

# FMFP - Lecture Notes Week 2

Ruben Schenk, [ruben.schenk@inf.ethz.ch](mailto:ruben.schenk@inf.ethz.ch)

March 16, 2022

## 0.1 First-Order Logic

### 0.1.1 Syntax

In **first-order logic** we have two syntactic categories: **terms** and **formulae**.

A **signature** consists of a set of function symbols  $\mathcal{F}$  and a set of predicate symbols  $\mathcal{P}$ . We write  $f^k$  (or  $p^k$ ) to indicate function symbol  $f$  (or predicate symbol  $p$ ) has arity  $k \in \mathcal{N}$ . Constants are 0-ary function symbols.

Now, let  $\mathcal{V}$  be a set of variables. Then:

**Definition:** *Term*, the **terms of first-order logic**, is the smallest set where:

1.  $x \in \text{Term}$  if  $x \in \mathcal{V}$ , and
2.  $f^n(t_1, \dots, t_n) \in \text{Term}$  if  $f^n \in \mathcal{F}$  and  $t_i \in \text{Term}$ , for all  $1 \leq i \leq n$ .

**Definition:** *Form*, the **formulae of first-order logic**, is the smallest set where:

1.  $\perp \in \text{Form}$ ,
2.  $p^n(t_1, \dots, t_n) \in \text{Form}$  if  $p^n \in \mathcal{P}$  and  $t_j \in \text{Term}$ , for all  $1 \leq j \leq n$ ,
3.  $A \circ B \in \text{Form}$  if  $A \in \text{Form}$ ,  $B \in \text{Form}$ , and  $\circ \in \{\wedge, \vee, \rightarrow\}$ , and
4.  $Qx.A \in \text{Form}$  if  $A \in \text{Form}$ ,  $x \in \mathcal{V}$ , and  $Q \in \{\forall, \exists\}$ .

Each occurrence of each variable in a formula is either **bound** or **free**. A variable occurrence  $x$  in a formula  $A$  is **bound** if  $x$  occurs within a subformula  $B$  of  $A$  of the form  $\exists x.B$  or  $\forall x.B$ .

### 0.1.2 Binding and $\alpha$ -conversion

Names of bound variables are irrelevant, they just encode the binding structure. We can rename *bound* variables, this process is called  **$\alpha$ -conversion**.

It is important to note that the renaming must *preserve the binding structure!*

Some notes on bindings and parentheses:

- $\wedge$  binds stronger than  $\vee$ , and  $\vee$  binds stronger than  $\rightarrow$ .
- $\rightarrow$  associates to the right, *land* and *lor* to the left.
- Negation binds stronger than binary operators.
- Quantifiers extend to the right as far as possible: to the end of the line or ')'

$$\begin{array}{c}
 \frac{\left( p \vee \left( \underline{q \wedge (\neg r)} \right) \right)}{p \rightarrow \left( \underline{(q \vee p) \rightarrow r} \right)} \rightarrow (p \vee q) \\
 \frac{p \wedge \left( \forall x. \left( \underline{q(x) \vee r} \right) \right)}{\neg \left( \forall x. \left( p(x) \wedge \left( \forall x. \left( \underline{(q(x) \wedge r(x)) \wedge s} \right) \right) \right) \right)}
 \end{array}$$

### 0.1.3 Semantics

A **structure** is a pair  $\mathcal{S} = \langle U_{\mathcal{S}}, I_{\mathcal{S}} \rangle$  where  $U_{\mathcal{S}}$  is a nonempty set, the **universe**, and  $I_{\mathcal{S}}$  is a mapping where:

1.  $I_{\mathcal{S}}(p^n)$  is an  $n$ -ary relation on  $U_{\mathcal{S}}$ , for  $p^n \in \mathcal{P}$ , and
2.  $I_{\mathcal{S}}(f^n)$  is an  $n$ -ary (total) function on  $U_{\mathcal{S}}$ , for  $f^n \in \mathcal{F}$

As a shorthand, we write  $p^{\mathcal{S}}$  for  $I_{\mathcal{S}}(p)$  and  $f^{\mathcal{S}}$  for  $I_{\mathcal{S}}(f)$ .

