Higher-Order Logic

Stefan Hetzl
stefan.hetzl@tuwien.ac.at

Vienna University of Technology

January 25, 2019

Contents

Overview

First-Order Logic
An example for a formula in first-order logic is

$$\forall x\,(\exists y\,R(x,y) \to R(x,f(x))).$$

This formula contains variables x and y which denote individual objects of the domain, it contains a function symbol f which denotes a (unary) function from the domain to the domain and it contains a predicate symbol R which denotes a (binary) relation of objects of the domain. The quantifiers $\forall x, \exists y$ range over *individual objects*. First-order logic is restricted to this form of quantification.

What is Second-Order Logic?
Second-order logic is obtained from allowing – in addition – to quantify over *relations* and *functions* of individual objects. This allows to write down sentences such as

$$\forall X \exists u \forall x\,(\exists y\,X(x,y) \to X(x,u(x))).$$

Why Second-Order Logic?
Second-order logic is more expressive than first-order logic. For example, there is no first-order sentence that defines the finite structures. On the other hand – as we will see – this is easy to do in second-order logic. This is a general pattern that applies to many situations, another example being the connectedness of a graph.

Another important example is the ability of second-order logic to uniquely define the natural numbers ("categoricity of second-order arithmetic"). Generalizations of this result include mechanisms for inductive definitions, in particular for inductive data types like e.g. lists, trees, etc. which are of importance in computer science.

This added expressivity however comes at a price: many results *about* the logic do no longer hold, in particular the completeness theorem, the compactness theorem, and the Löwenheim-Skolem theorem.

In addition to being more expressive than first-order logic, second-order logic is also a quite natural formalism: one can easily observe that in mathematical practice quantification over sets is often used. One reaction to this observation is to work in set theory, a first-order theory where all individual objects are sets. Another reaction is to move from first- to second-order logic, a formal system which distinguishes quantification over individuals from quantification over sets (and predicates and functions) of such individuals.

What is Higher-Order Logic?
Second-order logic is a formal system that distinguishes quantification over different *types* of

objects. We can systematically assign types to the objects about which we speak in the following way: let ι denote the type of individuals (the range of first-order quantifiers) and let o denote the type of formulas (a boolean value which is true or false). We can then write the type of a set of individuals as $\iota \to o$, to each individual we map a boolean value according to whether it is or is not in the set. Similarly we can write the type of a unary function as $\iota \to \iota$, a mapping of individuals to individuals.

Higher-order logic, or simple type theory as it is often called, is then obtained from turning the above into an inductive definition allowing the formation of more complicated types such as for example $(\iota \to \iota) \to (\iota \to o)$ and considering quantifiers for all such types.

A good basis for higher-order logic is the typed lambda calculus which also forms a useful background theory for studying properties of programming languages.

The quantification over such higher types appears frequently in mathematics. For example, the most basic notion in topology, that of an *open set* has the type $(\iota \to o) \to o$ because it is a property of sets of individuals. In functional analysis operators which map functions to functions play an important role. An object which maps a function to a function has the type $(\iota \to \iota) \to (\iota \to \iota)$.

Type theories find practical applications in proof assistants such as Isabelle, Coq, etc. These are software systems that allow a user to formalize a proof in an electronic format.

Helpful additional literature for this couse include [4, 6, 5, 1, 3, 2].

Chapter 1

First-Order Logic

This chapter is a brief repetition of the central definitions and results concerning first-order logic.

Syntax: A *language* contains constant symbols, function symbols and predicate symbols. Each symbol has an arity, the number of arguments it takes (written f/n for the symbol f with arity $n \in \mathbb{N}$). In addition, we assume a countably infinite supply of variable names at our disposal. The terms over a language L (the *L-terms*) are defined inductively from constant symbols, variables and function symbols. *L-formulas* are defined inductively from atoms (including equations $s = t$ for L-terms s, t as well as falsity $\bot$, pronounced "bottom"), the propositional connectives $\wedge$, $\vee$, $\neg$, $\to$ and the quantifiers $\forall x$, $\exists x$.

Example 1.1. $L = \{c/0, f/2, g/1, P/1, R/2\}$ is a language, $f(g(x), c)$ is an L-term, $f(c)$ is not an L-term, $\forall x\,(P(x) \to x = g(x))$ is an L-formula, $R(c, c) \neg P(c)$ is not an L-formula.

To simplify the notation we assume that $\wedge$ and $\vee$ bind stronger than $\to$, so $(A \wedge B) \to (C \vee D)$ can be written as $A \wedge B \to C \vee D$. Furthermore, implication is right-associative, so $A \to B \to C$ means $A \to (B \to C)$. And we define $A \leftrightarrow B$ as abbreviation for $(A \to B) \wedge (B \to A)$.

Semantics: An *L-structure* is a pair $\mathcal{S} = (D, I)$ where D is a set, the *domain* of $\mathcal{S}$ and I maps all constant symbols, function symbols and predicate symbols of L to elements, functions and relations respectively of D and some variables to elements of D. The interpretation I is extended to cover all terms by defining

$$I(f(t_1, \ldots, t_n)) = I(f)(I(t_1), \ldots, I(t_n)).$$

A formula may have free and bound variables, a formula without free variables is called *sentence*. The truth of a formula F in a structure $\mathcal{S} = (D, I)$ is written as $\mathcal{S} \models F$, pronounced as "F is true in $\mathcal{S}$" or "$\mathcal{S}$ satisfies F" or "$\mathcal{S}$ is a model of F", and defined inductively on the structure of F under the assumption that all free variables of F are interpreted by I For example,

$$\mathcal{S} \models A \vee B \text{ if } \mathcal{S} \models A \text{ or } \mathcal{S} \models B,$$
$$(D, I) \models \exists x\, A \text{ if there is an } m \in D \text{ s.t. } (D, I \cup \{x \mapsto m\}) \models A,$$

and so on. A sentence which is true in all structures is called *valid*. A sentence is called *satisfiable* if there is a structure in which it is true. A set of sentences Γ is called *satisfiable* if there is a structure in which all $F \in \Gamma$ are true.

Proofs: There are several different proof systems for classical first-order logic, most notably (the numerous variants of) sequent calculus, natural deduction and Hilbert-type systems. All of these are equivalent w.r.t. provability.

Definition 1.1. A proof in natural deduction for classical logic **NK** is a tree of formulas. A leaf of this tree is called axiom if it is of the form $A \vee \neg A$ for some formula A or $t = t$ for some term t. Otherwise a leaf is an assumption. Each assumption is either *open* or *discharged*. An open assumption is just a formula A, it can be discharged by a rule below it and is then written as $[A]^i$ where i is the number of the rule which has discharged it.

A proof of a formula A possibly containing axioms as well as open or discharged assumptions is written using dots as

$$\begin{array}{c}\vdots\\ A\end{array}$$

Sometimes it is convenient to give a name to a proof; this is written as

$$\begin{array}{c}\vdots\ \pi\\ A\end{array}$$

To emphasize the presence of certain open or discharged assumptions they are mentioned explicitly as in

$$\begin{array}{c}B\\ \vdots\\ A\end{array} \quad \text{or} \quad \begin{array}{c}[B]^i\\ \vdots\\ A\end{array} \quad \text{as well as} \quad \begin{array}{c}\Gamma\\ \vdots\\ A\end{array}$$

for a set Γ of open assumptions.

The set of **NK**-proofs is defined inductively allowing the formation of new proofs by applying any of the following rules:

$$\frac{\begin{array}{cc}\vdots & \vdots\\ A & A \to B\end{array}}{B}\ {\to_{\mathrm{E}}} \qquad \frac{\begin{array}{c}[A]^i\\ \vdots\\ B\end{array}}{A \to B}\ {\to^i_{\mathrm{I}}} \qquad \frac{\begin{array}{cc}\vdots & \vdots\\ A & \neg A\end{array}}{\bot}\ {\neg_{\mathrm{E}}} \qquad \frac{\begin{array}{c}[A]^i\\ \vdots\\ \bot\end{array}}{\neg A}\ {\neg^i_{\mathrm{I}}} \qquad \frac{\begin{array}{c}\vdots\\ \bot\end{array}}{A}\ \mathrm{efq}$$

$$\frac{\begin{array}{cc}\vdots & \vdots\\ A & B\end{array}}{A \wedge B}\ {\wedge_{\mathrm{I}}} \qquad \frac{\begin{array}{c}\vdots\\ A \wedge B\end{array}}{A}\ {\wedge_{\mathrm{E}_1}} \qquad \frac{\begin{array}{c}\vdots\\ A \wedge B\end{array}}{B}\ {\wedge_{\mathrm{E}_2}}$$

$$\frac{\begin{array}{c}\vdots\\ A\end{array}}{A \vee B}\ {\vee_{\mathrm{I}_1}} \qquad \frac{\begin{array}{c}\vdots\\ B\end{array}}{A \vee B}\ {\vee_{\mathrm{I}_2}} \qquad \frac{\begin{array}{ccc}\vdots & [A]^i & [B]^i\\ A \vee B & \vdots & \vdots\\ & C & C\end{array}}{C}\ {\vee^i_{\mathrm{E}}}$$

$$\frac{\begin{array}{c}\vdots\\ \forall x\, A\end{array}}{A[x\backslash t]}\ {\forall_{\mathrm{E}}} \qquad \frac{\begin{array}{c}\vdots\\ A[x\backslash t]\end{array}}{\exists x\, A}\ {\exists_{\mathrm{I}}}$$

provided t does not contain a variable that is bound in A

$$\frac{\begin{array}{c}\vdots\ \pi\\ A[x\backslash\alpha]\end{array}}{\forall x\, A}\ {\forall_{\mathrm{I}}}$$

provided α does not occur in A nor in the open assumptions of π

$$\dfrac{\exists x\, B \qquad \begin{array}{c}[B[x\backslash\alpha]]^i \\ \vdots\,\pi \\ A\end{array}}{A}\ \exists^i_{\mathrm{E}}$$

provided α does not occur in A nor in B nor in the open assumptions of π. In the cases $\forall_{\mathrm{I}}$ and $\exists_{\mathrm{E}}$, α is called *eigenvariable* and the side condition is called *eigenvariable condition.*

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ s = t & A[x\backslash s]\end{array}}{A[x\backslash t]} = \qquad \dfrac{\begin{array}{cc}\vdots & \vdots \\ s = t & A[x\backslash t]\end{array}}{A[x\backslash s]} =$$

provided s and t do not contain variables that are bound in A.

Example 1.2.

$$\dfrac{\dfrac{\dfrac{[\neg A\vee B]^1 \qquad \dfrac{\dfrac{[\neg A]^3 \quad [A]^2}{\bot}\,\neg_{\mathrm{E}}}{B}\,\mathrm{efq} \qquad [B]^3}{B}\,\vee^3_{\mathrm{E}}}{A\to B}\,\to^2_{\mathrm{I}}}{(\neg A\vee B)\to A\to B}\,\to^1_{\mathrm{I}}$$

Make sure you understand how this proof is constructed step by step successively discharging assumptions.

For a set of sentences Γ and a sentence A we write $\Gamma\vdash A$ if A is provable using assumptions from Γ. A set of sentences Γ is called *deductively closed* if $\Gamma\vdash A$ implies $A\in\Gamma$; a deductively closed set of sentences is – more briefly – called *theory.* A set of sentences Γ is called *consistent* if $\Gamma\nvdash\bot$.

The central properties of first-order logic are given by the following theorems:

Theorem 1.1 (Soundness)**.** *Every provable formula is valid.*

While soundness is usually straightforward to prove, the following results require more work.

Theorem 1.2 (Completeness)**.** *Every valid formula is provable.*

Theorem 1.3 (Compactness)**.** *If Γ is a set of sentences s.t. every finite $\Gamma_0\subseteq\Gamma$ is satisfiable, then Γ is satisfiable.*

Theorem 1.4 (Löwenheim-Skolem)**.** *If Γ has a model, then Γ has a countable model.*

The strongest version of these results from which Completeness, Compactness and the Löwenheim-Skolem theorem can be derived as simple corollaries is the following

Lemma 1.1. *Every consistent set of sentences has a countable model.*

The following exercises are helpful for refreshing your knowledge about first-order logic.

Exercise 1.1. *Let A be a formula. Show that $\exists x\, A$ is unsatisfiable iff $\forall x\,\neg A$ is valid.*

Exercise 1.2. *Show that $\forall x\,(P(x)\to Q(x))\to(\forall x\,P(x)\to\forall x\,Q(x))$ is valid but $(\forall x\,P(x)\to\forall x\,Q(x))\to\forall x\,(P(x)\to Q(x))$ is not.*

Exercise 1.3. *Find a formula in the empty language which is true in a structure $\mathcal{S}$ iff the domain of $\mathcal{S}$ has exactly 3 elements.*

Exercise 1.4. *Find a formula which has an infinite but no finite models.*

Exercise 1.5. *Of the following three statements one is wrong, one is trivial and one is a formulation of the compactness theorem. Which is which?*

1. *If there is a finite $\Gamma_0 \subseteq \Gamma$ which is satisfiable, then Γ is satisfiable.*
2. *If Γ is unsatisfiable then there is a finite $\Gamma_0 \subseteq \Gamma$ which is unsatisfiable.*
3. *If Γ is satisfiable then there is a finite $\Gamma_0 \subseteq \Gamma$ which is satisfiable.*

Justify your answers!

Exercise 1.6. *Prove the following three results*

Completeness Theorem. Every valid formula is provable.

Compactness Theorem. If Γ is a set of sentences s.t. every finite $\Gamma_0 \subseteq \Gamma$ is satisfiable, then Γ is satisfiable.

Löwenheim-Skolem Theorem. If Γ has a model, then Γ has a countable model.

from the following

Main Lemma. Every consistent set of sentences has a countable model.

