.:

1. ϕ is a tautology iff $\neg\phi$ is not satisfiable
2. ϕ is a contradiction iff ϕ is not satisfiable
3. ϕ is satisfiable iff ϕ and $\neg\phi$ are both satisfiable

Principle algorithms:

- construct truth tables (EXPTIME)
- search for valuation of atoms that lets formula evaluate to $\top$ (NP-complete)
  $\rightsquigarrow$ link to constraint-satisfaction problems (CSPs)
- natural deduction, resolution (next week), ...

Solving SAT instances, i.e. deciding satisfiability, has many important applications, e.g., when checking requirements in engineering

- SAT instances generated automatically from design software

Importance is reflected in **scientific competitions**: implementations compete at solving as many as possible SAT instances in a given time frame.

SAT Competitions

The international SAT Competitions web page

Current Competition

SAT 2019 Race	
Organizers	Marijn Heule, Matti Järvisalo, Martin Suda

Past Competitions

SAT 2018 Competition	
Organizers	Marijn Heule, Matti Järvisalo, Martin Suda

SAT 2017 Competition	
Organizers	Marijn Heule, Matti Järvisalo, Tomáš Balyo
Slides	Slides used at SAT 2017
Proceedings	Descriptions of the solvers and benchmarks
Benchmarks	Available here
Solvers	Available here

	Gold	Silver	Bronze	Gold	Silver	Bronze	Gold	Silver	Bronze
	Agile Track			Main Track			Random Track		
SAT+UNSAT	CaDiCaL Agile, CaDiCaL NoProof	Glu_VC	Glucose 4.1	Maple LCM Dist, Maple LCM, MapleLRB LCMOccRestart, MapleLRB LCM	MapleCOMSPS LRB VSIDS 2, MapleCOMSPS LRB VSIDS	COMiniSATPS Pulsar	YalSAT	tch glucose3	Score2SAT
	Parallel Track			No-Limit Track			Incremental Library Track		
SAT+UNSAT	Syrup24, Syrup48	Plingeling	Painless MapleCOMSPS	COMiniSATPS Pulsar	MapleCOMPSPS LRB VSIDS 2, MapleCOMPSPS LRB VSIDS	CaDiCaL NoProof	AbcdSAT	Glucose	Riss

screenshot satcompetition.org

- empirical methodology to identify algorithms that perform well
- analysed across different categories of SAT instances

Variations of SAT

For computational analysis, several variations of SAT are helpful, as they ease **reduction proofs**, e.g., to show NP-completeness.

- 3SAT: Like SAT, but at most 3 literals per clause (still NP-complete!)
- 2SAT: Like SAT, but at most 2 literals per clause (in P!)
- NON-ALL-EQUAL-3SAT: Like 3SAT, but at least one negated and one un-negated atom per clause (NP-complete)

Whenever an interpretation I makes a formula ϕ evaluate to $\top$, i.e., $V(\phi) = \top$, we say I provides a **model for** ϕ, written

$$I \models \phi$$

(read: models, type: `\models`)

- We also say that ϕ **is valid in** I.
- The set of tautologies is also called the set of **valid formulas**.

- In Knowledge Representation we also write

$$\{\psi_1, \ldots, \psi_k\} \models \phi \text{ or } \psi \models \phi$$

meaning that ϕ is valid in all interpretations that make $\psi_1 \wedge \ldots \wedge \psi_k$ (respectively ψ) evaluate to $\top$.

$\rightsquigarrow$ operator $\models$ represents **logical consequence**, also called **entailment**!

- For a tautology F, i.e., a valid formula F, we may write $\models F$
- Note that $\models$ is used both as model and entailment relation

Definition

Given an interpretation I and a formula ϕ, the task to decide whether $V \models \phi$ holds is called **model checking**.

- In propositional logic, the task is very simple: evaluate ϕ according to validity semantics (polynomial time).
- In more complex logics, the task can be computationally hard (NP time or even undecidable)
- Model checking is particularly important for software engineering (verification), e.g., to check whether observations $\models$ contracts is satisfied.

