Artificial Intelligence 1

Prof. Dr. Frank Hopfgartner
Dr. Matthias Horbach

Institute for Web Science and Technologies (WeST)
University of Koblenz

Overview

First-order Logic - Overview

- First-order logic is an extension of propositional logic that adds *relations*, *functions* and *objects* (Hilbert, Ackermann 1928)
- The foundation of a first-order logic are atomic statements such as

knows(John, Mary)	=	"John knows Mary"
brotherOf(Carl, Dave)	=	"Carl is the brother of Dave"

- As in propositional logic, atomic statements can be combined with $\wedge$ (AND), $\vee$ (OR), and $\neg$ (NOT) in order to obtain more complex statements
- Additionally we have quantification $\forall$, $\exists$
- The following introduction to first-order logic is structured analogously to the introduction of propositional logic

First-order logic - syntax 1/6

Definition

A *first-order signature* Σ is a triple $\Sigma = (U, P, F)$ with

- U is a set of constant symbols (objects)
- P is a set of predicate symbols
- F is a set of function symbols (functors)

For every $e \in P \cup F$ let $ar(e) \in \mathbb{N}$ denote the *arity* of e (=number of parameters).

Example

Let $\Sigma_1 = (U_1, P_1, F_1)$ with

$$U_1 = \{\text{john}, \text{carl}, \text{mary}, \text{dave}\} \qquad P_1 = \{\text{knows}, \text{brotherOf}\}$$
$$F_1 = \{\text{fatherOf}\}$$

and $ar(\text{knows}) = ar(\text{brotherOf}) = 2$, $ar(\text{fatherOf}) = 1$.

First-order logic - syntax 2/6

Definition

Let $\Sigma = (U, P, F)$ be a first-order signature and V a set of variables. The set of *terms* $\text{Terms}(\Sigma, V)$ is the minimal set T with the following properties:

1. $U \subseteq T$
2. $V \subseteq T$
3. For all $f \in F$ with $ar(f) = k$ and $t_1, \ldots, t_k \in T$, $f(t_1, \ldots, t_k) \in T$

Remark

To differentiate between variables and constants we will use the convention that variables start with an uppercase letter and constants with a lowercase letter.

Example

Let $\Sigma_1 = (U_1, P_1, F_1)$ with

$$U_1 = \{\text{john}, \text{carl}, \text{mary}, \text{dave}\} \qquad P_1 = \{\text{knows}, \text{brotherOf}\}$$
$$F_1 = \{\text{fatherOf}\}$$

with $ar(\text{knows}) = ar(\text{brotherOf}) = 2$ and $ar(\text{fatherOf}) = 1$. Let $V_1 = \{X, Y, Z\}$ be a set of variables. Then

$$\text{john}$$
$$\text{X}$$
$$\text{fatherOf}(mary)$$
$$\text{fatherOf}(Y)$$
$$\text{fatherOf}(\text{fatherOf}(mary))$$

are terms in $\text{Terms}(\Sigma_1, V_1)$.

First-order logic - syntax 4/6

Definition

Let $\Sigma = (U, P, F)$ be a first-order signature, V a set of variables, $p \in P$ a predicate symbol with $ar(p) = k$, and $t_1, \ldots, t_k \in \text{Terms}(\Sigma, V)$. Then

$$p(t_1, \ldots, t_k)$$

is a *first-order atom*.

Example

Let Σ_1 and V_1 be as before. Then

$$\text{knows}(\text{john}, \text{mary})$$
$$\text{brotherOf}(\text{john}, \text{fatherOf}(mary))$$
$$\text{brotherOf}(\text{X}, \text{fatherOf}(\text{fatherOf}(Y)))$$

are first-order atoms.

Definition (Syntax)

Let $\Sigma = (U, P, F)$ be a first-order signature and V a set of variables. The *first-order language* $\mathcal{L}(\Sigma, V)$ is the minimal set $\mathcal{L}$ with

1. If $p(t_1, \ldots, t_k)$ is a first-order atom then $p(t_1, \ldots, t_k) \in \mathcal{L}$
2. $\top, \bot \in \mathcal{L}$ (tautology and contradiction) and
3. for all $\phi, \psi \in \mathcal{L}$ and $X \in V$
   - 3.1 $\phi \wedge \psi \in \mathcal{L}$
   - 3.2 $\phi \vee \psi \in \mathcal{L}$
   - 3.3 $\neg\phi \in \mathcal{L}$
   - 3.4 $\forall X : \phi \in \mathcal{L}$
   - 3.5 $\exists X : \phi \in \mathcal{L}$

First-order logic - syntax 6/6

Remark

Again, we are assuming that parentheses "(" and ")" are part of the syntax. Implication $\Rightarrow$ and equivalence $\Leftrightarrow$ is also defined as in propositional logic.