Exercise 1.7. *Two structures (D_1, I_1) and (D_2, I_2) in the same language L are called* isomorphic*, often written as $(D_1, I_1) \simeq (D_2, I_2)$, if there is a bijection $\psi : D_1 \to D_2$ s.t.*

1. *$\psi(I_1(c)) = I_2(c)$,*
2. *$\psi(I_1(f)(m_1, \ldots, m_n)) = I_2(f)(\psi(m_1), \ldots, \psi(m_n))$ for all $m_1, \ldots, m_n \in D_1$, and*
3. *$(m_1, \ldots, m_n) \in I_1(P)$ iff $(\psi(m_1), \ldots, \psi(m_n)) \in I_2(P)$ for all $m_1, \ldots, m_n \in D_1$.*

The theory of a structure $\mathcal{S}$ *is defined as* $\mathrm{Th}(\mathcal{S}) = \{A \text{ sentence} \mid \mathcal{S} \models A\}$. *$\mathcal{S}_1$ and $\mathcal{S}_2$ are called* elementarily equivalent *if* $\mathrm{Th}(\mathcal{S}_1) = \mathrm{Th}(\mathcal{S}_2)$.

(a) Show that two isomorphic structures are elementarily equivalent.
Hint: first show $\psi(I_1(t)) = I_2(t)$ by induction on the term structure of t and then (a) by induction on formula structure.

The language of arithmetic is $L_{\mathbb{N}} = \{0/0, s/1, +/2, \cdot/2, </2\}$. The standard model *of arithmetic is the $L_{\mathbb{N}}$-structure $\mathcal{N} = (\mathbb{N}, I)$ where I is the obvious standard-interpretation of the symbols in $L_{\mathbb{N}}$.*

(b) Show that there is a structure which is elementary equivalent but not isomorphic to $\mathcal{N}$.
Hint: Add a new constant symbol c to $L_{\mathbb{N}}$, successively force c to be larger than each natural number and apply the compactness theorem.

A structure as in (b) is called non-standard *model of arithmetic.*

Exercise 1.8. *A theory T is called* countably categorical *if, whenever $\mathcal{S}_1$ and $\mathcal{S}_2$ are countably infinite models of T, then $\mathcal{S}_1$ and $\mathcal{S}_2$ are isomorphic.*

A theory T is called complete *if, for every sentence A either $T \vdash A$ or $T \vdash \neg A$.*

Show that a countably categorical theory without finite models is complete.

Chapter 2

Second-Order Logic

2.1 Syntax and Semantics

The syntax of second-order logic is obtained from that of first-order logic by adding predicate variables and function variables as well as quantification over these newly introduced variables. As a notational convention we use letters $X, Y, \ldots$ for predicate variables and $u, v, \ldots$ for function variables. Often the arity of these variables will be clear from the context or irrelevant. If, not it is indicated as $u/r, X/r$. As in first-order logic we use $x, y, \ldots$ for individual variables.

When moving from first- to second-order logic the definition of a language and of a structure is not changed. Only the notion of terms, formulas and accordingly the truth definition is extended.

Definition 2.1. For a language L, the set of L-terms is as follows:

1. Individual variables are L-terms
2. Constant symbols in L are L-terms
3. If $t_1, \ldots, t_n$ are L-terms and f is an n-ary function symbol of L, then $f(t_1, \ldots, t_n)$ is an L-term.
4. If $t_1, \ldots, t_n$ are L-terms and u is a function variable of arity n, then $u(t_1, \ldots, t_n)$ is an L-terms.

Definition 2.2. For a language L, the set of L-formulas is defined as follows:

1. If $t_1, \ldots, t_n$ are L-terms and P is an n-ary predicate symbol of L, then $P(t_1, \ldots, t_n)$ is an L-formula.
2. If $t_1, \ldots, t_n$ are L-terms and X is an n-ary relation variable, then $X(t_1, \ldots, t_n)$ is an L-formula.
3. If t_1 and t_2 are L-terms, then $t_1 = t_2$ is an L-formula.
4. $\bot$ is an L-formula.
5. If A is an L-formula, then $\neg A$, $\forall x\, A$, $\exists x\, A$, $\forall X\, A$, $\exists X\, A$, $\forall u\, A$, and $\exists u\, A$ are L-formulas.
6. If A and B are L-formulas, then $(A \wedge B)$, $(A \vee B)$, and $(A \to B)$ are L-formulas.

The free variables and bound variables of a formula are defined to include predicate and function variables in the obvious way. As in first-order logic, a sentence is a formula without free variables.

Definition 2.3. We say that a structure $\mathcal{S} = (D, I)$ *interprets* a formula A if every constant symbol, function symbol, predicate symbol and free variable of A is in the domain of I.

Definition 2.4. Let A be a formula and $\mathcal{S} = (D, I)$ be a structure that interprets A. Then the truth of A in $\mathcal{S}$ is defined by induction on A:

1. $\mathcal{S} \models P(t_1, \ldots, t_n)$ if $(I(t_1), \ldots, I(t_n)) \in I(P)$.
2. $\mathcal{S} \models X(t_1, \ldots, t_n)$ if $(I(t_1), \ldots, I(t_n)) \in I(X)$.
3. $\mathcal{S} \models t_1 = t_2$ if $I(t_1) = I(t_2)$.
4. $\mathcal{S} \not\models \bot$
5. $\mathcal{S} \models A \wedge B$ if $\mathcal{S} \models A$ and $\mathcal{S} \models B$.
6. $\mathcal{S} \models \neg A$ if $\mathcal{S} \not\models A$.
7. $\mathcal{S} \models \forall x\, A$ if for all $m \in D$: $(D, I \cup \{x \mapsto m\}) \models A$.
8. $\mathcal{S} \models \forall X\, A$ if for all $R \subseteq D^{\mathrm{arity}(X)}$: $(D, I \cup \{X \mapsto R\}) \models A$.
9. $\mathcal{S} \models \forall u\, A$ if for all $\varphi : D^{\mathrm{arity}(u)} \to D$: $(D, I \cup \{u \mapsto \varphi\}) \models A$.

and analogously for the disjunction $\vee$ and the existential quantifiers $\exists x$, $\exists X$ and $\exists u$.

As the above definitions show, the move from first- to second-order logic does not require substantially new ideas on the level of the basic syntax and semantics. We only generalize quantifiers in a straightforward way to speak also about functions and relations. However, this generalization considerably increases the expressive power of the logic (as we will see in the next sections).

Example 2.1. An example for a second-order formula with one free function variable $u/1$ is

$$\mathrm{Inv}(u) \;\equiv\; \exists v\, \forall x\, \forall y\, (u(x) = y \leftrightarrow v(y) = x)$$

which asserts the existence of an inverse function for u. Let us now consider the structure $(\mathbb{R}, I)$ where $I(u)(x) = 2x$. We want to show that $(\mathbb{R}, I) \models \mathrm{Inv}(u)$.

$$\begin{aligned}
&(\mathbb{R}, I) \models \mathrm{Inv}(u)\\
&\text{iff } (\mathbb{R}, I) \models \exists v\, \forall x\, \forall y\, (u(x) = y \leftrightarrow v(y) = x)\\
&\text{iff there is } \varphi : \mathbb{R} \to \mathbb{R} \text{ s.t. } (\mathbb{R}, I \cup \{v \mapsto \varphi\}) \models \forall x\, \forall y\, (u(x) = y \leftrightarrow v(y) = x)\\
&\text{iff there is } \varphi : \mathbb{R} \to \mathbb{R} \text{ s.t. for all } \lambda, \mu \in \mathbb{R}:\\
&\quad (\mathbb{R}, \underbrace{I \cup \{v \mapsto \varphi, x \mapsto \lambda, y \mapsto \mu\}}_{I^*}) \models u(x) = y \leftrightarrow v(y) = x\\
&\text{iff there is } \varphi : \mathbb{R} \to \mathbb{R} \text{ s.t. for all } \lambda, \mu \in \mathbb{R}:\\
&\quad I^*(u)(I^*(x)) = I^*(y) \text{ iff } I^*(v)(I^*(y)) = I^*(x) \text{ in } \mathbb{R}, \text{ i.e.}\\
&\quad 2\lambda = \mu \text{ iff } \varphi(\mu) = \lambda \text{ in } \mathbb{R}
\end{aligned}$$

and there is such φ, namely $x \mapsto \frac{x}{2}$ hence $(\mathbb{R}, I) \models \mathrm{Inv}(u)$.

2.2 Definability

We first develop some syntactic sugar to speak about sets in second-order logic. First of all, note that a set is just a unary predicate. In this special case of unary predicates we use the notation $x \in Y$ for $Y(x)$. But be careful, the symbol $\in$ in this expression is not a predicate itself, it is only part of a notational convention. Another useful notational convention are bounded quantifiers:

$$\begin{aligned} \forall x \in X\, A &\equiv \forall x\, (x \in X \to A) \\ \exists x \in X\, A &\equiv \exists x\, (x \in X \wedge A) \end{aligned}$$

Let $X/2, Y/1$ and define

$$\begin{aligned} \mathrm{Trans}(X) &\equiv \forall x\, \forall y\, \forall z\, (X(x,y) \to X(y,z) \to X(x,z)), \\ \mathrm{Suc}(X,Y) &\equiv \forall x \in Y\, \exists y \in Y\, X(x,y), \\ \mathrm{Fix}(X,Y) &\equiv \exists x \in Y\, X(x,x), \text{ and} \\ \mathrm{Fin}(Y) &\equiv \forall X\, (\mathrm{Trans}(X) \to \mathrm{Suc}(X,Y) \to \mathrm{Fix}(X,Y)) \vee \forall x\, x \notin Y. \end{aligned}$$

Proposition 2.1. *Let (D, I) be a structure that interprets $Y/1$, then*

$$I(Y) \text{ is finite iff } (D,I) \vDash \mathrm{Fin}(Y)$$

Proof. For the left-to-right direction assume that $I(Y)$ is finite. If $I(Y) = \varnothing$, then $(D,I) \vDash \forall x\, x \notin Y$ and we are done, so assume $I(Y) \neq \varnothing$. Let R be a binary relation which is transitive and has successors on $I(Y)$, then we will show that R has a fixed point. To that aim, pick $m_1 \in I(Y) \neq \varnothing$. As R has successors on $I(Y)$ there is $m_2 \in I(Y)$ s.t. $(m_1, m_2) \in R$, etc. Continuing this process we obtain a sequence $m_1, m_2, m_3, \ldots$ in $I(Y)$ with $(m_i, m_{i+1}) \in R$. But by transitivity of R and finiteness of $I(Y)$ there is i s.t. $(m_i, m_i) \in R$ and we are done.

For the right-to-left direction suppose that $I(Y)$ is infinite. Then it is non-empty hence $(D,I) \nvDash \forall x\, x \notin Y$. Furthermore, we can choose a set $M = \{m_1, m_2, m_3, \ldots\} \subseteq I(Y)$ s.t. the m_i are pairwise different. We define a binary relation R on D as the transitive closure of

$$R = \{(k, m_1) \mid k \notin M\} \cup \{(m_i, m_{i+1}) \mid i \geqslant 1\}$$

Then R is transitive, has successors on $I(Y)$ but no fixed points and hence $(D,I) \nvDash \mathrm{Fin}(Y)$. □

A very useful notation in second- and even more so in higher-order logic is provided by the *lambda calculus*. To explain it, consider the syntactic expression $x^2 + 5$. This can be read as a term in a first-order language containing the binary function symbols $+$ and "to the power of" as well as the constant symbols 2 and 5 and the variable x. Often when writing such expressions we want to consider the function that maps the value $x^2 + 5$ to x. In ordinary mathematical notation, this function is often written as $x \mapsto x^2 + 5$. In the lambda calculus this function is denoted as $\lambda x.x^2 + 5$. This notation has the advantage that nested expressions are easier to read which is helpful in higher-order logic. The step from the term $x^2 + 5$ to the function which associates the value $x^2 + 5$ to x is called *abstraction*. This notation is also used for expressions which contain more variables, so for example $\lambda x \lambda y.x^3 + x \cdot y - 5 \cdot z$ is a function with the two arguments x and y and the parameter z.

The *application* of a function f to an argument x is denoted as fx in the lambda calculus. So for example $(\lambda x.x^2 + 5)3$ denotes the application of the above function to 3. An abstraction

followed by an application can then be *reduced* by replacing the variable of the abstraction by the argument:

$$(\lambda x.x^2 + 5)3 \;\rightarrow\; 3^2 + 5.$$

We will use the λ-notation and this kind of reduction (called β-reduction) at various occasions in an informal ways until we study the lambda calculus more thoroughly in the context of higher-order logic.

Using the notation of the lambda calculus, we can for example define new predicates from existing ones, e.g. given $P/1$ and $Q/1$ the predicate $R/1$ which is true if both P and Q are true can be written as:

$$R \;\equiv\; \lambda x.P(x) \wedge Q(x)$$

Similarly, we can define the empty set explicitely as

$$\varnothing \;\equiv\; \lambda x.\bot$$

and the universe as

$$\mathcal{U} \;\equiv\; \lambda x.\top.$$

The sentence $\mathrm{Fin}(\mathcal{U})$ is hence true in a structure $\mathcal{S}$ iff $\mathcal{S}$ is finite.

Definition 2.5. A class of structures $\mathcal{C}$ is called *first-order definable* (second-order definable) if there is a set of first-order (second-order) sentences Γ s.t. the models of Γ are exactly the elements of $\mathcal{C}$.

If a class $\mathcal{C}$ is definable by a *finite* set of sentences Γ, then it is definable by the sentence $\bigwedge_{A\in\Gamma} A$ because $\mathcal{M} \models \Gamma$ iff $\mathcal{M} \models \bigwedge_{A\in\Gamma} A$. In this case we say that $\mathcal{C}$ is *definable by a sentence* (of first- or second- order logic respectively).

Proposition 2.2.