Symbolic manipulation rules as basis for reasoning algorithms:

- purely **based on syntax**
- manipulation operator $\boxed{\vdash: \mathcal{F} \to \mathcal{F}}$
- example: $\neg\neg a \vdash a$
- often multiple manipulations are required

$$\neg\neg\neg\neg a \vdash \neg\neg a \vdash a,$$

which motivates considering $\vdash^*$ (Kleene operator *)

$$\begin{aligned} \vdash^0 &:= \{(\phi, \phi) | \phi \in \mathcal{F}\} \\ \vdash^{i+1} &:= \{(\phi, \psi) |\ \phi \vdash^i \phi' \wedge \phi' \vdash \psi\} \\ \vdash^* &:= \bigcup_{i=0}^{\infty} \vdash^i \end{aligned}$$

We often aim at symbolic manipulation operators that capture logical consequence.

Definition

A set of manipulation rules is called **sound** iff (iff: if and only if)

$$\phi \vdash^* \psi \rightarrow \phi \models \psi$$

is satisfied for all manipulations and all formulae ϕ.

Definition

A set of manipulation rules is called **complete** iff for all formulae ϕ, ψ

$$\phi \models \phi \rightarrow \phi \vdash^* \psi$$

holds.

- Sound manipulation does not change validity. (It may miss some consequences, though.)
- Complete manipulation does not miss any consequences. (It may change validity, though.)

Manipulation operators that are **sound and complete** implement entailment, i.e., they lift logic reasoning to symbolic manipulation.

For some logics, it may be impossible to provide a sound and complete set of operators. In some applications, we might not even aim at soundness!

Example (abduction)

street-is-wet $\vdash$ it-rained can be a useful inference, but
street-is-wet $\not\models$ it-rained, only it-rained $\models$ street-is-wet holds.

According to chosen validity semantics, operators U, O and values $\top, \bot$ constitute the **Boolean algebra** $\langle \top, \bot, \wedge, \vee, \neg \rangle$.

Corollary

The set of axioms of Boolean algebras constitutes a set of sound manipulation operators.

Examples:

- $(a_1 \vee a_2) \vdash (a_2 \vdash a_1)$ (commutative law)
- $\top \vee a_1 \vdash a_1$ (absorption)
- $a_1 \vdash \top \vee a_1$
- $(a_1 \vee a_2) \vdash \neg(\neg a_1 \wedge \neg a_2)$ (De Morgan)

Theorem

In propositional logic $\vdash$ defined according to axioms of Boolean algebras is a sound and complete set of manipulation operators.

Proof Idea.

Soundness directly entailed by Boolean algebra. Completeness requires us to show that $\phi \vdash^* \phi$ can be achieved. Completeness proof is based on an earlier theorem that every formula can be transferred into DNF (or CNF).

Theorem

In propositional logic $\vdash$ defined according to axioms of Boolean algebras is a sound and complete set of manipulation operators.

Proof Idea.

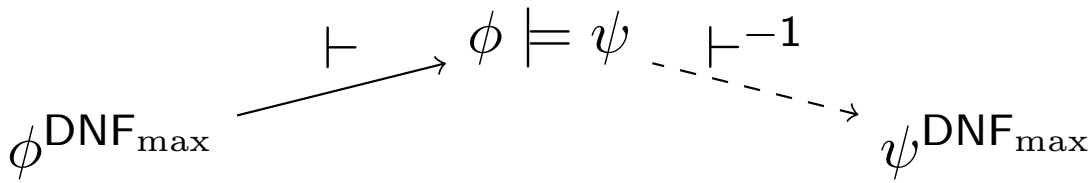
Achieving normal form accomplished by sound set of manipulations $M = \{\alpha_1 \vdash \beta_1, \ldots, \alpha_k \vdash \beta_k\}$. Wlog. one can choose a normal form $DNF_{\max}$ in which every clause has n literals, with n being the total number of atoms occurring in ϕ and ψ.

Theorem

In propositional logic $\vdash$ defined according to axioms of Boolean algebras is a sound and complete set of manipulation operators.

Proof Idea.