Example

Let Σ_1 and V_1 be as before. Then

$$\phi_1 = \mathrm{knows}(\mathrm{john}, \mathrm{mary})$$
$$\phi_2 = \neg\mathrm{brotherOf}(\mathrm{john}, \mathrm{fatherOf}(mary)) \vee \mathrm{knows}(\mathrm{john}, \mathrm{mary})$$
$$\phi_3 = \forall X : \exists Y : \mathrm{knows}(X, Y)$$

are formulas in $\mathcal{L}(\Sigma_1, V_1)$.

First-order logic - syntax (summary)

Let $\Sigma = (U, P, F)$ be a *first-order signature* and V a set of variables. The set U contains *constants*, P *predicates*, and F *functors*. Each $p \in P$ and $f \in F$ has *arity* $ar(p)$, $ar(f)$.

- a **term** is a symbol t such that either:
  - $t \in U$
  - $t \in V$
  - $t = f(t_1, \ldots, t_n)$ (where $t_1, \ldots, t_n$ are terms and $n = ar(f)$)
- an **atom** is a symbol $p(t_1, \ldots, t_n)$
  (where $p \in P$, $t_1, \ldots, t_n$ are terms, and $n = ar(p)$)
- a **formula** is either:
  - an atom
  - $\phi \wedge \psi$, $\phi \vee \psi$ (where ϕ, ψ are formulas)
  - $\neg\phi$ (where ϕ is a formula)
  - $\forall X : \phi$ (where $X \in V$ and ϕ is a formula)
  - $\exists X : \phi$ (where $X \in V$ and ϕ is a formula)
  - $\top$ or $\bot$

First-order logic - semantics 1/8

- Meaning is assigned to first-order formulas through interpretations
- Every interpretation represents a *possible* world by enumerating everything that is true in this world

Definition

Let $\Sigma = (U, P, F)$ be a first-order signature with $P = \{p_1, \ldots, p_n\}$ and $F = \{f_1, \ldots, f_m\}$. A *first-order interpretation* I is a tuple $I = (U_I, f_I^U, P_I, F_I)$ with

1. U_I is a non-empty set of objects (the *universe* or *domain*)
2. f_I^U is a function $f_I^U : U \to U_I$
3. P_I is a set of relations $P_I = \{p_1^I, \ldots, p_n^I\}$ with $p_i^I \subseteq U_I^k$ for $ar(p_i) = k$ $(i = 1, \ldots, n)$ and
4. F_I is a set of functions $F_I = \{f_1^I, \ldots, f_m^I\}$ with $f_i^I : U_I^k \to U_I$ for $ar(f_i) = k$ $(i = 1, \ldots, m)$

Example

Let Σ_1 be as before. Define $I = (U_I, f_I^U, P_I, F_I)$ through

$$U_I = \{\text{“John Johnsson”}, \text{“Carl Carlson”}, \text{“Mary Meyer”}, \text{“Dave Davidson”}\}$$

$$f_I^U(\text{john}) = \text{“John Johnsson”} \quad \ldots$$

$$P_I = \{\text{knows}^I, \text{brotherOf}_I\}$$

$$\text{with knows}^I = \{(\text{“John Johnsson”}, \text{“Carl Carlson”}), (\text{“Carl Carlson”}, \text{“Dave Davidson”})\}$$

$$\text{brotherOf}^I = \{(\text{“Mary Meyer”}, \text{“Dave Davidson”})\}$$

$$F_I = \{\text{fatherOf}^I\}$$

$$\text{with fatherOf}^I(\text{“Mary Meyer”}) = \text{“Carl Carlson”'} \quad \ldots$$

First-order logic - semantics 3/8

What about variables?

Definition

Let $\Sigma = (U, P, F)$ be a first-order signature, $I = (U_I, f_I^U, P_I, F_I)$ a first-order interpretation and V a set of variables. A *variable assignment* VA for V wrt. Σ and I is a function $VA : V \rightarrow U_I$.

Example

Let Σ_1, V_1, and I as before. Then $VA_1 : V_1 \rightarrow U_I$ defined via

$$\begin{aligned} VA(X) &= \text{“Carl Carlsson”} \\ VA(Y) &= \text{“Mary Meyer”} \\ VA(Z) &= \text{“Carl Carlsson”} \end{aligned}$$

is a variable assignment.