1. *For each $n \in \mathbb{N}$ the class of structures of cardinality n is definable by a first-order sentence.*
2. *The class of finite structures ...*
   (a) *... is not first-order definable.*
   (b) *... is definable by a second-order sentence.*
3. *The class of infinite structures ...*
   (a) *... is not definable by a first-order sentence.*
   (b) *... is first-order definable.*
   (c) *... is definable by a second-order sentence.*

Proof. Let $L_n \;\equiv\; \exists x_1 \ldots \exists x_n \bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n} x_i \neq x_j$, then $\mathcal{S} \models L_n$ iff $|\mathcal{S}| \geqslant n$. Furthermore, define $E_n \;\equiv\; L_n \wedge \neg L_{n+1}$ then $\mathcal{S} \models E_n$ iff $|\mathcal{S}| = n$.

1. The class of structures of cardinality n is hence defined by E_n.

2a. Suppose there is a set of sentences Γ s.t. $\mathcal{S} \models \Gamma$ iff $\mathcal{S}$ is finite. Let $\Gamma' = \Gamma \cup \{L_i \mid i \geqslant 1\}$ and observe that Γ' is unsatisfiable: a model of Γ' would need to be finite to satisfy Γ but larger than any $i \in \mathbb{N}$.

Let Γ_0 be an arbitrary finite subset of Γ', then there is an $k \in \mathbb{N}$ s.t. $L_i \in \Gamma_0$ implies that $i < k$. Now let $\mathcal{S}$ be any structure of size k, then $\mathcal{S} \models \Gamma$ and $\mathcal{S} \models L_i$ for all $L_i \in \Gamma_0$ hence $\mathcal{S} \models \Gamma_0$. So every finite $\Gamma_0 \subseteq \Gamma$ has a model.

Therefore, by the compactness theorem, Γ' has a model. Contradiction.

2b. The second-order sentence $\mathrm{Fin}(\mathcal{U})$ defines the class of finite structures.

3a. Suppose there is an A s.t. $\mathcal{S} \models A$ iff $\mathcal{S}$ is infinite, then $\mathcal{S} \models \neg A$ iff $\mathcal{S}$ is finite which contradicts 2a.

3b. The class of infinite structures is defined by $\{L_i \mid i \geqslant 1\}$.

3c. The second-order sentence $\neg\mathrm{Fin}(\mathcal{U})$ defines the class of infinite structures.

□

This strategy for showing the undefinability of a property in first-order logic by using the compactness theorem as in point 2a of the above theorem is important and will reappear at a number of occasions during this course. Make sure you understand how it works.

Also note how subtle a line we were able to draw here: a particular class, that of infinite structures, can be defined by an infinite set of formulas in first-order logic not however by a finite set. A situation such as in Proposition 2.2 is quite typical for the definability of many properties that are related to infinity.

Theorem 2.1. *The compactness theorem fails in Second-Order Logic.*

Proof. Using the notation of the proof of Proposition 2.2, let

$$\Delta = \{\mathrm{Fin}(\mathcal{U})\} \cup \{L_i \mid i \geqslant 1\}$$

and observe that every finite $\Delta_0 \subseteq \Delta$ is satisfiable but Δ is not. □

Compare this proof to that of Proposition 2.2/2a by letting $\Gamma = \{\mathrm{Fin}(\mathcal{U})\}$. Both proofs rely on the same argument which shows that one cannot have the compactness theorem in a setting where the class of finite models is definable. We are in the situation of a drawback: we can only have one but not both. In the case of first-order logic we know that the compactness theorem holds hence the class of finite structures cannot be defined. In the case of second-order logic we know that the class of finite structures can be defined and hence the compactness theorem cannot hold.

We will now move on to see that also the Löwenheim-Skolem theorem fails in second-order logic. We abbreviate

$$X \subseteq Y \;\equiv\; \forall x\,(x \in X \to x \in Y)$$

and use bounded quantification also for this relation

$$\forall X \subseteq Y\, A \;\equiv\; \forall X\,(X \subseteq Y \to A).$$

Now we want to compare sets w.r.t. their size. For finite sets this is easy: just count the number of elements, the set that has more elements is larger. For infinite sets the issue is a bit more subtle. We can say that an infinite set S is larger than an infinite set R if there is a surjection from S to R. Consequently, for $X/1, Y/1$ define

$$X \leqslant Y \;\equiv\; \exists u\, \forall x \in X\, \exists y \in Y\, u(y) = x$$

which expresses exactly that. Furthermore, define

$$X \sim Y \;\equiv\; X \leqslant Y \wedge Y \leqslant X$$

which is equivalent to the existence of a bijection between the two sets. We can now finally define

$$\text{Countable}(Y) \;\equiv\; \forall X \subseteq Y\,(\text{Fin}(X) \vee X \sim Y)$$

which expresses that Y countable: it is obviously true for finite Y and if Y is countably infinite and we pick some subset X of Y one of two things happen: either Y is finite or it is of the same cardinality as X. In other words: $(D, I) \models \text{Countable}(Y)$ iff $I(Y)$ is countable.

Theorem 2.2. *The class of countable structures is definable by a second-order formula.*

Proof. Use Countable($\mathcal{U}$). □

Corollary 2.1. *The class of uncountable structures is definable by a second-order formula.*

Proof. Use ¬Countable($\mathcal{U}$). □

Corollary 2.2. *The Löwenheim-Skolem theorem fails in Second-Order Logic.*

Proof. The formula ¬Countable($\mathcal{U}$) is satisfiable but has no countable model. □

Exercise 2.1. *Show that a set S is finite iff every injective $f : S \to S$ is surjective. Use this property to give a second-order sentence that defines the class of finite structures.*

Exercise 2.2. *In this exercise we work in a language which contains a single binary predicate symbol E and two constant symbols c and d. The structures of this language are (finite or infinite) graphs with two designated vertices, a source c and a sink d. A* path *is a finite list of vertices connected by edges. The* length *of a path is the number of edges it contains.*
Show that:

1. *For every $k \in \mathbb{N}$ the graphs containing a path of length k from c to d are definable by a first-order sentence.*
2. *The graphs which do contain a path from c to d ...*
   (a) *... are not first-order definable.*
   (b) *... are definable by a second-order sentence.*
3. *The graphs which do not contain a path from c to d ...*
   (a) *... are not definable by a first-order sentence.*
   (b) *... are first-order definable.*
   (c) *... are definable by a second-order sentence.*

Exercise 2.3. *We work in the (first-order) language $L = \{0/0, 1/0, +/2, -/1, \cdot/2, {}^{-1}/1\}$ of rings.*

1. *Show that the class of rings is definable by a first-order sentence.*
   Hint: consult your algebra textbook or wikipedia.

The characteristic of a ring is the smallest number $n > 0$ s.t.

$$\underbrace{1 + 1 + \cdots + 1}_{n \text{ times}} = 0.$$

if such a number n exists. If no such n exists, the characteristic of the ring is 0*. A characteristic $n \neq 0$ is also called* finite characteristic*. For example, the ring of integers modulo m has characteristic m, the ring $\mathbb{Z}$ has characteristic* 0*.*

Show that:

2. *For every $k \geqslant 2$, the class of rings of characteristic k is definable by a first-order sentence.*
3. *The class of rings of finite characteristic ...*
   (a) *... is not first-order definable.*
   (b) *... is definable by a second-order sentence.*
4. *The class of rings of characteristic* 0 *...*
   (a) *... is not definable by a first-order sentence.*
   (b) *... is first-order definable.*
   (c) *... is definable by a second-order sentence.*

2.3 Proofs

A calculus for first-order logic can be extended to second-order logic in a natural way. For the case of natural deduction, in order to obtain **NK2** it suffices to add the following rules to **NK**:

$$\frac{\begin{array}{c}\vdots\\ \forall X\, A\end{array}}{A[X\backslash\lambda\bar{x}.B]}\ \forall_{\mathrm{E}} \qquad \frac{\begin{array}{c}\vdots\\ \forall u\, A\end{array}}{A[u\backslash\lambda\bar{x}.t]}\ \forall_{\mathrm{E}}$$

if the number of variables in $\bar{x}$ is the arity of X (or of u respectively) and B (or t respectively) does not contain a variable that is bound in A.

$$\frac{\begin{array}{c}\vdots\ \pi\\ A[X\backslash X_0]\end{array}}{\forall X\, A}\ \forall_{\mathrm{I}} \qquad \frac{\begin{array}{c}\vdots\ \pi\\ A[u\backslash u_0]\end{array}}{\forall u\, A}\ \forall_{\mathrm{I}}$$

if X_0 (or u_0 respectively) has the same arity as X (or u respectively) and X_0 (or u_0 respectively) does not occur in A nor in the open assumptions of π. For the existential quantifier:

$$\frac{\begin{array}{c}\vdots\\ A[X\backslash\lambda\bar{x}.B]\end{array}}{\exists X\, A}\ \exists_{\mathrm{I}} \qquad \frac{\begin{array}{c}\vdots\\ A[u\backslash\lambda\bar{x}.t]\end{array}}{\exists u\, A}\ \exists_{\mathrm{I}}$$

if the number of variables in $\bar{x}$ is the arity of X (or of u respectively) and B (or t respectively) does not contain a variable that is bound in A.

$$\frac{\begin{array}{cc}\vdots & [B[X\backslash X_0]]^i\\ \exists X\, B & \vdots\ \pi\end{array}}{A}\ \exists^i_{\mathrm{E}} \qquad \frac{\begin{array}{cc}\vdots & [B[u\backslash u_0]]^i\\ \exists u\, B & \vdots\ \pi\end{array}}{A}\ \exists^i_{\mathrm{E}}$$

if X_0 (or u_0 respectively) has the same arity as X (or u respectively) and X_0 (or u_0 respectively) does not occur in A nor in the open assumptions of the right-hand proof of A.

Observe that the extension to the second-order system is quite natural; it does not require any new ideas, just the adaption of the first-order system to the case of the second-order quantifiers, all side conditions are analogous. The extension of the semantics has been an equally straightforward adaption of the first-order semantics. The question is hence natural whether the relationship between syntax and semantics generalizes from first- to second-order logic in the same straightforward way. While the soundness of the above inference rules is an easy exercise we will see in the next section that the completeness theorem will fail in second-order logic.

Example 2.2. The sentence

$$\forall u\, \exists X\, \forall y\, (X(y) \leftrightarrow \exists x\, u(x) = y)$$

states that every function has a range and is provable in **NK2**. A proof in **NK2** is:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ \forall y\, (\exists x\, u_0(x) = y \leftrightarrow \exists x\, u_0(x) = y)\end{array}}{\forall y\, ((\lambda z.\exists x\, u_0(x) = z)y \leftrightarrow \exists x\, u_0(x) = y)}\ \exists_{\mathrm{I}}}{\exists X\, \forall y\, (X(y) \leftrightarrow \exists x\, u_0(x) = y)}}{\forall u\, \exists X\, \forall y\, (X(y) \leftrightarrow \exists x\, u(x) = y)}\ \forall_{\mathrm{I}}$$

where the vertical dots represent a straightforward proof in first-order logic. Usually the line containing the lambda-expression is not written down, it is done here for expository purposes only.

Exercise 2.4. *Show that* $\forall X\, \forall Y\, (X \subseteq Y \to X \leqslant Y)$ *is valid by giving a proof in* **NK2**. *Show that* $\forall X\, \forall Y\, (X \leqslant Y \to X \subseteq Y)$ *is not valid by specifying a counterexample.*

Exercise 2.5. *There is a philosophical principle attributed to Leibniz which states that equality of two objects means that they have all properties in common ("Leibniz equality"). Mathematically speaking, a property is just a set, so this can be formulated in second-order logic:*

To do so, we define a translation L *of second-order formulas containing equality into such that do not by defining:*

$$\begin{aligned}
(s = t)^{\mathrm{L}} &\equiv \forall X\, (X(s) \leftrightarrow X(t))\\
A^{\mathrm{L}} &\equiv A \text{ for all atoms which are not equations}\\
(\neg A)^{\mathrm{L}} &\equiv \neg A^{\mathrm{L}}\\
(A \wedge B)^{\mathrm{L}} &\equiv A^{\mathrm{L}} \wedge B^{\mathrm{L}}\\
(\forall X\, A)^{\mathrm{L}} &\equiv \forall X\, A^{\mathrm{L}},\ (\forall u\, A)^{\mathrm{L}} \equiv \forall u\, A^{\mathrm{L}},\ (\forall x\, A)^{\mathrm{L}} \equiv \forall x\, A^{\mathrm{L}}
\end{aligned}$$

In this exercise we assume that formulas only contain the connectives $\neg, \wedge, \forall$ *in order to avoid repetition of analogous cases.*

1. *Show that* $A \leftrightarrow A^{\mathrm{L}}$ *is valid for any formula* A.
2. *Show that* $(t = t)^{\mathrm{L}}$ *is provable in* **NK2** *for any term* t.
3. *Show that* $A^{\mathrm{L}}[x\backslash t] \equiv A[x\backslash t]^{\mathrm{L}}$ *for any formula* A *and any term* t.

4. *Show that:* $A[x\backslash t]^{\mathrm{L}}$ *is provable in* **NK2** *from the open assumptions* $(s = t)^{\mathrm{L}}$ *and* $A[x\backslash s]^{\mathrm{L}}$.
   Show that: $A[x\backslash s]^{\mathrm{L}}$ *is provable in* **NK2** *from the open assumptions* $(s = t)^{\mathrm{L}}$ *and* $A[x\backslash t]^{\mathrm{L}}$.