Assume, we first would turn ϕ, ψ into $DNF_{\max}$, $\phi \vdash^* \phi^{\mathsf{DNF}_{\max}}, \psi \vdash^* \psi^{\mathsf{DNF}_{\max}}$. Then, we simply had to inverse $\psi \vdash^* \psi^{\mathsf{DNF}_{\max}}$. Literals and clauses may need to be re-sorted using commutativity. Since ψ is unknown in advance, we have to search sorting and inverse applications of M. Inversion does not affect soundness. Thus, the set $M \cup \{\beta \vdash \alpha | \alpha \vdash \beta \in M\}$ is sound and complete.



Definition

The logic reasoning procedure to infer ψ, given ϕ such that $\phi \models \psi$ holds, is called **deduction**.

- $\phi \models \psi$ holds iff $\phi \rightarrow \psi$ is a tautology

$\rightsquigarrow$ Apply SAT-Solving to verify!

- hypothesis ψ must be given in advance

Idea of **natural deduction**: explore the space of formulae constructed by iteratively applying sound manipulation operators.

Manipulation rules are sometimes denoted graphically:

$$\frac{\phi_1, \ldots, \phi_k}{\psi}$$

- $\phi_1, \ldots \phi_k$ are called premises
- ψ is called conclusion
- Example: $\dfrac{\neg\neg\phi}{\phi}$
- Unlike $\vdash$, the rule notation can be applied to tasks in which multiple premises are required, e.g., resolution discussed next week.
- Some authors define rules that introduce conclusions and rules that eliminate facts

KR using Propositional Logic

Some challenges when modelling knowledge:

- No relations as in, e.g., `(is-a penguin bird)`
- No variables to build abstractions
- Interpretation only using Boolean validity

Possible approaches:

- Introduce atom for any relation, e.g., $a_{\text{is-a-penguin-bird}} = \top$, $a_{\text{is-penguin-mammal}} = \bot \ldots$
- Introduce explicit disjunctions for variables (only possible for finite, closed domains!), e.g., `(is-a ?X bird)` $\rightsquigarrow$ $a_{\text{is-a-tweety-bird}} \vee a_{\text{is-a-fred-bird}} \vee \ldots$
- Simulate other domains using Boolean atoms and an explicit **domain theory**, e.g., if $a_{\text{very-fast}}$ represents a very fast car, $a_{\text{very-fast}} \rightarrow a_{\text{fast}} \wedge a_{\text{slow}}$ need to hold.

Propositional Logic in KR

Little importance as language for modelling knowledge:

- Lacks intuitive concept-building elements, e.g., relations as used in semantic networks
- Lacks expressivity to generalise statements

High importance as part of other approaches:

- Boolean logic contained in many more complex logics
- NP-completeness of SAT allows reduction from other NP languages ($\rightsquigarrow$ a somewhat "general problem solver" – see Norvig's Principles of AI Programming)
- SAT-solving tools highly evolved
- Several successful languages are closely related to propositional logic and SAT-solving, e.g., Satisfiability-Modulo Theories (SMT), Answer-Set Programming (ASP)
- Even tasks like **planning** get tackled using SAT-solving

Many shortcomings of propositional logic can be overcome by using **first-order logic (FOL)**, sometimes called predicate logic.

- Provides variables and quantification
- Interpretation assigns values from complex domains, called **universes**

However, in some situations even FOL may not offer sufficient expressivity ($\rightsquigarrow$ **higher-order logics**), or it may be overly complicated and computationally infeasible ($\rightsquigarrow$ specialised, less expressive logics, e.g., **temporal logic**)

Let countable infinite sets of symbols be given

1. **constant symbols** C
2. **variable symbols** V
3. **function symbols** F and **arity** $a_f : F \to \mathbb{N} \setminus \{0\}$
4. **predicate symbols** P and **arity** $a_p : P \to \mathbb{N} \setminus \{0\}$

$\rightsquigarrow$ Other definitions use functions $f \in F$ with $a_f(f) = 0$ as constants

Definition

The set $\mathcal{T}$ of **Terms** is defined inductively, terms are exactly...