First-order logic - semantics 4/8

Given an interpretation $I = (U_I, f_I^U, P_I, F_I)$ and variable assignment $VA : V \to U_I$ we abbreviate

$$VA(c) = f_I^U(c) \qquad \text{if } c \in U$$
$$VA(f(t_1, \ldots, t_k)) = f^I(VA(t_1), \ldots, VA(t_k))$$

We now turn to the satisfaction relation of first-order logic

Definition

Let $\Sigma = (U, P, F)$ be a first-order signature, V a set of variables, $I = (U_I, f_I^U, P_I, F_I)$ a first-order interpretation, $X \in V$, VA a variable assignment, and $\phi, \psi \in \mathcal{L}(\Sigma, V)$. Define inductively

$$
\begin{aligned}
(I, VA) &\models p(t_1, \ldots, t_k) && \text{iff } (VA(t_1), \ldots, VA(t_k)) \in p^I \\
(I, VA) &\models \phi \vee \psi && \text{iff } (I, VA) \models \phi \text{ or } (I, VA) \models \psi \\
(I, VA) &\models \phi \wedge \psi && \text{iff } (I, VA) \models \phi \text{ and } (I, VA) \models \psi \\
(I, VA) &\models \neg\phi && \text{iff } (I, VA) \not\models \phi
\end{aligned}
$$

First-order logic - semantics 6/8

$$
\begin{array}{ll}
(I, VA) \models \forall X : \phi & \text{iff for all } VA' \text{ with } VA' = VA \text{ except possibly} \\
& VA'(X) \neq VA(X) \text{ we have } (I, VA') \models \phi \\
(I, VA) \models \exists X : \phi & \text{iff for at least one } VA' \text{ with } VA' = VA \text{ except possibly} \\
& VA'(X) \neq VA(X) \text{ we have } (I, VA') \models \phi
\end{array}
$$

Define additionally $(I, VA) \models \top$ and $(I, VA) \not\models \bot$ for every I and VA and

$$I \models \phi \qquad \text{iff for all } VA \text{ we have } (I, VA) \models \phi$$

Remark

The terms *model*, *entailment* and *equivalence* are defined as in propositional logic.

Consider

$$\phi = \forall X : \exists Y : \mathrm{knows}(X, Y)$$

and the interpretation $I = (U_I, f_I^U, P_I, F_I)$ defined via

$$\begin{aligned}
U_I &= \{\text{"John Johnsson"}, \text{"Carl Carlson"}, \text{"Mary Meyer"}\} \\
f_I^U(\mathrm{john}) &= \text{"John Johnsson"} \quad \ldots \\
P_I &= \{\mathrm{knows}^I\} \\
&\qquad \text{with } \mathrm{knows}^I = \{(\text{"John Johnsson"}, \text{"Carl Carlson"}), \\
&\qquad\qquad (\text{"Carl Carlson"}, \text{"John Johnsson"}) \\
&\qquad\qquad (\text{"Mary Meyer"}, \text{"Carl Carlson"'})\} \\
F_I &= \{\}
\end{aligned}$$

Is it true that $I \models \phi$?

$$
\begin{array}{lll}
\multicolumn{3}{l}{I \models \forall X : \exists Y : \mathrm{knows}(X, Y)} \\
\iff & \text{for all } VA : (I, VA) \models \forall X : \exists Y : \mathrm{knows}(X, Y) \\
\iff & \text{for all } VA : \text{ for all } VA' \text{ with } VA' = VA \\
 & \qquad \text{except possibly } VA'(X) \neq VA(X) \\
 & \qquad \text{we have } (I, VA') \models \exists Y : \mathrm{knows}(X, Y) \\
\iff & \text{for all } VA : (I, VA) \models \exists Y : \mathrm{knows}(X, Y) \\
\iff & \text{for all } VA : \text{for some } VA' \text{ with } VA' = VA \\
 & \qquad \text{except possibly } VA'(Y) \neq VA(Y) \\
 & \qquad \text{we have } (I, VA') \models \mathrm{knows}(X, Y)
\end{array}
$$

First-order logic - semantics 8/8

We have either

$$VA(X) = \text{“John Johnsson” or}$$
$$VA(X) = \text{“Carl Carlson” or}$$
$$VA(X) = \text{“Mary Meyer”}.$$

Let's try $VA(X) =$ “John Johnsson”. Then define VA' with $VA'(Y) =$ “Carl Carlson” and $VA' = VA$ otherwise, consider:

$$\begin{aligned}
&(I, VA') \models \text{knows}(X, Y) \\
&\iff \quad (VA'(X), VA'(Y)) \in \text{knows}^I \\
&\iff \quad (\text{“John Johnsson”}, \text{“Carl Carlson”}) \in \text{knows}^I \\
&\iff \text{TRUE}
\end{aligned}$$

For $VA(X) =$ “Carl Carlson” and $VA(X) =$ “Mary Meyer” we can find an appropriate VA' as well.