An **interpretation** is a pair  $\mathcal{I} = \langle \mathcal{S}, v \rangle$ , where  $\mathcal{S} = \langle U_{\mathcal{S}}, I_{\mathcal{S}} \rangle$  is a structure and  $v : \mathcal{V} \rightarrow U_{\mathcal{S}}$  is a valuation.

The **value** of a term  $t$  under the interpretation  $\mathcal{I} = \langle \mathcal{S}, v \rangle$  is written as  $\mathcal{I}(t)$  and defined by:

1.  $\mathcal{I}(x) = v(x)$ , for  $x \in \mathcal{V}$ , and
2.  $\mathcal{I}(f(t_1, \dots, t_n)) = f^{\mathcal{S}}(\mathcal{I}(t_1), \dots, \mathcal{I}(t_n))$ .

**Satisfiability** is the smallest relation  $\models \subseteq \text{Interpretations} \times \text{Form}$  satisfying:

- $\langle \mathcal{S}, v \rangle \models p(t_1, \dots, t_n)$  if  $(\mathcal{I}(t_1), \dots, \mathcal{I}(t_n)) \in p^{\mathcal{S}}$ , where  $\mathcal{I} = \langle \mathcal{S}, v \rangle$ .
- $\langle \mathcal{S}, v \rangle \models \forall x.A$  if  $\langle \mathcal{S}, v[x \rightarrow a] \rangle \models A$ , for all  $a \in U_{\mathcal{S}}$ .
- $\langle \mathcal{S}, v \rangle \models \exists x.A$  if  $\langle \mathcal{S}, v[x \rightarrow a] \rangle \models A$ , for some  $a \in U_{\mathcal{S}}$ .

Here,  $v[x \rightarrow a]$  is the valuation  $v'$  identical to  $v$ , except that  $v'(x) = a$ .

When  $\langle \mathcal{S}, v \rangle \models A$ , we say that  $A$  is *satisfied with respect to*  $\langle \mathcal{S}, v \rangle$  or *language*  $\langle \mathcal{S}, v \rangle$  is a **model** of  $A$ . Note that if  $A$  does not have free variables, satisfaction does not depend on the valuation  $v$ . We write  $\mathcal{S} \models A$ . When every interpretation is a model, we write  $\models A$  and say that  $A$  is **valid**.

$A$  is **satisfiable** if there is at least one model for  $A$  (and said to be **contradictory** otherwise).

**Example:** Consider the following examples:

- $\forall x. \exists y. y * 2 = x$  satisfied w.r.t. rationals.
- $\forall x. \forall y. x < y \rightarrow \exists z. x < z \wedge z < y$  satisfied w.r.t. any dense order.
- $\exists x. x \neq 0$  satisfied w.r.t. structures  $\mathcal{S}$  with  $\geq 2$  elements in  $U_{\mathcal{S}}$ .
- $(\forall x. p(x, x)) \rightarrow p(a, a)$  is valid.

### 0.1.4 Substitution

**Substitution** describes the process of replacing in  $A$  all occurrences of a free variable  $x$  with some term  $t$ . We write  $A[x \rightarrow t]$  to indicate the substitution.

**Example:**

$$\begin{aligned} A &\equiv \exists y. y * x = x * z \\ A[x \rightarrow 2 - 1] &\equiv \exists y. y * (2 - 1) = (2 - 1) * z \\ A[x \rightarrow z] &\equiv \exists y. y * z = z * z \end{aligned}$$

All free variables of  $t$  must still be free in  $A[x \rightarrow t]$ . Avoid *capture*! If necessary,  $\alpha$ -convert  $A$  before substitution.

### 0.1.5 Universal Quantification

The rules are as follows:

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x. A} \forall\text{-I}^* \quad \frac{\Gamma \vdash \forall x. A}{\Gamma \vdash A[x \mapsto t]} \forall\text{-E}$$

The side condition  $*$  is:  $x$  must not be free in any assumption in  $\Gamma$ .