The points 2.-4. above show that a proof in **NK2** *of a sentence* A *can be transformed into a proof of* A^{L} *in which equality does no longer appear by replacing reflexivity axioms by the proof constructed in 2. and instances of the equality rules by the proofs constructed in 4.*

2.4 Second-Order Arithmetic

First some reminders: the language of arithmetic is $L_{\mathbb{N}} = \{0/0, s/1, +/2, \cdot/2, </2\}$. The *standard model* is $\mathcal{N} = (\mathbb{N}, I_{\mathrm{s}})$ where I_{s} interprets the constant symbol 0 by the natural number 0, the unary function symbol s by the successor function and so on. We have already seen in the exercises that there are non-standard models of arithmetic, i.e. structures that satisfy the same first-order sentences as $\mathcal{N}$ but which are *not* isomorphic to $\mathcal{N}$. These models are rather strange creatures which possess, in addition to the standard numbers also non-standard numbers which start after (in the sense of the interpretation of $<$) infinity. We will now see that second-order logic allows to rule out these non-standard models and thus to uniquely define the natural numbers.

Definition 2.6. The theory Q of *minimal arithmetic* is the deductive closure of the following set of axioms:

$$\begin{array}{ll} \forall x\, 0 \neq s(x) & \forall x\, x + 0 = x \\ \forall x \forall y\, (s(x) = s(y) \to x = y) & \forall x \forall y\, x + s(y) = s(x + y) \\ \forall x\, x \cdot 0 = 0 & \forall x\, x \not< 0 \\ \forall x \forall y\, x \cdot s(y) = (x \cdot y) + x & \forall x \forall y\, (x < s(y) \leftrightarrow (x < y \vee x = y)) \\ & \forall x \forall y\, (x < y \vee x = y \vee y < x) \end{array}$$

Theorem 2.3 (Σ_1-completeness of Q)**.** *If* $\mathcal{N} \models \exists x\, A$ *where* A *is quantifier-free, then* $Q \vdash \exists x\, A$

Before we sketch the proof another reminder: a *numeral* is a term of the form $s^n(0)$. It is the canonical way to represent the number $n \in \mathbb{N}$ in the language of arithmetic.

Proof Sketch. If $\mathcal{N} \models \exists x\, A$ then there is an $n \in \mathbb{N}$ s.t. $\mathcal{N} \models A[x\backslash \bar{n}]$. By a straightforward (but long) argument one can show that every such $A[x\backslash \bar{n}]$ has a Q-proof[1]. Hence $Q \vdash \exists x\, A$ by a single application of $\exists_{\mathrm{I}}$. □

Definition 2.7 (Second-Order Arithmetic)**.** The induction axiom is

$$\mathrm{Ind} \;\equiv\; \forall X\, (\; X(0) \to \forall x\, (X(x) \to X(s(x))) \to \forall x\, X(x)\;).$$

Denote with SOA the conjunction of Ind and all axioms of Q.

We will now see that there are no non-standard models of second-order arithmetic. The sentence SOA uniquely defines the natural numbers in the sense that – up to isomorphism – the only model of SOA is the standard model. More precisely what we will show is:

Theorem 2.4. $\mathcal{S} \models \mathrm{SOA}$ *iff* $\mathcal{S} \simeq \mathcal{N}$.

[1] Try it for a simple $A[x\backslash \bar{n}]$, e.g. for $s(s(0)) + s(0) = s(s(s(0)))$ as an exercise.

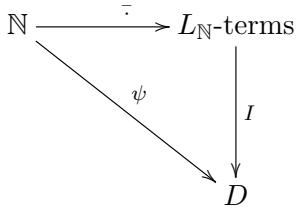

Figure 2.1: The isomorphism in the proof of Theorem 2.4

This theorem is sometimes formulated as the statement "Second-order arithmetic is categorical". This refers to the terminology of calling a theory *categorical* if all its models are isomorphic. As a reminder, the definition of isomorphism for structures is repeated below.

Definition 2.8. Two structures (D_1, I_1) and (D_2, I_2) in the same language L are called *isomorphic*, written as $(D_1, I_1) \simeq (D_2, I_2)$, if there is a bijection $\psi : D_1 \to D_2$ s.t.

1. $\psi(I_1(c)) = I_2(c)$ for all constant symbols c in L,
2. $\psi(I_1(f)(m_1, \ldots, m_n)) = I_2(f)(\psi(m_1), \ldots, \psi(m_n))$ for all function symbols f in L and all $m_1, \ldots, m_n \in D_1$, and
3. $(m_1, \ldots, m_n) \in I_1(P)$ iff $(\psi(m_1), \ldots, \psi(m_n)) \in I_2(P)$ for all predicate symbols P in L and all $m_1, \ldots, m_n \in D_1$.

As our aim is to define the natural numbers one could wonder whether the definition can be made even more precise: Is there a sentence F s.t. $\mathcal{M} \models F$ iff $\mathcal{M} = \mathcal{N}$? In the exercises we have shown that two isomorphic structures satisfy the same (first-order) sentences. This result carries over to second-order logic.

Proposition 2.3. *If $\mathcal{M}_1 \simeq \mathcal{M}_2$ and A is a second-order sentence, then $\mathcal{M}_1 \models A$ iff $\mathcal{M}_2 \models A$.*

Proof. Analogous to first-order logic[2]. □

So when we try to define a particular structure by a set of sentences (be that in first- or in second-order logic) we can never expect more than a definition up to isomorphism.

Proof of Theorem 2.4. First note that $\mathcal{N} \models \mathrm{SOA}$ as both induction as well as the axioms of Q are true in $\mathcal{N}$. As $\mathcal{M} \simeq \mathcal{N}$, the right-to-left direction is done by Proposition 2.3.

For the left-to-right direction, let $(D, I) \models \mathrm{SOA}$ and define a function

$$\psi : \mathbb{N} \to D, n \mapsto I(\bar{n})$$

which we will show to be the required isomorphism. For understanding this proof, it is important to be aware of the types of the objects involved as we are dealing with three different domains here: the natural numbers, $L_{\mathbb{N}}$-terms and M, see Figure 2.1.

First, we have

$$(D, I) \models \forall X \, (\, X(0) \to \forall x \, (X(x) \to X(s(x))) \to \forall x \, X(x) \,).$$

[2]This is a good opportunity to review Exercise 1.7/a.

Define $I' = I \cup \{X \mapsto \mathrm{img}(\psi)\}$, then we have

$$(D, I') \vDash X(0) \to \forall x\,(X(x) \to X(s(x))) \to \forall x\, X(x).$$

For all numerals $\bar{n}$ we have $I(\bar{n}) = I'(\bar{n})$. We have $I(0) \in \mathrm{img}(\psi)$, so

$$(D, I') \vDash X(0).$$

Now we want to show

$$(D, I') \vDash \forall x\,(X(x) \to X(s(x))),$$

i.e. if $m \in \mathrm{img}(\psi)$ then $I(s)(m) \in \mathrm{img}(\psi)$. Now if $m \in \mathrm{img}(\psi)$ then there is $m_0 \in \mathbb{N}$ s.t. $\psi(m_0) = m$. But then $I(s)(m) = I(s)(I(\overline{m_0})) = I(\overline{m_0 + 1}) = \psi(m_0 + 1) \in \mathrm{img}(\psi)$. Combining the above we obtain

$$(D, I') \vDash \forall x\, X(x)$$

i.e. $D = \mathrm{img}(\psi)$, in other words: ψ is surjective.

Moreover, let $m, n \in \mathbb{N}$ with $m \neq n$. Then by Σ_1-completeness of Q, we know that $Q \vdash \bar{m} \neq \bar{n}$ and as (D, I) is a model of Q, it also satisfies $\bar{m} \neq \bar{n}$, i.e. $I(\bar{m}) \neq I(\bar{n})$ and therefore ψ is injective.

In remains to show that ψ is a homomorphism

- $\psi(0) = I(\bar{0})$.
- $\psi(n + 1) = I(\overline{n + 1}) = I(s(\bar{n})) = I(s)(I(\bar{n})) = I(s)(\psi(n))$.
- $\psi(m + n) = I(\overline{m + n})$. As $\overline{m + n} = \bar{m} + \bar{n}$ is true in $\mathcal{N}$ it is (by Σ_1-completeness) provable in Q and as (D, I) is a model of Q we have $I(\overline{m + n}) = I(\bar{m} + \bar{n})$. Furthermore $I(\bar{m} + \bar{n}) = I(+)(I(\bar{m}), I(\bar{n})) = I(+)(\psi(m), \psi(n))$.
- $\psi(m \cdot n) = I(\cdot)(\psi(m), \psi(n))$ is shown analogously to the case of addition (do it as an exercise!).
- For less-than note that if $m < n$, then $Q \vdash \bar{m} < \bar{n}$ by Σ_1-completeness hence $(D, I) \vDash \bar{m} < \bar{n}$. If, on the other hand, $m \not< n$, then $Q \vdash \neg\bar{m} < \bar{n}$ hence $(D, I) \vDash \neg\bar{m} < \bar{n}$, i.e. $(D, I) \nvDash \bar{m} < \bar{n}$. Therefore $m < n$ iff $(D, I) \vDash \bar{m} < \bar{n}$ iff $(I(\bar{m}), I(\bar{n})) \in I(<)$ iff $(\psi(m), \psi(n)) \in I(<)$.

□

We want to use this result for showing that the completeness theorem fails in second-order logic. There is a subtle issue here that is worth discussing: the completeness theorem can be formulated in two different ways: one *abstract* and one *concrete*. You are probably most familiar with the concrete version which, for a *particular* calculus, says that each valid formula is provable *in this calculus*. For a setting where the completeness theorem fails, one could – in principle – prove the negation of this statement, i.e. show for a particular calculus (like **NK2**) that it does not prove all valid (second-order) formulas. While this result is true it is not very meaningful: it does not rule out the possibility that a simple extension of the calculus by a few more inference rules would result in a complete system.

It is this point where the abstract statement of the completeness theorem becomes important. The abstract version states that the set of valid first-order formulas is *recursively enumerable*. As a reminder, a set is called recursively enumerable if – informally – there is a program that

outputs exactly the elements of the set. A proof calculus can be used as basis for such a program as follows: compute all possible proofs by starting with any axiom and applying iteratively all inference rules to all already generated proofs and for each proof thus generated output the formula it has proved. Therefore the abstract version of the completeness theorem follows from its concrete version.

In our setting of second-order logic – where completeness fails – we will prove the negation of the abstract version because it shows (by the above argument) that there is *no calculus* which is sound and complete for second-order logic. This will be based on the following result.

Theorem 2.5. $\mathrm{Th}(\mathcal{N}) = \{A \textit{ first-order sentence} \mid \mathcal{N} \models A\}$ *is not recursively enumerable.*

Proof Sketch. This follows from Tarski's theorem of the arithmetical undefinability of arithmetical truth or from Gödel's first incompleteness theorem[3]. □

Theorem 2.6. *The set of valid sentences in second-order logic is not recursively enumerable.*

Proof. Let A be a first-order sentence, then we first claim that $\mathcal{N} \models A$ iff $\mathrm{SOA} \to A$ is valid. The right-to-left direction follows directly from the observation that $\mathcal{N} \models \mathrm{SOA}$. For the left-to-right direction suppose $\mathrm{SOA} \to A$ is not valid. Then there is a structure $\mathcal{S}$ with $\mathcal{S} \models \mathrm{SOA}$ but $\mathcal{S} \not\models A$. But the categoricity of SOA then entails $\mathcal{S} \simeq \mathcal{N}$ hence $\mathcal{N} \not\models A$ which finishes the proof of the claim.

For finishing the proof, suppose the set of valid second-order sentences would be recursively enumerable. Take a program P which outputs exactly the valid second-order sentences and apply the following new program to every sentence B output by P: if B is of the form $\mathrm{SOA} \to A$ output A else output nothing. By the above claim this would be a recursive enumeration of $\mathrm{Th}(\mathcal{N})$ which does not exist by Theorem 2.5. Contradiction. □

So the completeness theorem (as the compactness- and the Löwenheim-Skolem-theorem) fails in second-order logic. These failures of important results about the logic are the price we have to pay for its high expressivity.

2.5 Inductive Definitions

In this section we will examine the proof of the categoricity of second-order arithmetic from a more abstract point of view in order to better understand the aspect of second-order logic that we have used for defining the natural numbers. What we will find is a very powerful mechanism: inductive definitions. Once more the pattern repeats: the increase in expressive power goes hand-in-hand with the loss of certain nice properties of the logic – now we are trading completeness for closure under inductive definitions.

Fix a structure $\mathcal{S} = (D, I)$ and observe that

$$\begin{aligned}
&(D,I) \models \mathrm{Ind} \\
\text{iff } &(D,I) \models \forall X\,(\; X(0) \to \forall x\,(X(x) \to X(s(x))) \to \forall y\, X(y)\;) \\
\text{iff } &(D,I) \models \forall y\, \underbrace{\forall X\,(\; X(0) \to \forall x\,(X(x) \to X(s(x))) \to X(y)\;)}_{\mathrm{Nat}} \\
\text{iff } &(D,I) \models \forall y\, \mathrm{Nat}.
\end{aligned}$$

[3] If you are interested in a detailed proof, consult the course notes for "Mathematical Logic 2" or any textboook on mathematical logic.

So a structure where Ind is true is one that satisfies $\forall y\,\mathrm{Nat}$. Nat is a formula with one free variable y. Given a structure $\mathcal{S} = (D, I)$, a formula F with one free variable y defines a subset of D, the set of $m \in D$ s.t. $(D, I \cup \{y \mapsto m\}) \models F$.

Example 2.3. The formula $y \neq 1 \wedge \forall x\,(\exists z\, x \cdot z = y \to x = 1 \vee x = y)$ defines the set of prime numbers in the standard model of arithmetic.