1. variables $v \in V$
2. constants $c \in C$
3. expressions $f(t_1, \ldots, t_n)$ with $f \in F$, $n = a_f(f)$, and $t_1, \ldots t_n$ being terms

Definition

The set $\mathcal{F}$ of well-formed formula in FOL is defined inductively:

1. $\top \in \mathcal{F}$, $\bot \in \mathcal{F}$
2. $p(t_1, \ldots, t_n) \in \mathcal{F}$, if $p \in P$, $n = a_p(p)$, and $t_1, \ldots, t_n \in \mathcal{T}$
3. $(\ominus\phi) \in \mathcal{F}$, if $\phi \in \mathcal{F}, \ominus \in \{\neg\}$
4. $(\phi \oplus \psi) \in \mathcal{F}$, if $\phi, \psi \in \mathcal{F}, \oplus \in \{\wedge, \vee\}$
5. $(Qx.\phi) \in \mathcal{F}$, if $\phi \in \mathcal{F}$ and $x \in V, Q \in \{\forall, \exists\}$

Conventions of propositional logic to omit parentheses are applied, giving lowest precedence to **quantifiers** $\exists, \forall$.

Corollary

For every FOL formula there exists exactly one way to construct it using the syntax rules.

Definition

- We call a Formula an **atomic formula**, if it takes the form $\top$, $\bot$, or $p(t_1, \ldots, t_{a_p(p)}), p \in P$.
- A formula containing $\exists$ or $\forall$ is called a **quantified formula**.
- In a quantified formula $(\forall x.\phi)$ or $(\exists x.\phi)$, x is called the **quantified variable** and ϕ is the **scope** of the quantification.
- Occurrences of quantified variables within their scope are called **bound**.
- A variable $x \in V$ is called free in a formula ϕ, if it not bound in any scope.
- A **closed formula** or **sentence** is a formula without free variables.

Examples

Which of the following are WFF in FOL?

1. $\top \vee x$
2. $p(x, \top)$
3. $\forall x.(\exists y.(\forall z.p(x, y) \wedge p(y, z)))$
4. $\exists x.(\exists y.p(x \wedge y, x))$
5. $\exists x.p(x) \rightarrow y$

Which of the following are atomic, quantified, or closed formulae?

6. $p(x_1, x_2)$
7. $\exists x.p(x, y)$
8. $\exists x.(\exists y.p(x, y))$
9. $p(x)$

Definition

A **substitution** is a mapping $\sigma : X \to \mathcal{T}$. Notation $[t_1/x_1, \ldots t_n/x_n]$ for pairwise different x_i denotes the substitution with domain $\{x_1, \ldots, x_n\}$ and maps $x_i \mapsto t_i, i = 1, \ldots, n$.
Substitutions can be applied to terms and replace all occurrences of variables in the domain of the substitution by the respective terms.
We write $\sigma[t/x]$ to extend a substitution:

$$\sigma[t/x](y) := \begin{cases} t & x = y \\ \sigma(y) & \text{otherwise} \end{cases}$$

Substitutions are applied recursively to formulae

1. $\sigma(\top) = \top$, $\sigma(\bot) = \bot$
2. $\sigma\left(p(t_1, \ldots, t_n)\right) = p(\sigma(t_1), \ldots, \sigma(t_n))$
3. $\sigma\left(\neg\phi\right) = \neg\sigma(\phi)$
4. $\sigma\left(\phi \oplus \psi\right) = \sigma(\phi) \oplus \sigma(\psi), \oplus \in \{\wedge, \vee\}$
5. $\sigma\left(Qx.\phi\right) = Qx.\sigma[x/x](\phi), Q \in \{\exists, \forall\}$

Example

Let $a, b \in C$, $x, y, z \in V$, and $p \in P$:

$$[a/x, b/y]\left(p(a, x) \wedge (\exists y.p(y, x)) \vee (\forall z.p(x, z))\right)$$