### 0.1.6 Existential Quantification

The rules are as follows:

$$\frac{\Gamma \vdash A[x \mapsto t]}{\Gamma \vdash \exists x. A} \exists\text{-I} \quad \frac{\Gamma \vdash \exists x. A \quad \Gamma, A \vdash B}{\Gamma \vdash B} \exists\text{-E}^*$$

The side condition  $*$  is:  $x$  is neither free in  $B$  nor free in  $\Gamma$ .

## 0.2 Equality

**Equality** is a logical symbol with associated proof rules. One speaks of *first-order logic with equality* rather than equality just being another predicate:

- Extended language:  $t_1 = t_2 \in \text{Form}$  if  $t_1, t_2 \in \text{Term}$
- extended definition of semantic entailment  $\models$ :  $\mathcal{I} \models t_1 = t_2$  if  $\mathcal{I}(t_1) = \mathcal{I}(t_2)$

Equality is an *equivalence* relation with the following rules:

$$\frac{}{\Gamma \vdash t = t} \text{ref} \quad \frac{\Gamma \vdash t = s}{\Gamma \vdash s = t} \text{sym} \quad \frac{\Gamma \vdash t = s \quad \Gamma \vdash s = r}{\Gamma \vdash t = r} \text{trans}$$

And equality is also a *congruence* on terms and all definable relations:

$$\frac{\Gamma \vdash t_1 = s_1 \quad \dots \quad \Gamma \vdash t_n = s_n}{\Gamma \vdash f(t_1, \dots, t_n) = f(s_1, \dots, s_n)} \text{cong}_1$$

$$\frac{\Gamma \vdash t_1 = s_1 \quad \dots \quad \Gamma \vdash t_n = s_n \quad \Gamma \vdash p(t_1, \dots, t_n)}{\Gamma \vdash p(s_1, \dots, s_n)} \text{cong}_2$$

## 0.3 Correctness

**Correctness** is important! But what does correctness mean? What properties should hold?

- *Termination*: Important for many, but not all, programs.
- *Functional behavior*: Function should return "correct" value.

### 0.3.1 Termination

If  $f$  is defined in terms of functions  $g_1, \dots, g_k$  ( $g_i \neq f$ ), and each  $g_i$  terminates, then so does  $f$ . The problem we encounter here is *recursion*, i.e. when some  $g_i = f$ .

A sufficient condition for termination is that arguments must be smaller along a well-founded order on function's domain:

- An order  $>$  on a set  $S$  is **well-founded** iff. there is no infinite decreasing chain  $x_1 > x_2 > x_3 > \dots$  for  $x_i \in S$ .

We can construct new well-founded relations from existing ones:

Let  $R_1$  and  $R_2$  be binary relations on a set  $S$ . The composition of  $R_1$  and  $R_2$  is defined as:

$$R_2 \circ R_1 \equiv \{(a, c) \in S \times S \mid \exists b \in S. a R_1 b \wedge b R_2 c\}$$

Note: For binary relation  $R$ , we write  $a R b$  for  $(a, b) \in R$ .

Let  $R \subseteq S \times S$ . Define:

$$\begin{aligned} R^1 &\equiv R \\ R^{n+1} &\equiv R \circ R^n, \text{ for } n \geq 1 \\ R^+ &\equiv \bigcup_{n \geq 1} R^n \end{aligned}$$

So  $a R^+ b$  iff.  $a R^i b$  for some  $i \geq 1$ .

**Lemma:** Let  $R \subseteq S \times S$ . Let  $s_0, s_i \in S$  and  $i \geq 1$ . Then  $s_0 R^i s_i$  iff. there are  $s_1, \dots, s_{i-1} \in S$  such that  $s_0 R s_1 R \dots R s_{i-1} R s_i$ .

**Theorem:** If  $>$  is a well-founded order on set  $S$ , then  $>^+$  is also well-founded on  $S$ .

**Example:** Consider the following function:

```
fac 0 = 1
fac n = n * fac (n - 1)
```

`fac n` has only `fac (n - 1)` as a recursive call, and  $n > n - 1$ . Here,  $>$  is the standard ordering over the natural numbers. Therefore, the function terminates.