More generally, we can define an n-ary relation by a formula with n free variables as follows:

Definition 2.9. Let (D, I) be a structure and F a formula whose free variable are $y_1, \ldots, y_n$, abbreviated as $\bar{y}$. Then the expression $\lambda\bar{y}.F$ is called *definition of the relation*

$$I(\lambda\bar{y}.F) = \{\bar{m} = (m_1, \ldots, m_n) \in D^n \mid (D, I \cup \{\bar{y} \mapsto \bar{m}\}) \models F\}.$$

Note that $I(\lambda\bar{y}.F) \subseteq D^n$. In this notation, the expression $\lambda y.\mathrm{Nat}$ defines the set

$$I(\lambda y.\mathrm{Nat}) = \{m \in D \mid (D, I \cup \{y \mapsto m\}) \models \forall X\,(X(0) \to \forall x\,(X(x) \to X(s(x))) \to X(y))\}.$$

In other words, an element $m \in D$ is in $I(\lambda y.\mathrm{Nat})$ if it is in every set which contains $I(0)$ and is closed under $I(s)$, i.e. if it is in the smallest set which contains $I(0)$ and is closed under $I(s)$. How can we obtain this set? By a limit process which successively adds elements thus converging towards $I(\lambda y.\mathrm{Nat})$.

Consider the following rules:

$$\frac{}{I(0) \in S} \qquad \frac{m \in S}{I(s)(m) \in S}$$

The set $I(\lambda y.\mathrm{Nat})$ can be obtained as limit of the process which applies these rules to all members of the set thus constructing a sequence $S_0 = \varnothing, S_1, S_2, \ldots$ where S_{i+1} is obtained from S_i by closing it under all rules.

$$\begin{aligned} S_0 &= \varnothing \\ S_1 &= \{I(0)\} \\ S_2 &= \{I(0), I(s)(I(0))\} \\ &\vdots \end{aligned}$$

More abstractly and in the case of an n-ary relation we are dealing with an operator $\varphi : \mathcal{P}(D^n) \to \mathcal{P}(D^n)$ with $\varphi(S_i) = S_{i+1}$.

Definition 2.10. An operator $\varphi : \mathcal{P}(D^n) \to \mathcal{P}(D^n)$ is called *monotone* if $S_1 \subseteq S_2$ implies $\varphi(S_1) \subseteq \varphi(S_2)$.

Definition 2.11. Let $\varphi : \mathcal{P}(D^n) \to \mathcal{P}(D^n)$ be an operator. A set $S \in \mathcal{P}(D^n)$ is called *fixed point of* φ if $\varphi(S) = S$. A fixed point S of φ is called *least fixed point of* φ if every fixed point S' of φ satisfies $S \subseteq S'$.

We will see that every monotone operator has a unique least fixed point. The natural numbers can be defined as the least fixed point of the operator which adds 0 to a set and for every element m of the set adds $m + 1$ to it and many data structures of relevance to computer science have similar inductive definitions.

The definitional principle that is behind the categoricity-theorem proved above then is that second-order logic is closed under least fixed points of monotone operators, i.e. if a monotone operator is second-order definable so is its least fixed point.

From this definition follows directly that if $\varphi : \mathcal{P}(D^n) \to \mathcal{P}(D^n)$ has a least fixed point, then it is unique: for assume φ has two least fixed points S_1 and S_2, then $S_1 \subseteq S_2$ because S_1 is least fixed point and $S_2 \subseteq S_1$ because S_2 is least fixed point and hence $S_1 = S_2$.

Theorem 2.7. *Every monotone operator $\varphi : \mathcal{P}(D^n) \to \mathcal{P}(D^n)$ has a least fixed point.*

Proof. Let $\mathcal{C} = \{A \subseteq D^n \mid \varphi(A) \subseteq A\}$. Since $D^n \in \mathcal{C}$, $\mathcal{C}$ is non-empty, so $\bigcap \mathcal{C}$ is well-defined. Let $R = \bigcap \mathcal{C}$. Let us first prove that R is a fixed point of φ. To that aim, let $S \in \mathcal{C}$, then $\varphi(S) \subseteq S$. Moreover, $R \subseteq S$ and, by monotonicity of φ, $\varphi(R) \subseteq \varphi(S)$. Therefore $\varphi(R) \subseteq S$ and, since S was arbitrary, $\varphi(R) \subseteq R$. Moreover, again by monotonicity of φ, we have $\varphi^2(R) \subseteq \varphi(R)$. So $\varphi(R) \in \mathcal{C}$ and hence $R \subseteq \varphi(R)$. Therefore, $\varphi(R) = R$, i.e., R is a fixed point of φ. Now, suppose that $R' \subseteq D^n$ is a fixed point of φ. Then $R' \in \mathcal{C}$ and since $R = \bigcap \mathcal{C}$ we have $R \subseteq R'$, so R is the least fixed point of φ. □

For a monotone operator $\varphi : \mathcal{P}(D^n) \to \mathcal{P}(D^n)$ we write $\mathrm{lfp}(\varphi)$ for its least fixed point. Note furthermore that for monotone φ we have $\mathrm{lfp}(\varphi) = \bigcap_{\varphi(R)=R} R$. The left-to-right direction follows from $\mathrm{lfp}(\varphi)$ being contained in all R s.t. $\varphi(R) = R$. The right-to-left inclusion follows from $\mathrm{lfp}(\varphi)$ being a fixed point and hence being in the intersection.

Example 2.4. Let $D = \mathbb{Z}$ and let φ be the operator which to a given $S \subseteq \mathbb{Z}$ adds 0 and $x + 1$ for all $x \in S$, i.e.

$$\varphi(S) = S \cup \{0\} \cup \{x + 1 \mid x \in S\}.$$

Then φ is monotone and the fixed points of φ are the sets of the form $[-k, \infty[\subseteq \mathbb{Z}$ for $k \in \mathbb{N}$ and $\mathbb{Z}$ itself. The least fixed point is $\mathrm{lfp}(\varphi) = \mathbb{N}$.

The above observations are of a purely algebraic nature. The fact that a monotone operator has a unique least fixed point does not rely on logic. But we return to logic now by considering definitions of operators in second-order logic.

Definition 2.12. Let (D, I) be a structure and F be a formula whose free variables are X/n and $\bar{y} = y_1, \ldots, y_n$. Then the expression $\lambda X \lambda \bar{y}.F$ is called *definition of the operator* $\varphi : \mathcal{P}(D^n) \to \mathcal{P}(D^n)$ given by

$$\varphi(R) = (I \cup \{X \mapsto R\})(\lambda \bar{y}.F).$$

We also write $I(\lambda X \lambda \bar{y}.F)$ for this operator φ.

Example 2.5. The operator φ of Example 2.4 can be defined by

$$\lambda X \lambda y.\ y \in X \vee y = 0 \vee \exists z \in X\ y = s(z).$$

The main theorem of this section is the following:

Theorem 2.8. *Let $\mathcal{S} = (D, I)$ be a structure and $\varphi : \mathcal{P}(D^n) \to \mathcal{P}(D^n)$ be a monotone operator which is second-order definable, then* $\mathrm{lfp}(\varphi)$ *is second-order definable.*

Before we prove it we need two auxiliary observations on set-definitions.

Lemma 2.1. *Let $\mathcal{S} = (D, I)$ be a structure and A be a formula with X/n and $\bar{z} = z_1, \ldots, z_n$ as only free variables, then*

$$I(\lambda \bar{z}.\forall X\ A) = \bigcap_{R \subseteq D^n} (I \cup \{X \mapsto R\})(\lambda \bar{z}.A)$$

Proof. We have

$$\begin{aligned} \bar{m} \in I(\lambda \bar{z}.\forall X A) \text{ iff } & (D, I \cup \{\bar{z} \mapsto \bar{m}\}) \models \forall X\ A \\ & \text{iff for all } R \subseteq D^n : (D, I \cup \{\bar{z} \mapsto \bar{m}, X \mapsto R\}) \models A \\ & \text{iff for all } R \subseteq D^n : \bar{m} \in (I \cup \{X \mapsto R\})(\lambda \bar{z}.A) \\ & \text{iff } \bar{m} \in \bigcap_{R \subseteq D^n} (I \cup \{X \mapsto R\})(\lambda \bar{z}.A). \end{aligned}$$

□

Lemma 2.2. *Let $\mathcal{S} = (D, I)$ be a structure, C be a sentence and A be a formula with $\bar{z} = z_1, \ldots, z_n$ as only free variables, then*

$$I(\lambda \bar{z}.C \to A) = \begin{cases} I(\lambda \bar{z}.A) & \text{if } (D, I) \models C \\ D^n & \text{otherwise} \end{cases}$$

Proof. If $(D, I) \models C$, then

$$\begin{aligned} \bar{m} \in I(\lambda \bar{z}.C \to A) &\text{ iff } (D, I \cup \{\bar{z} \mapsto \bar{m}\}) \models C \to A \\ &\text{iff } (D, I \cup \{\bar{z} \mapsto \bar{m}\}) \models A \\ &\text{iff } \bar{m} \in I(\lambda \bar{z}.A). \end{aligned}$$

If $(D, I) \not\models C$, then $(D, I \cup \{\bar{z} \mapsto \bar{m}\}) \models C \to A$ for all $\bar{m}$ and hence $I(\lambda \bar{z}.C \to A) = D^n$. □

Proof of Theorem 2.8. Let $\lambda X \lambda \bar{y}.F$ be a definition of φ, then $\bar{y} = y_1, \ldots, y_n$ and we have

$$\varphi(R) = (I \cup \{X \mapsto R\})(\lambda \bar{y}.F).$$

Define a formula

$$\text{FP} \;\equiv\; \forall \bar{y}\ (F \leftrightarrow X(\bar{y}))$$

and observe that FP has X/n as its only free variable. Now

$$(D, I \cup \{X \mapsto R\}) \models \forall \bar{y}\ (F \leftrightarrow X(\bar{y}))$$

is equivalent[4] to

$$\bar{m} \in (I \cup \{X \mapsto R\})(\lambda \bar{y}.F) \quad \text{iff} \quad \bar{m} \in (I \cup \{X \mapsto R\})(\lambda \bar{y}.X(\bar{y}))$$

which is in turn equivalent to

$$\bar{m} \in \varphi(R) \quad \text{iff} \quad \bar{m} \in R$$

and hence we obtain:

$$(D, I \cup \{X \mapsto R\}) \models \text{FP iff } \varphi(R) = R. \tag{*}$$

Towards defining the least fixed point, let

$$\text{LFP} \;\equiv\; \forall X\,(\text{FP} \to X(\bar{z}))$$

which has only $\bar{z}$ as free variables. Then by Lemma 2.1

$$I(\lambda \bar{z}.\text{LFP}) = \bigcap_{R \subseteq D^n} (I \cup \{X \mapsto R\})(\lambda \bar{z}.\text{FP} \to X(\bar{z})).$$

Now fixing an $R \subseteq D^n$ we have

$$(I \cup \{X \mapsto R\})(\lambda \bar{z}.\text{FP} \to X(\bar{z})) = \begin{cases} (I \cup \{X \mapsto R\})(\lambda \bar{z}.X(\bar{z})) & \text{if } (D, I \cup \{X \mapsto R\}) \models \text{FP} \\ D^n & \text{otherwise} \end{cases}$$

by Lemma 2.2 and

$$= \begin{cases} R & \text{if } \varphi(R) = R \\ D^n & \text{otherwise} \end{cases}.$$

[4] Exercise: show this equivalence in detail.

by $(*)$. Therefore

$$I(\lambda\bar{z}.\mathrm{LFP}) \quad = \quad \bigcap_{\substack{R\subseteq D^n \\ \varphi(R)=R}} R \quad = \quad \mathrm{lfp}(\varphi).$$

□

For $\lambda X\lambda\bar{y}.F$ being the definition of a monotone operator φ, define the formula

$$\mu X.F \;\equiv\; \forall X\,(\forall\bar{y}(F \leftrightarrow X(\bar{y})) \rightarrow X(\bar{z}))$$

which has $\bar{z}$ as only free variables and satisfies $I(\lambda\bar{z}\mu X.F) = \mathrm{lfp}(\varphi)$ as shown in the above theorem. This is a common notation for the least fixed point in the literature.

Example 2.6. Continuing Examples 2.4 and 2.5 we have

$$I(\lambda\bar{z}\mu X.F) = I(\lambda\bar{z}.\forall X\,(\forall y((y \in X \vee y = 0 \vee \exists z \in X\, y = s(z)) \leftrightarrow y \in X) \rightarrow z \in X)) = \mathbb{N}$$

Let us now go back to the inductive definition of the set of natural numbers in order to make a proof-theoretic observation.

$$\mathrm{Nat} \;\equiv\; \forall X\,(\; 0 \in X \rightarrow \forall x \in X\, s(x) \in X \rightarrow y \in X\;)$$

From the proof-theoretic point of view the above definition provides the ability to prove a statement by induction in the following form:

$$\dfrac{\dfrac{\dfrac{\vdots}{\forall x\,(A(x) \rightarrow A(s(x)))} \qquad \dfrac{\dfrac{\vdots}{A(0)} \qquad \dfrac{[x_0 \in \mathrm{Nat}]^1}{A(0) \rightarrow \forall x\,(A(x) \rightarrow A(s(x))) \rightarrow A(x_0)}\,\forall_{\mathrm{E}}}{\forall x\,(A(x) \rightarrow A(s(x))) \rightarrow A(x_0)}\,\rightarrow_{\mathrm{E}}}{A(x_0)}\,\rightarrow_{\mathrm{E}}}{\dfrac{x_0 \in \mathrm{Nat} \rightarrow A(x_0)}{\forall x \in \mathrm{Nat}\, A(x)}\,\forall_{\mathrm{I}}}\,\rightarrow^1_{\mathrm{I}}$$

We can continue to define other data types like trees, etc. in a similar fashion. For defining functions we now consider a mechanism that allows to introduce a new function symbol after having specified a functional relationship. As a reminder, the graph of a function $f : X^n \rightarrow Y$ is the set

$$\{(\bar{x}, y) \in X^n \times Y \mid f(\bar{x}) = y\}$$

Our logical setting allows inductive definitions of sets and in order to mimic a recursive definition of a function one can proceed as follows:

Definition 2.13. Let T be a second-order theory[5] in some language L. If $T \vdash \forall\bar{x}\,\exists!y\,F(\bar{x}, y)$ for a formula F whose free variables are among $\bar{x}, y$ then we define a *functional extension* by adding a new function symbol f and defining T' as the deductive closure of

$$T \cup \{\forall\bar{x}\forall y\,(F(\bar{x}, y) \leftrightarrow f(\bar{x}) = y)\}.$$

This provides a mechanism for defining new functions. One way of defining a new function is by recursion. For example, Q contains axioms that define addition in terms of the successor. To mimic such a definition in our setting we proceed as follows

[5]Reminder: a theory is a deductively closed set of sentences – in the case of second-order logic deductively closed means that by **NK2** no new consequences can be derived

1. Give an inductive definition $F(\bar{x}, y)$ of the graph of the function.

2. Give an **NK2**-proof of $\forall \bar{x} \exists! y\, F(\bar{x}, y)$.

3. Obtain a new function symbol f by taking the functional extension T' of T by f.

An advantage of this procedure is the above definition of a functional extension is all we need beyond plain **NK2**.