It follows that $I \models \forall X : \exists Y : \text{knows}(X, Y)$.

First-order logic - calculus, implementation

- Automatic reasoning with first-order logic is also called "(automatic) theorem proving"
- For general first-order logic, the problem "is there an I with $I \models \phi$?" *is undecidable* [Turing, 1937]
- (Impossible to construct an algorithm that always leads to a correct yes-or-no answer)
- There is a wide range of syntactic restrictions of first-order logic (such as description logics) which are decidable.
- Implementations/systems for first-order logic:
  - EProver: `http://wwwlehre.dhbw-stuttgart.de/~sschulz/E/E.html`
  - Vampire: `https://vprover.github.io`
  - Hyper: `https://userpages.uni-koblenz.de/~obermaie/hyper/hyper.htm`

First-order logic and propositional logic

- First-order logic is a *faithful generalisation* of propositional logic
- Any propositional logic can be embedded into a first-order logic
- If At is a propositional signature then $\Sigma_{\mathrm{At}} = (\emptyset, \mathrm{At}, \emptyset)$, with $ar(a) = 0$ for every $a \in \mathrm{At}$, is the corresponding first-order signature
- Then $\mathcal{L}(\mathrm{At}) \subseteq \mathcal{L}(\Sigma, \emptyset)$
- All other notions (satisfaction, entailment) are then equivalent

$\rightarrow$ whenever we can use propositional logic formulas we can also use first-order logic

Chapter 2.1: Classical logics

Summary

Chapter 2.1: Summary

- Syntax and semantics of formal logics
- Propositional logic
  - Syntax: signature, $\wedge$, $\vee$, $\neg$
  - Semantics: interpretations, models, satisfaction relation
  - Inference, equivalence
  - Satisfiability, CNF, SAT
- First-order logic
  - Syntax: signature, variables, terms, atoms, $\wedge$, $\vee$, $\neg$, $\forall$, $\exists$
  - Semantics: interpretations, variable assignments, models, satisfaction relation
  - Automatic theorem proving

Overview

First-order logic and Horn logic 1/2

- Drawback of first-order logic: too expressive and therefore not decidable in general
- Wanted: Subset of first-order logic that is expressive but decidable
  - For modelling (ontologies, semantic web): description logic
  - For programming: Horn logic
- Horn logic is the foundation of most logical programming languages such as Prolog

First- order logic and Horn logic 2/2

- Recall definition *clause* (propositional logic): a disjunction of literals
- Definition is the same for first-order logic, example:

$$s(a, b) \vee \neg t(a) \vee \neg r(b, c)$$

Definition

A *Horn clause* is a clause with a most one positive literal.

Examples:

- $s(a, b) \vee \neg t(a)$ is a Horn clause.
- $p(X) \vee \neg t(Y) \vee \neg u(X, Y)$ is a Horn clause.
- $q \vee \neg v \vee \neg x$ is a Horn clause.
- $s(a, b) \vee t(a) \vee \neg r(b, c)$ is not a Horn clause.

- A Horn clause $h \vee \neg b_1 \vee \ldots \vee \neg b_n$ is logically equivalent to an implication

$$h \Leftarrow (b_1 \wedge \ldots \wedge b_n)$$

- Rules (=implications) are the building blocks of Prolog.
- The syntax of Prolog is very similar to the syntax of first-order logic with few exceptions

First-order logic	Prolog	
$\Leftarrow$	`:-`	Rule
$\wedge$	`,`	Conjunction
$\vee$	`;`	Disjunction
$\neg$	`not`	Negation*

*Not exactly the same

Prolog programs 1/4

Definition

A Prolog program P consists of

1. a data base D
2. and a rule base R

Definition

A data base D is a set of first-order atoms without variables (and without functors). Every atom is terminated by a full stop ".".

Prolog programs 2/4

Example

```
childOf(maria, claudia).
childOf(claudia, berta).
female(maria).
female(claudia).
female(berta).
```

Convention: variables start with an uppercase letter (or with _) and constants start with a lowercase letter. Predicate symbols also start with a lowercase letter.