Substitutions are applied recursively to formulae

1. $\sigma(\top) = \top$, $\sigma(\bot) = \bot$
2. $\sigma\left(p(t_1, \ldots, t_n)\right) = p(\sigma(t_1), \ldots, \sigma(t_n))$
3. $\sigma\left(\neg\phi\right) = \neg\sigma(\phi)$
4. $\sigma\left(\phi \oplus \psi\right) = \sigma(\phi) \oplus \sigma(\psi), \oplus \in \{\wedge, \vee\}$
5. $\sigma\left(Qx.\phi\right) = Qx.\sigma[x/x](\phi), Q \in \{\exists, \forall\}$

Example

Let $a, b \in C$, $x, y, z \in V$, and $p \in P$:

$$[a/x, b/y]\left(p(a, x) \wedge (\exists y.p(y, x)) \vee (\forall z.p(x, z))\right)$$

$$p(a, a) \wedge (\exists y.p(y, a)) \vee (\forall z.p(a, z)$$

Unlike propositional logic, FOL formulae are interpreted over arbitrary **universes**:

Definition

Let $\mathcal{U}$ be an non-empty, possibly infinite set, called **universe**. A function I is called an **interpretation** if it comprises the following elements:

1. for each constant $c \in C$, $I(c) \in \mathcal{U}$
2. for each function $f \in F$, $I(f) \in \mathcal{U}^{a_f(f)} \to \mathcal{U}$
3. for each predicate $p \in P$, $I(p) \in \mathcal{U}^{a_p(p)} \to \{\top, \bot\}$

The pair $(\mathcal{U}, I)$ is called a **structure**.

Definition

Given a universe $\mathcal{U}$, we call a mapping $X \to \mathcal{U}$ an **assignment** of variables. Assignments are a special form of substitutions.

In the following let $\mathcal{M} = (\mathcal{U}, I)$ be a structure and α be an assignment.

Definition

The value of a term with respect to assignment α, written I_α is defined as follows:

1. $I_\alpha(x) := \alpha(x), x \in X$
2. $I_\alpha(c) := I(c), c \in C$
3. $I_\alpha(f(t_1, \ldots, t_n)) := I(f)(I_\alpha(t_1), \ldots, I_\alpha(t_n)), f \in F, a_f(f) = n$

Definition

Let ϕ, ψ be closed formulas. The validity of a formula with respect to $\mathcal{M} = (\mathcal{U}, I)$ and α is inductively defined as follows (note that for a formula without free variables, α may be empty):

$$
\begin{aligned}
I_{\mathcal{M},\alpha}(\top) &:= \top \\
I_{\mathcal{M},\alpha}(\bot) &:= \bot \\
I_{\mathcal{M},\alpha}(p(t_1, \ldots, t_{a_p(p)})) &:= I(p)(I_\alpha(t_1), \ldots, I_{\alpha(t_{a_p(p)})}) \\
I_{\mathcal{M},\alpha}(\neg\phi) &:= \begin{cases} \top & \text{if } I_{\mathcal{M},\alpha}(\phi) = \bot \\ \bot & \text{otherwise} \end{cases} \\
I_{\mathcal{M},\alpha}(\phi \vee \psi) &:= \begin{cases} \top & \text{if } I_{\mathcal{M},\alpha}(\phi) = \top \text{ or } I_{\mathcal{M},\alpha}(\psi) = \top \\ \bot & \text{otherwise} \end{cases}
\end{aligned}
$$

$$I_{\mathcal{M},\alpha}(\phi \wedge \psi) := \begin{cases} \top & \text{if } I_{\mathcal{M},\alpha}(\phi) = \top \text{ and } I(\psi) = \top \\ \bot & \text{otherwise} \end{cases}$$

$$I_{\mathcal{M},\alpha}(\exists x.\phi) := \begin{cases} \top & \text{if } I_{\mathcal{M},\alpha[a/x]}(\phi) \text{ for some } a \in \mathcal{U} \\ \bot & \text{otherwise} \end{cases}$$

$$I_{\mathcal{M},\alpha}(\forall x.\phi) := \begin{cases} \top & \text{if } I_{\mathcal{M},\alpha[a/x]}(\phi) \text{ for every } a \in \mathcal{U} \\ \bot & \text{otherwise} \end{cases}$$