Remark 2.1. An alternative to the above notion of functional extension would be to enrich our object-level language by a *selection operator* ι which is given a predicate as argument and will return an object satisfying this predicate (if there is any such object – if not, it may return anything). For example, the predecessor function p could be defined by

$$p \;\equiv\; \lambda x.\iota(\lambda y.s(y) = x)$$

as the function which, given x as input will return a y for which the property $s(y) = x$ holds (if there is any such y).

However, we refrain from using selection operators right now as they require a tighter integration with the lambda calculus.

Exercise 2.6. *In this exercise we work in the language of rings* $L = \{0/0, 1/0, +/2, -/1, \cdot/2, {}^{-1}/1\}$ *and in the structure* $(\mathbb{R}, I)$ *where* I *is the standard-interpretation of* L *with the convention that* $I(0^{-1}) = I(0)$ *to ensure totality of the multiplicative inverse. Let* $\varphi : \mathcal{P}(\mathbb{R}) \to \mathcal{P}(\mathbb{R})$ *be the mapping which adds to a given* $S \subseteq \mathbb{R}$ *the elements* $0, 1, x + y, x \cdot y, -x, x^{-1}$ *for all* $x, y \in S$.

1. *Show that* φ *is a monotone operator.*

2. *Find a formula* F *which defines* φ.

3. *What is the least fixed point of* φ*?*

4. *Find a formula* G *which defines the least fixed point of* φ.

5. *Are there other fixed points? If yes, give at least one. If no, show why.*

Exercise 2.7. *Let* $\mathcal{S} = (D, I)$ *be a structure, let* F_1, F_2 *be formulas whose only free variables are* $\bar{y} = y_1, \ldots, y_n$. *Show that*

1. $I(\lambda \bar{y}.F_1 \wedge F_2) = I(\lambda \bar{y}.F_1) \cap I(\lambda \bar{y}.F_2)$

2. $I(\lambda \bar{y}.F_1 \vee F_2) = I(\lambda \bar{y}.F_1) \cup I(\lambda \bar{y}.F_2)$

Let F *be a formula whose only free variables are* $\bar{y} = y_1, \ldots, y_n$ *and* x. *Show that*

3. $I(\lambda \bar{y}.\forall x\, F) = \bigcap_{m \in D}(I \cup \{x \mapsto m\})(\lambda \bar{y}.F)$

4. $I(\lambda \bar{y}.\exists x\, F) = \bigcup_{m \in D}(I \cup \{x \mapsto m\})(\lambda \bar{y}.F)$

A formula is said to be in negation-normal form (NNF) *if it does not contain implication and negation appears only immediately above atoms. It is well known that every formula can be transformed into a logically equivalent NNF.*

5. *Let F be a formula in NNF without second-order quantifiers s.t. $\lambda X \lambda \bar{y}.F$ defines an operator $\varphi : \mathcal{P}(D^n) \to \mathcal{P}(D^n)$ and no occurrence of X in F appears below a negation. Show that φ is monotone.*
   Hint: proceed by induction on F

Exercise 2.8. *We work in the language $L = \{0/0, s/1\}$ and the theory T, defined as deductive closure of the axioms*

$$\forall y\, \mathrm{Nat} \qquad \forall x\, 0 \neq s(x) \qquad \forall x \forall y\, (s(x) = s(y) \to x = y)$$

In this exercise we will do some programming based on this theory. The primitive recursive definition of addition of natural numbers is

$$x + 0 \to x, \quad x + s(y) \to s(x + y)$$

Our aim for this exercise is to obtain a new theory T' where the addition-function is available and then to prove properties of addition.

To that aim, first define the graph of the addition function*:*

$$\mathrm{Add}(x, y, z) \;\equiv\; \forall X\, (\forall x'\, X(x', 0, x') \to \forall x' y' z'\, (X(x', y', z') \to X(x', s(y'), s(z'))) \to X(x, y, z))$$

Give formal proofs of

1. $\forall x\, \mathrm{Add}(x, 0, x)$ *and*
2. $\forall x \forall y \forall z\, (\mathrm{Add}(x, y, z) \to \mathrm{Add}(x, s(y), s(z)))$.

From now on we assume that also $T \vdash \forall x \forall y \exists! z\, \mathrm{Add}(x, y, z)$ (but showing this is not part of this exercise). This allows to add a new binary function symbol $+$, written in infix notation, to obtain the theory T' as deductive closure of T and $\{\forall x \forall y \forall z\, (\mathrm{Add}(x, y, z) \leftrightarrow x + y = z)\}$.

Give formal proofs in T' of

3. $\forall x\, x + 0 = x$ *and*
4. $\forall x \forall y\, x + s(y) = s(x + y)$.

Using this procedure we can extend our working theory by any function which has a primitive recursive definition. Beyond **NK2** *this procedure only uses the functional extension of a theory as described in the course notes. Once we have extended the theory by a newly defined function (such as $+$) we can prove properties of this function.*

Give a formal proof in T' of

5. $\forall x_1 \forall x_2\, x_1 + x_2 = x_2 + x_1$
   Hint: Establish the symmetric versions of properties 3. and 4. above first by induction: $\forall x\, 0 + x = x$ and $\forall x \forall y\, s(x) + y = s(x + y)$. Then do an induction on either x_1 or x_2.

Chapter 3

Higher-Order Logic

3.1 The Untyped Lambda Calculus

The untyped lambda calculus is an abstract notation for functions and is based on two fundamental operations on functions and terms. The first of them is **application**: FA denotes the application of the function F to the argument A. The second is **abstraction**: given a term A (usually – but not necessarily – containing x) we can form the function which maps a given x to A by writing $\lambda x.A$.

For example, from the term $x^3 + 1$ we can build the function $\lambda x.x^3 + 1$ which maps x to $x^3 + 1$. Applying this function to 2 is written as the term $(\lambda x.x^3 + 1)2$ which we will want to *reduce* to $2^3 + 1$ by inserting the argument 2 for the parameter x of the function.

Let us now make these ideas precise:

Definition 3.1 (Lambda Terms)**.** The set of lambda terms, denoted Λ, is built up from a countably infinite set of variables V by the following rules:

1. If $x \in V$, then $x \in \Lambda$.
2. If $M, N \in \Lambda$, then $MN \in \Lambda$.
3. If $x \in V$ and $M \in \Lambda$, then $\lambda x.M \in \Lambda$.

Example 3.1. $(\lambda x.yx)z$ is a lambda term.

Parenthesis associate to the left, so MNO abbreviates $(MN)O$. The scope of a lambda-binding extends as far to the right as possible, so $\lambda x.MN$ abbreviates $\lambda x.(MN)$ (but **not** $(\lambda x.M)N$). We often abbreviate $\lambda x.\lambda y.M$ by $\lambda xy.M$. The notions of free and bound variables are defined as usual (with λ being the only binder).

Definition 3.2 (β-Reduction)**.** The rule of β-reduction is $(\lambda x.M)N \mapsto_\beta M[x\backslash N]$. We write $\rightarrow_\beta$ for the reflexive and transitive closure of $\mapsto_\beta$.

A term of the form $(\lambda x.M)N$ is called a *β-redex* and a term which does not contain a redex is called β-normal form. A normal form cannot be further reduced.

Definition 3.3 (α-Equivalence)**.** Two lambda terms are called α-equivalent if they only differ in naming of bound variables, this is written as $M =_\alpha N$.

For example $\lambda xy.yx$ and $\lambda xy'.y'x$ are α-equivalent. We will identify terms which are α-equivalent and allow renaming of bound variables at any time. During β-reduction this may indeed sometimes become necessary as the following β-reduction sequence shows

$$(\lambda x.xx)(\lambda yz.yz) \to_\beta (\lambda yz.yz)(\lambda yz.yz) \to_\beta \lambda z.(\lambda yz.yz)z =_\alpha \lambda z.(\lambda yz'.yz')z \to_\beta \lambda zz'.zz'$$

The following theorem shows that the lambda calculus is confluent, this is also sometimes called the Church-Rosser property.

Theorem 3.1. *If $M \to_\beta M_1$ and $M \to_\beta M_2$ then there is an N s.t. $M_1 \to_\beta N$ and $M_2 \to_\beta N$*

Proof. Without proof (see a course on the lambda calculus). □

There are terms whose reduction does not terminate:

$$(\lambda x.xx)(\lambda x.xx) \to_\beta (\lambda x.xx)(\lambda x.xx) \to_\beta \cdots$$

A way to avoid this is to forbid self-application as in the term xx by introducing types, we will see how to do this soon.

The lambda calculus is a very strong formalism. It allows to represent all computable functions. To show this we need a representation of natural numbers in the lambda calculus, a good way to do this are *Church numerals*. Define the n-fold application of a term F to a term M, written as F^nM by $F^0M \equiv M$ and $F^{n+1}M \equiv F(F^nM)$. Then the lambda term representing $n \in \mathbb{N}$ is $\bar{n} \equiv \lambda fx.f^nx$.

Example 3.2. $\bar{3} = \lambda fx.f(f(fx))$

Theorem 3.2. *If $f : \mathbb{N} \hookrightarrow \mathbb{N}$ is computable*[1]*, then there is a lambda term T s.t. $T\bar{n} \to_\beta \bar{m}$ iff $f(n) = m$.*

Proof. Without proof (see a course on the lambda calculus). □

Exercise 3.1. *Let $A \equiv \lambda mnfx.mf(nfx)$ and $E \equiv \lambda xy.yx$.*

1. *Compute the β-normal form of $A\bar{1}\bar{2}$.*
2. *Compute the β-normal form of $A\bar{3}\bar{1}$.*
3. *Compute the β-normal form of $E\bar{2}\bar{3}$.*
   Hint: it may be helpful to split this reduction using several auxiliary calculations.
4. *Show that $A\bar{m}\bar{n} \to_\beta \overline{m+n}$ for all $m, n \in \mathbb{N}$.*

3.2 Simple Type Theory

The lambda calculus allows to apply terms to themselves, like in the case xx. In some situations this is natural: for example a C-compiler which is written in C can take its own code as input. In other situations it is not, for example the expression cos(cos) (intended to denote application of the cosine function to itself) does not make sense, but $\cos(\pi)$ does. The reason for this is that $\pi \in \mathbb{R}$ and $\cos : \mathbb{R} \to \mathbb{R}$, more verbosely: cos is a function taking a real number as argument so $\cos(\pi)$ is well-typed but cos(cos) is not because cos is not a real number. This observation builds the basis for the simply typed lambda calculus.

[1] in other terminology: partial recursive

Definition 3.4 (simple types)**.** The set of simple types $\mathbb{T}$ is defined inductively as follows:

1. $\iota \in \mathbb{T}$ (the *type of individuals*)
2. $o \in \mathbb{T}$ (the *type boolean*)
3. If $\tau, \sigma \in \mathbb{T}$, then $\tau \to \sigma \in \mathbb{T}$ (the *type of functions from* τ *to* σ)

Types – as implications – associate to the right, so $\tau \to \sigma \to \rho$ abbreviates $\tau \to (\sigma \to \rho)$.

Example 3.3. We can introduce "set" as abbreviation of $\iota \to o$. An operator such as those considered in the previous section is a function that maps sets to sets and hence of type set $\to$ set, i.e. $(\iota \to o) \to (\iota \to o)$.

Variables will directly be annotated with a type using the notation x^τ for $x \in V$ and $\tau \in \mathbb{T}$. Consequently two variables of different type are always considered different and two variables of the same type are the same iff they have the same underlying untyped variable. An expression of the form $M : \tau$ is called *type judgment* and is read as "M is of type τ".

Definition 3.5. The logical constants are:

1. $\supset : o \to o \to o$
2. $\forall_\tau : (\tau \to o) \to o$ for each type $\tau \in \mathbb{T}$.

Definition 3.6. A simply typed language L is a set of typed constants (different from the logical constants).

Example 3.4. In our type-theoretic setting, the language of arithmetic is $L = \{0 : \iota, s : \iota \to \iota, + : \iota \to \iota \to \iota, \cdot : \iota \to \iota \to \iota, <: \iota \to \iota \to o\}$.

Definition 3.7 (expressions of simple type theory)**.** Let V be a countably infinite set of variables and let L be a simply typed language. M is called *L-expression of simple type theory* if $M : \tau$ is derivable by the following rules for some $\tau \in \mathbb{T}$.

$$c : \tau \text{ if } c : \tau \in L \text{ or } c : \tau \text{ is a logical constant}$$

$$x^\tau : \tau \text{ for all } x \in V \text{ and } \tau \in \mathbb{T}$$

$$\frac{M : \sigma \to \tau \quad N : \sigma}{MN : \tau} \qquad \frac{M : \tau}{\lambda x^\sigma . M : \sigma \to \tau}$$

Derivations in this calculus are called *type derivations*. A *formula* is an expression of type o.