Prolog programs 3/4

Definition

A rule base R is a set of Horn clauses of the form

```
H :- B1,...,BN.
```

with `H,B1,...,BN` being first-order atoms (without functors).

Example

```
grandchildOf(X,Z) :- childOf(X,Y), childOf(Y,Z).
mother(X) :- female(X), childOf(_,X).
male(X) :- not(female(X)).
```

Remark: _ is an anonymous variable and can be used if the actual value is not important.

Evaluation of Prolog programs 1/3

- Prolog programs represent a knowledge base (facts and rules)
- Prolog programs are "executed" by asking specific *queries*

Definition

A *Prolog query* (also *goal*) is a first-order atom (possibly with variables) with a prefixed `?-`.

Example

- `?- female(maria)` *Is Maria female?*
- `?- female(X)` *For what X is it true that X is female?*
- `?- grandchildOf(maria, X)` *For what X is it true that $maria$ is a grandchild of X?*

Evaluation of Prolog programs 2/3

- Executing a query means to determine whether the program entails the query.
- If query contains variables: determine *for which assignments of these variables* the program entails the query (could be none, could be more than one).
- Semantics of Prolog follows the *Closed World Assumption* (Statements that are not entailed are assumed to be false.)
- Execution of queries via depth-first backward chaining, where rules are applied *in the order in which they occur in the program*.

Evaluation of Prolog programs 3/3

Data base:

```
childOf(maria, claudia). childOf(claudia, berta).
female(maria). female(claudia). female(berta).
```

Rule base:

```
grandchildOf(X,Z) :- childOf(X,Y), childOf(Y,Z).
mother(X) :- female(X), childOf(_,X).
male(X) :- not(female(X)).
```

- `?- childOf(maria, berta)` No (Not entailed)
- `?- female(carla)` No (Carla is not mentioned)
- `?- male(dieter)` Yes (Why?)
- `?- mother(X)` Yes, with multiple assignments for `X`:
  - `X = claudia`
  - `X = berta`

Prolog interpreter: SWI-Prolog 1/4

- Prolog is an *interpreted* programming language
- We use *SWI-Prolog* (`http://www.swi-prolog.org/`) for this lecture and for the tutorials

Installation of SWI-Prolog in Unix/Linux

```
$ sudo apt-get install swi-prolog
```

Prolog interpreter: SWI-Prolog 2/4

simpleFamily.pl:

```
childOf(maria, claudia).
childOf(claudia, berta).
female(maria).
female(claudia).
female(berta).
grandchildOf(X,Z) :- childOf(X,Y), childOf(Y,Z).
mother(X) :- female(X), childOf(_,X).
male(X) :- not(female(X)).
```

Execution using SWI-Prolog

```
$ swipl simpleFamily.pl
```

Prolog interpreter: SWI-Prolog 3/4

Interpreter:

```
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 6.6.6)
Copyright (c) 1990-2013 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?-
```

Prolog interpreter: SWI-Prolog 4/4

Asking queries:

```
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 6.6.6)
Copyright (c) 1990-2013 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- grandchildOf(maria, X).
X = berta.

?-
```

Interpreter is terminated by entering "halt."

Arithmetic operators 1/3

- Prolog has built-in functionalities for arithmetics, natural numbers are predefined constants
  - Addition: `X+2`
  - Subtraction: `Y-5`
  - Multiplication: `3*5`
  - Division: `12/4`
- Comparison operators
  - Syntactic equality: `=`
  - Semantic equality/inequality: `=:=`
  - less than/greater than: `>,<,>=,=<`

Example:

```
?- 3+4 = 1+6.
false.

?- 3+X = Y+4.
X = 4,
Y = 3.
```

Arithmetic operators 2/3

More examples:

```
?- 2*5 =:= 10.
true.

?- 3*3 < 10.
true.
```

arithmetics1.pl:

```
isSumOf(X,Y,Z) :- X =:= Y+Z.
```

```
?- isSumOf(10,6,4).
true.

?- isSumOf(7,1,3).
false.
```

Arithmetic operators 3/3

- Value assignment is realised via `is`: X is *Arithmetic expression*

```
?- X is (2*3+4).
X=10.
```

arithmetics2.pl:

```
even(0).
even(X) :- X > 0, Y is (X-1), odd(Y).
odd(X) :- not(even(X)).
```

```
?- even(2).
true.

?- even(7).
false.
```

Termination

Note: Prolog program may not terminate.

Be cautious when implementing, in particular with recursive definitions and arithmetic expressions:

```
t(X) :- t(X+1).
```

```
?- t(1).
ERROR: Out of global stack
```