### 0.3.2 Proofs

Consider the following program:

```
maxi :: Int -> Int -> Int
maxi n m
  | n >= m    = n
  | otherwise = m
```

Can we prove that `maxi n m >= n`? We to a **reasoning by cases**:

We have  $n \geq m \vee \neg(n \geq m)$ . Now we show that `maxi n m >= n` for both cases:

- Case 1:  $n \geq m$ , then `max n m = n` and  $n \geq n$ .

- Case 2:  $\neg(n \geq m)$ , then  $\max_i n \ m = m$ . But  $m > n$ , so  $\max_i n \ m \geq n$ .

But how do we prove a formula  $P$  (with free variable  $n$ ), for all  $n \in \mathcal{N}$ ? For example, how do we prove the following equality:

$$\forall n \in \mathcal{N}. 0 + 1 + 2 + \dots + n = n \cdot (n + 1) / 2$$

We can do a **proof by induction**:

- Base case: Prove  $P[n \rightarrow 0]$
- Step case: For an arbitrary  $m$  not free in  $P$ , prove  $P[n \rightarrow m + 1]$  under the assumption  $P[n \rightarrow m]$ .

**Example:** We have the following conjecture:  $\forall n \in \mathcal{N}. (\text{sumPowers } n) + 1 = \text{power2 } (n + 1)$  with the following code:

```
power2 :: Int -> Int
power2 0 = 1
power2 r = 2 * power2 (r - 1)

sumPowers :: Int -> Int
sumPowers 0 = 1
sumPowers r = sumPowers (r - 1) + power2 r
```

We want to proof: Let  $P \equiv (\text{sumPowers } n) + 1 = \text{power2 } (n + 1)$ . We show  $\forall m \in \mathcal{N}. P$  by induction on  $n$ .

**Base case:** Show  $P[n \rightarrow 0]$ :

$$\begin{aligned} (\text{sumPowers } 0) + 1 &= 1 + 1 = 2 \\ \text{power2 } (0 + 1) &= 2 \cdot \text{power2 } 0 = 2 \cdot 1 = 2 \end{aligned}$$

**Step case:** Assume  $P[n \rightarrow m]$  for an arbitrary  $m$  (not in  $P$ ), i.e.

$$(\text{sumPowers } m) + 1 = \text{power2 } (m + 1)$$

and prove  $P[n \rightarrow m + 1]$ , i.e.

$$(\text{sumPowers } (m + 1)) + 1 = \text{power2 } ((m + 1) + 1).$$

Proof:

$$\begin{aligned} (\text{sumPowers } (m + 1)) + 1 &= \text{sumPowers } ((m + 1) - 1) + \text{power2 } (m + 1) + 1 \quad (\text{def.}) \\ &= \text{sumPowers } (m) + 1 + \text{power2 } (m + 1) \quad (\text{arithmetic}) \\ &= \text{power2 } (m + 1) + \text{power2 } (m + 1) \quad (\text{ind- hypothesis}) \\ &= 2 \cdot \text{power2 } (m + 1) \quad (\text{arithmetic}) \\ &= \text{power2 } (m + 2) \quad (\text{def.}) \end{aligned}$$

We have proven  $(\text{sumPowers } n) + 1 = \text{power2 } (n + 1)$ .

The general schema for **well-founded induction** is given as:

- *To prove:*  $\forall n \in \mathcal{N}. P$
- *Fix:* An arbitrary  $m$  not free in  $P$
- *Assume:*  $\forall l \in \mathcal{N}. l < m \rightarrow P[n \rightarrow l]$  (*induction hypothesis*)
- *Prove:*  $P[n \rightarrow m]$

# 1 More on Haskell

## 1.1 Lists

### 1.1.1 List Type

We introduce a new type constructor: **List types**, i.e. if  $T$  is a type, then  $[T]$  is a type. The elements of  $[T]$  are:

- *Empty list*:  $[] :: [T]$
- *Non-empty list*:  $(x : xs) :: [T]$  if  $x :: T$  and  $xs :: [T]$

*Syntactic sugar*: We can write  $1 : (2 : (3 : []))$  as  $[1, 2, 3]$ .