Example 3.5. Let $L = \{P : \iota \to o, Q : \iota \to o, f : \iota \to \iota\}$, the following is a type derivation.

$$\dfrac{\forall_\iota : \iota \qquad \dfrac{\dfrac{\supset : o \to o \to o \qquad \dfrac{P : \iota \to o \quad x^\iota : \iota}{Px^\iota : o}}{\supset (Px^\iota) : o \to o} \quad \dfrac{Q : \iota \to o \qquad \dfrac{f : \iota \to \iota \quad x^\iota : \iota}{fx^\iota : \iota}}{Q(fx^\iota) : o}}{\supset (Px^\iota)Q(fx^\iota) : o}}{\lambda x^\iota . \supset (Px^\iota)Q(fx^\iota) : \iota \to o}}{\forall_\iota \lambda x^\iota . \supset (Px^\iota)Q(fx^\iota) : o}$$

This type derivation shows that $\forall_\iota \lambda x^\iota . \supset (Px^\iota)Q(fx^\iota)$ is a formula. In the standard notation of first-order logic this is written as: $\forall x\, (P(x) \to Q(f(x)))$.

The extension of the semantics from first- to second-order logic was rather straightforward in that we have taken set quantifiers to quantify over sets of the domain and function quantifiers to quantify over functions. The extension to higher-order logic is as straightforward: having quantification over any simple type we interpret a quantifier of type τ by an object of type τ. For example a quantifier of type $\iota \to o$ will just be the set quantifier $\forall X$ familiar from second-order logic. A quantifier of type $(\iota \to o) \to o$ quantifies over all sets of sets, etc.

Definition 3.8. Let D be a set. Define

1. $D_\iota = D$,
2. $D_o = \{\text{true}, \text{false}\}$ where true and false are assumed to not occur in D, and
3. $D_{\rho\to\sigma}$ as the set of functions from D_ρ to D_σ.

Definition 3.9. Let L be a simply typed language. An *L-structure* is a pair $\mathcal{S} = (D, I)$ where D is a set and I maps each L-constant of type τ to an element of D_τ.

Definition 3.10. Let $\mathcal{S} = (D, I)$ be an L-structure. We extend the interpretation I from the constants of the language to all expressions as follows:

The interpretation of the logical constants is:

$$I(\supset)(a,b) = \begin{cases} \text{true} & \text{if } b = \text{true or } a = \text{false} \\ \text{false} & \text{otherwise} \end{cases}$$
$$I(\forall_\tau)(F) = \begin{cases} \text{true} & \text{if } F(m) = \text{true for all } m \in D_\tau \\ \text{false} & \text{otherwise} \end{cases}$$

The expressions are interpreted as:

$$I(MN) = I(M)(I(N))$$
$$I(\lambda x^\tau.M)(m) = (I \cup \{x^\tau \mapsto m\})(M)$$

For $M : o$ we write $\mathcal{S} \vDash M$ for $I(M) = \text{true}$.

The above choice of logical constants suffices to define all others *inside the system.*

Example 3.6. Define

$$\begin{aligned} \bot &\equiv \forall_o \lambda x^o.x^o, \text{ and} \\ \neg &\equiv \lambda x^o. \supset x^o \bot. \end{aligned}$$

Observe that $\bot : o$ and $\neg : o \to o$ as the following type derivations show:

$$\dfrac{\forall_o : (o \to o) \to o \quad \dfrac{x^o : o}{\lambda x^o.x^o : o \to o}}{\forall_o \lambda x^o.x^o : o} \qquad \dfrac{\dfrac{\dfrac{\supset : o \to o \to o \quad x^o : o}{\supset x^o : o \to o} \quad \bot : o}{\supset x^o \bot : o}}{\lambda x^o. \supset x^o \bot : o \to o}$$

Also the semantics of these defined logical constants has the expected behavior. Let $\mathcal{S} = (D, I)$ be a structure and observe that $I(\bot) = \text{false}$ and $I(\neg)$ is negation:

$$I(\bot) = I(\forall_o \lambda x^o.x^o) = I(\forall_o)(I(\lambda x^o.x^o)) = I(\forall_o)(\text{id}_o) = \begin{cases} \text{true} & \text{if } \text{id}_o(b) = \text{true for all } b \in D_o \\ \text{false} & \text{otherwise} \end{cases}$$
$$= \text{false}$$

and

$$I(\neg)(b) = \underbrace{(I \cup \{x^o \mapsto b\})}_{I'}(\supset x^o \bot) = \begin{cases} \text{true} & \text{if } I'(\bot) = \text{true or } I'(x^o) = \text{false} \\ \text{false} & \text{otherwise} \end{cases}$$

$$= \begin{cases} \text{true} & \text{if } b = \text{false} \\ \text{false} & \text{if } b = \text{true} \end{cases}$$

To ease notation we will often omit type information on bound variables, so that e.g. $\lambda y^{\iota\to\iota} x^\iota . y^{\iota\to\iota} x^\iota$ is more briefly written as $\lambda y^{\iota\to\iota} x^\iota . yx$. We also write binary logical operations in the usual infix notation, e.g. $A \supset B$ instead of $\supset AB$ and quantifiers in the familiar way as $\forall x^\tau A$ instead of $\forall_\tau \lambda x^\tau . A$.

Exercise 3.2. *We have defined $\bot$ and $\neg$ as simply typed expressions. Analogously to these, define binary disjunction $\vee$, binary conjunction $\wedge$, logical equivalence $\leftrightarrow$ and the existential quantifier $\exists_\tau$ as simply typed expression. Give type derivations and show that the semantics has the expected behavior.*

3.3 Proofs

Theorem 3.3 (Strong Normalization)**.** *If M is a simply typed expression, then every sequence of β-reductions that starts with M eventually terminates.*

Proof. Without proof (see a course on the lambda calculus). □

This theorem states that *all* β-reduction sequences of a simply typed lambda term are finite. There are reduction systems that only possess the weaker property that *there is* a finite reduction sequence. This latter property is called *weak normalization.*

This theorem also shows that, in particular, the non-terminating lambda term $(\lambda x.xx)(\lambda x.xx)$ is not typable. Indeed, in a certain sense, the very point of types is that they guarantee termination. By strong normalization and confluence every term has exactly one beta-normal form. Hence inter-reducibility by beta-reduction and alpha-renaming is a decidable equivalence relation and will be denoted by $=_{\alpha\beta}$.

We are now in a position to define a proof system **NK**ω for simple type theory. The rules are:

$$\frac{\begin{array}{c}\vdots\\ M\end{array}}{\begin{array}{c}\vdots\\ N\end{array}} =_{\alpha\beta}$$

whenever $M =_{\alpha\beta} N$. For the propositional symbols we have

$$\frac{\begin{array}{c}\vdots\\ M\end{array} \quad \begin{array}{c}\vdots\\ M \supset N\end{array}}{\begin{array}{c}\vdots\\ N\end{array}} \supset_{\mathrm{E}} \qquad \frac{\begin{array}{c}[M]^i\\ \vdots\\ N\end{array}}{\begin{array}{c}\vdots\\ M \supset N\end{array}} \supset^i_{\mathrm{I}} \qquad \frac{\begin{array}{c}\vdots\\ \neg(\neg M)\end{array}}{\begin{array}{c}\vdots\\ M\end{array}} \text{ dne}$$

where the double negation elimination is responsible for obtaining classical (and not intuitionistic logic). For the quantifier we have the rules:

$$\frac{\begin{array}{c}\vdots\\ \forall x^\tau A\end{array}}{\begin{array}{c}\vdots\\ A[x^\tau \backslash M]\end{array}} \forall_{\mathrm{E}}$$

for an expression M of type τ and

$$\dfrac{\begin{array}{c}\vdots\ \pi\\ A[x^\tau \backslash x_0^\tau]\end{array}}{\forall x^\tau\ A}\ \forall_{\mathrm{I}}$$

if x_0^τ does not occur in A nor in the open assumptions of π. Equality reasoning in **NK**ω is done using Leibniz equality by defining:

$$=_\tau \quad \equiv \quad \lambda x_1^\tau x_2^\tau . \forall Y^{\tau\to o}(Yx_1 \leftrightarrow Yx_2)$$

and adding the exioms of propositional extensionality

$$\forall x^o \forall y^o((x \leftrightarrow y) \leftrightarrow x =_o y)$$

and functional extensionality

$$\forall f^{\tau\to\sigma} \forall g^{\tau\to\sigma}((\forall x^\tau fx =_\sigma gx) \supset f =_{\tau\to\sigma} g).$$

Exercise 3.3. *Show that the usual rules for* $\bot$, $\neg$, $\vee$, $\wedge$, $\leftrightarrow$ *and* $\exists_\tau$ *(as defined in Exercise 3.2) are derivable in* **NK**ω.

Exercise 3.4. *Show that the following formulas are provable in* **NK**ω *for arbitrary types* τ, σ*:*

$$\begin{aligned}
&\textit{reflexivity: } \forall x^\tau\ x =_\tau x\\
&\textit{symmetry: } \forall x^\tau \forall y^\tau\ (x =_\tau y \supset y =_\tau x)\\
&\textit{transitivity: } \forall x^\tau \forall y^\tau \forall z^\tau\ (x =_\tau y \supset y =_\tau z \supset x =_\tau z)\\
&\textit{compatibility: } \forall f^{\tau\to\sigma} \forall x^\tau \forall y^\tau\ (x =_\tau y \supset fx =_\sigma fy)
\end{aligned}$$

Exercise 3.5. *Define set-complementation* comp *as a simply typed expression. Give a type derivation and show that the semantics has the expected behavior. Define a simply typed formula* F *that states: "for any set of individuals* X*: the complement of the complement of* X *is* X*". Give a type derivation for* F*. Prove* F *in* **NK**ω*. You may assume at your disposal the usual inference rules for* $\wedge, \vee, \leftrightarrow, \neg, \exists_\tau$ *in* **NK**ω *(see Exercise 3.3).*

3.4 Subsystems

We can now recover first- and second-order logic within simple type theory. A language in both first- and second-order logic consisted of constant symbols, function symbols and predicate symbols. Constant and function symbols have types from

$$T_F = \{\iota, \iota \to \iota, \iota \to \iota \to \iota, \ldots\}$$

while predicate symbols have types from

$$T_P = \{o, \iota \to o, \iota \to \iota \to o, \ldots\}.$$

Definition 3.11. Let A be a simply typed formula s.t. every non-logical symbol occurring in A is of a type in T_F or in T_P. Then A is said to be a *formula of second-order logic*. Furthermore, if all $\forall_\tau$ in A satisfy $\tau = \iota$ then A is said to be a *formula of first-order logic*.

From a purely formal point of view, simple type theory is one formal system, first-order logic another and second-order logic yet another. The above definition could be made more precise by first defining additional connectives like $\wedge, \vee, \exists_\tau$ in simple type theory, then defining (the straightforward) translations from first- and second-order logic to simple type theory and finally showing that these translations preserve truth (and hence validity). For time constraints we refrain from carrying out this in detail here, partially this will be done in the exercises. In the end the moral is that first- and second-order logic can be considered subsystems of simple type theory and in particular: whenever something is first- or second-order definable it is also definable in simple type theory. This leads to the following results:

Theorem 3.4. *The compactness theorem fails in simple type theory.*

Proof. Let L_i be the first-order formula that defines the structures of size at least i and remember that finiteness can be defined by a second-order formula Fin. Let $\Gamma = \{\text{Fin}\} \cup \{L_i \mid i \geqslant 1\}$ and observe that every finite $\Gamma_0 \subseteq \Gamma$ is satisfiable but Γ is not. □

Theorem 3.5. *The Löwenheim-Skolem theorem fails in simple type theory.*

Proof. Remember that there was a second-order formula Countable($\mathcal{U}$) which defines the countable structures. The second-order and hence higher-order formula $\neg$Countable($\mathcal{U}$) is satisfiable but has no countable model, hence the Löwenheim-Skolem theorem fails. □

Theorem 3.6. *The completeness theorem fails in simple type theory.*

Proof. Note that there is a program P which when given a higher-order formula determines whether it is a second-order formula or not (by checking the types of all quantifiers).

Suppose that the valid formulas of higher-order logic are recursively enumerable by some program Q, then the valid formulas of second-order logic are recursively enumerable as well (by using both P and Q). Contradiction. □

3.5 Henkin-Completeness

So, as in second-order logic, the completeness theorem does not hold in simple type theory. Nevertheless, there is a natural proof formalism – **NK**ω – and the question of finding a semantic characterization of the formulas provable in **NK**ω is natural as well. In this section we will sketch such a characterization.

The starting point are **NK**ω-proofs and the observation that one can make explicit (in the form of first-order axioms) the properties of types that are used in such proofs. Formally, this will mean recovering simple type theory as a *first-order theory*, i.e. as a formal system obtained from pure first-order logic by adding these axioms.

To that aim we will define a translation from simply typed formulas to formulas in this first-order theory. The basic idea is to add, for each type $\tau \in \mathbb{T}$ a unary predicate symbol T_τ intended to denote "is of type τ". This will allow to express quantification over higher-types as bounded first-order quantification by translating

$$\forall x^\tau A \text{ to } \forall x\,(T_\tau(x) \to A).$$

In order to carry out this translation we first introduce a version of simple type theory that is based on relations only.

Definition 3.12. *Relational type theory* is defined as follows.

1. ι and o are relational types
2. if $\tau_1, \ldots, \tau_n$ are relational types, then $\tau_1 \to \cdots \to \tau_n \to o$ is a relational type.