FOL vs. Propositional Logic

Use of universe $\mathcal{U}$ and predicates P imply significant differences between the logics, in particular with respect to qualification:

$$\exists x.p(x) \sim \bigvee_{x \in \mathcal{U}} p(x)$$

$$\forall x.p(x) \sim \bigwedge_{x \in \mathcal{U}} p(x)$$

- $\mathcal{U}$ may be infinite, possible uncountably infinite: $\bigvee_{x \in \mathcal{U}}, \bigwedge_{x \in \mathcal{U}}$ are not possible to construct!
- Model checking requires strong restrictions on the structure in order to become feasible. The problem needs to be encoded finitely, to start with!
- Example: without restriction we could formulate Fermat's last theorem in FOL and ask whether the universe of integers provides a model to it, thus effectively solving the theorem!
- FOL overcomes the shortcomings of propositional logic for KR discussed earlier.

Model Checking

Recall your assignment to implement a method for retrieving facts from a knowledge base Knowledge base facts:

- `(is-a river waterway)`
- `(is-a canal waterway)`
- `(instance-of regnitz river)`
- ...

Queries:

- `(and (is-a ?R river) (joins ?R main))`
- `(or (is-a ?R river) (is ?R canal))`

You have implemented a model checking for a fragment of FOL over a finite universe (the knowledge base)!

Validity in FOL

Our definitions for propositional logic are general enough to apply to FOL as well!
However, note the subtle difference of interpretations in FOL which has to interpret function and predicate *symbols*: Is the following formula...

$$\exists x. > (\sin(x), 1) \qquad [= \exists x. \sin(x) > 1]$$

1. satisifiable,
2. a tautology,
3. or a contradiction?

Our definitions for propositional logic are general enough to apply to FOL as well!
However, note the subtle difference of interpretations in FOL which has to interpret function and predicate *symbols*: Is the following formula...

$$\exists x.\, >(\sin(x), 1) \qquad [= \exists x.\, \sin(x) > 1]$$

1. **satisfiable,**
2. a tautology,
3. or a contradiction?

It is satisfiable since interpretations choose concrete functions and predicates for symbols $\sin, >$! We thus could choose $>:\equiv \top$ or $<:\equiv \bot$ to make the formula evaluate to $\top$ or $\bot$.

Definition

The task of **axiomatisation** is to pin down the intended semantics of a predicate or function using logic statements.

Example

To axiomatise a predicate $\mathrm{married}$ we could write

$$\forall x.\,(\forall y.\,(\mathrm{married}(x,y) \rightarrow \mathrm{married}(y,x))) \wedge \forall x.\,(\neg\mathrm{married}(x,x))$$

Domain Theory

The set of statements for axiomatising all predicates and functions required to represent some domain is called the **domain theory**.

- Developing a suitable domain theory is a fundamental task in AI and computer science in general
- Different axiomatisations of the same domain theory are possible
    - Some of which may allow for more efficient reasoning techniques

Summary

- Syntax and validity semantics for defining a logic
  - Inductively defined well-formed formulae
  - Inductively defined semantics using an interpretation function
- Reasoning problems: Entailment (deduction), model checking, abduction, induction
- Inference rules for reasoning
- Propositional logic and FOL as two fundamental logics with Boolean validity
- KR in logic: Axiomatisation

Literature

- S. Russel & P. Norvig (2010, 3rd edition). Artificial Intelligence: A Modern Approach, Chapter 7 "First-Order Logic"
- M. Fitting (1996). First-Order Logic and Automated Theorem Proving (Second Edition). Springer
- Uwe Schöning (2001). Theoretische Informatik–kurzgefasst. Spektrum, Akademischer Verlag, Heidelberg, ISBN 3-8274-1099-1
- Mordechai Ben-Ari (2012). Mathematical Logic for Computer Science, Springer (online accessible from the University!)
- David Poole and Alan Mackworth (2010). Artificial Intelligence: Foundations of computational agents, Cambridge University Press, Chapter 5, `http://artint.info/html/ArtInt.html`