A simply typed language L is called *relational* if all its elements have types in $T_F \cup T_R$.
The *relational formulas* are defined as follows:

1. The *terms* of type ι are defined as in first-order logic.
2. The *terms* of a relational type $\sigma \neq \iota$ are the variables of type σ.
3. Let $t_1 : \tau_1, \ldots, t_n : \tau_n$ and $\sigma = \tau_1 \to \cdots \to \tau_n \to o$.
   (a) If $R : \sigma \in L$ then $Rt_1 \cdots t_n$ is an *atom*.
   (b) If X^σ is a variable, then $X^\sigma t_1 \cdots t_n$ is an *atom*.
4. *Formulas* are obtained from atoms by closing under $\supset$ and $\forall_\sigma$ for all relational σ.

Proposition 3.1. *For every formula A there is a relational formula A' s.t.* $\mathbf{NK}\omega \vdash A \leftrightarrow A'$.

Proof Sketch. The central idea of the proof is that a function $f : \tau \to \sigma$ can be represented by a relation $R : \tau \to \sigma \to o$ by defining $R(x, y)$ iff $f(x) = y$ and hence quantification of arbitrary types can be replaced by quantification over relational types. □

Definition 3.13. Let L be a relational language. Define the following first-order language L^{s}:

1. If $c : \tau \in L$, then $\tau \in T_F \cup T_R$ and put $c \in L^{\mathrm{s}}$.
2. For each relational type σ add a unary predicate symbol T_σ (whose intended interpretation is "is of type σ") to L^{s}.
3. For each relational type $\sigma = \tau_1 \to \cdots \to \tau_n \to o$ add an $n+1$-ary predicate symbol E_σ (whose intended interpretation is "is element of").

Definition 3.14. Define a translation of a relational formula A to a first-order L^{s}-formula A^{s} recursively as follows:

$$\begin{aligned}
(x^\sigma t_1 \cdots t_n)^{\mathrm{s}} &\equiv E_\sigma(t_1, \ldots, t_n, x) \text{ where } \sigma = \tau_1 \to \cdots \to \tau_n \to o \\
A^{\mathrm{s}} &\equiv A \text{ for another atomic formula } A \\
(A \supset B)^{\mathrm{s}} &\equiv A^{\mathrm{s}} \to B^{\mathrm{s}} \\
(\forall x^\sigma\, A)^{\mathrm{s}} &\equiv \forall x\, (T_\sigma(x) \to A^{\mathrm{s}})
\end{aligned}$$

Definition 3.15. Let L be a relational language and define the first-order theory TT_0 in L^{s} by the following axioms:

1. $T_\tau(c)$ if $c : \tau \in L$.
2. disjointness: $\forall x\, (T_\tau(x) \to \neg T_\sigma(x))$ if $\tau \neq \sigma$.

..., see Leivant-article for the full list.

Furthermore, define the comprehension axioms: let A be a higher-order formula of type o with free variables $x_1 : \tau_1, \ldots, x_n : \tau_n$. Let $\sigma = \tau_1 \to \cdots \to \tau_n \to o$. Then the comprehension axiom for A is the sentence

$$\exists Y^\sigma \forall x_1^{\tau_1} \cdots \forall x_n^{\tau_n} \; (Y x_1 \cdots x_n \leftrightarrow A)$$

Let TT be the theory obtained by adding to TT_0 all s-translations of all comprehension axioms.

Example 3.7. Comprehension axioms are essential for mathematics, they allow to define sets such as for example the set of even numbers:

$$\exists Y^{\iota \to o} \forall x^\iota \, (Yx \leftrightarrow \exists z^\iota \, 2 \cdot z = x)$$

Lemma 3.1. *Let A be a relational formula. Then A is provable in* **NK**ω *iff* $\mathrm{TT} \vdash A^{\mathrm{s}}$.

Without Proof.

The above result characterizes provability in **NK**ω by provability in the *first-order theory* TT. In first-order logic however we *do* have a completeness theorem which, when applied to the above lemma, has the following immediate corollary:

$$\mathrm{TT} \vdash A^{\mathrm{s}} \text{ iff } \mathcal{S} \models \mathcal{A}^{\mathrm{s}} \text{ for all } \mathcal{S} \text{ with } \mathcal{S} \models \mathrm{TT}.$$

This shows that there is a class of structures w.r.t. which **NK**ω is complete, namely the structures $\mathcal{S}$ satisfying TT. Such a structure interprets the translation A^{s} of a simply typed formula A. In the rest of this section we will primarily concern ourselves with how to transfer back the satisfaction of TT to the level of semantics. This will give rise to the notion of Henkin-structure and lead to the characterization of the formulas provable in **NK**ω as those which are true in all Henkin-structures.

The above Lemma 3.1 is an important result for the following reason: the completeness theorem is a strong connection between the syntactic side of proofs and the semantic side of structures. We have seen that the completeness theorem does not survive the transition from first- to second- and higher-order logic – and this despite the fact that this transition is quite natural on both, the syntactic and the semantic side. The above Lemma 3.1 helps to explain *why* this connection breaks down: the generalization of proofs from first- to higher-order logic is, in fact, quite weak: provability in **NK**ω can be characterized by provability in a first-order theory. This is in stark contrast to the semantic side where the interpretation of set-quantifiers as ranging over *all* subsets gives definitional power that does not exist in first-order logic as we had the chance to witness at several occasions in this course.

The central difference between interpreting A and A^{s} is that in the standard semantics of simply typed L-formulas, the interpretation of a type τ is *fixed to be* D_τ. In contrast to that, when interpreting an A^{s} in a structure $\mathcal{S}$, the interpretation of the predicate T_τ *is part of the structure* $\mathcal{S}$.

This observation forms the starting point of the *Henkin semantics* of simple type theory. Instead of fixing the interpretation of a type τ to be D_τ we will make the interpretation of τ part of the structure.

Definition 3.16. Let L be a simply typed language. A *Henkin-L-prestructure* is a pair $\mathcal{H} = (D, I)$ s.t.

- $I(o) = \{\mathrm{true}, \mathrm{false}\}$ and $I(\iota) = D$.
- $I(\tau) \subseteq D_\tau$ for every $\tau \in \mathbb{T}$.

- $I(c) \in I(\tau)$ for every $c : \tau \in L$.

Example 3.8. $(\mathbb{R}, I)$ where $I(o) = \{\text{true}, \text{false}\}$, $I(\iota) = \mathbb{R}$, $I(\iota \to o) =$ the measurable subsets of $\mathbb{R}$, etc.

Definition 3.17. Let $\mathcal{S} = (D, I)$ be a Henkin-L-prestructure. The interpretation I is extended to all simply typed expressions as in the case of L-structures (see Definition 3.10) with the only exception of the universal quantifier which is interpreted as:

$$I(\forall_\tau)(F) = \begin{cases} \text{true} & \text{if } F(m) = \text{true for all } m \in I(\tau) \\ \text{false} & \text{otherwise} \end{cases}$$

As always, when L is irrelevant or clear from the context, we simply speak about a *Henkin-prestructure*.

Definition 3.18. Let $\mathcal{H} = (D, I)$ be a Henkin-L-prestructure, we obtain a first-order L^{s}-structure $\mathcal{H}^{\mathrm{s}} = (D^{\mathrm{s}}, I^{\mathrm{s}})$ as follows:

1. $D^{\mathrm{s}} = \bigcup_{\tau \in \mathbb{T}} I(\tau)$,
2. $I^{\mathrm{s}}(c) = I(c)$ for $c \in L$,
3. $I^{\mathrm{s}}(T_\tau) = I(\tau)$, and
4. $I^{\mathrm{s}}(E_\tau) = \in$.

The above translation turns a Henkin-L-prestructure $\mathcal{H}$ into a first-order L^{s}-structure $\mathcal{H}^{\mathrm{s}}$ and it preserves truth in the following sense:

Lemma 3.2. *Let $\mathcal{H}$ be a Henkin-L-prestructure and A be a simply typed L-formula. Then $\mathcal{H} \models A$ iff $\mathcal{H}^{\mathrm{s}} \models A^{\mathrm{s}}$.*

Without Proof.

In the structure $\mathcal{H}^{\mathrm{s}}$, the predicates T_τ and E_τ behave like type membership and elementhood, and it is not difficult to verify that $\mathcal{H}^{\mathrm{s}} \models \mathrm{TT}_0$ for all Henkin-prestructures $\mathcal{H}$. On the other hand, there are Henkin-prestructures which do not satisfy the comprehension axioms. This motivates the following definition.

Definition 3.19. A *Henkin-L-structure* is a Henkin-L-prestructure (D, I) that is closed under definability, i.e. for every term M of type τ we have $I(M) \in I(\tau)$.

We then obtain:

Lemma 3.3. *Let $\mathcal{S}$ be a L^{s}-structure. Then $\mathcal{S} \models \mathrm{TT}$ iff there is a Henkin-L-structure $\mathcal{H}$ s.t. $\mathcal{S} = \mathcal{H}^{\mathrm{s}}$*

Without Proof.

Definition 3.20. A simply typed formula is called *Henkin-valid* if it is true in all Henkin structures.

Theorem 3.7. *A relational formula is provable in* **NK**ω *iff it is Henkin-valid.*

Proof. **NK**ω proves A iff TT $\vdash A^s$ (by Lemma 3.1)
iff $\mathcal{S} \models A^s$ for all $\mathcal{S}$ with $\mathcal{S} \models$ TT (by first-order completeness)
iff $\mathcal{H}^s \models A^s$ for all Henkin-structures $\mathcal{H}$ (by Lemma 3.3)
iff $\mathcal{H} \models A$ for all Henkin-structures A (by Lemma 3.2)

□

The above theorem – the completeness of Henkin semantics – and its proof shows that Henkin-validity is a semantical notion much closer to first-order validity than to higher-order validity. The interest in Henkin-structures does therefore lie not so much in this completeness result itself but rather in the technical usefulness of Henkin structures for various purposes. For example, they allow to show of a valid formula that it is unprovable in **NK**ω by giving a Henkin-countermodel (see Theorem 5.5.3. in Leivant's article cited on the website for a proof of the unprovability of the (type-theoretic version of) the axiom of choice). As another application, Henkin-structures allow semantic cut-elimination proofs by showing Henkin-soundness of cut followed by Henkin-completeness of the cut-free calculus.

3.6 Type Theory at Work

Higher-order logic as we have seen it in the last few sections builds a popular logical basis for many *proof assistants* because of its flexibility and powerful type checking mechanisms. These proof assistants are interactive theorem proving environments that assist a user in fully formalizing a proof. Many of these systems are quite flexible with applications ranging from software verification to formalization of large mathematical proofs.

Among the most fully developed proof assistants are the systems

- Isabelle, http://www.cl.cam.ac.uk/research/hvg/isabelle/
- Coq, http://coq.inria.fr/
- Mizar, http://www.mizar.org/

The following is a proof script in Isabelle/HOL which is essentially a mechanization of higher-order logic as we have used in this chapter (with tons of additional features and syntactic sugar). For more information about Isabelle, consult the documentation that comes with the system.

```
theory demo

imports Main

begin

(*** propositional logic ***)

lemma lem1: "A \<and> B \<longrightarrow> A"
apply(rule impI)
apply(drule conjunct1)
apply(assumption)
done
```

```
lemma conj_comm: "A \<and> B \<Longrightarrow> B \<and> A"
apply(rule conjI)
apply(drule conjunct2)
apply(assumption)
apply(drule conjunct1)
apply(assumption)
done

lemma "A \<and> B \<longrightarrow> (B \<and> A) \<or> C"
apply(rule impI)
apply(rule disjI1)
apply(rule conj_comm)
apply(assumption)
done


(*** inductive data types ***)

datatype 'a mylist = nil | cons 'a "'a mylist"

primrec myapp :: "'a mylist \<Rightarrow> 'a mylist \<Rightarrow> 'a mylist" where
  "myapp nil ys = ys" |
  "myapp (cons x xs) ys = (cons x (myapp xs ys))"

value "nil"
value "(cons (3::nat) nil)"
value "(cons (2::nat) (cons (1::nat) nil))"
value "(myapp (cons (3::nat) nil) (cons (2::nat) (cons (1::nat) nil)))"

primrec mylen :: "'a mylist \<Rightarrow> nat" where
  "mylen nil = 0" |
  "mylen (cons x xs) = Suc (mylen xs)"

value "mylen nil"
value "mylen (cons (3::nat) nil)"
value "mylen (myapp (cons (3::nat) nil) (cons 17 (cons 1 nil)))"

lemma "mylen (myapp xs ys) = (mylen xs) + (mylen ys)"
apply(induct_tac xs)
apply(auto)
(* play with these commands
apply(simp del: myapp.simps)
apply(auto)
apply(simp_all)
apply(simp)
*)
done

lemma "(myapp xs (myapp ys zs)) = (myapp (myapp xs ys) zs)"
```

```
apply(induct_tac xs)
apply(auto)
done


(*** the Isar language: human-readable proof scripts,
                       closer to mathematical practice ***)

(* Cantor's theorem: there is no surjective function from a set to its powerset *)
theorem "\<exists> S. S \<notin> range (f :: 'a \<Rightarrow> 'a set)"
proof
  let ?S = "{x. x \<notin> f x}"
  show "?S \<notin> range f"
  proof
    assume "?S \<in> range f"
    then obtain y where "?S = f y" ..
    show False
    proof cases
      assume "y \<in> ?S"
      with `?S = f y` show False by auto
    next
      assume "y \<notin> ?S"
      with `?S = f y` show False by auto
    qed
  qed
qed

end (* end of theory *)
```

Exercise 3.6. *Use Isabelle/HOL to define the inductive datatype of trees (of some type* `'a`*) as we did for lists above. Define a function* `numleaves` *which computes the number of leaves of a given tree. Define a function* `insert` *which takes two trees T and S and replaces every leaf of T by the tree S. Prove that* `numleaves (insert T S) = (numleaves T) * (numleaves S)`*.*

Bibliography

[1] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof*. Academic Press, 1986.

[2] Hendrik Pieter Barendregt. *Lambda Calculi with Types*, volume 2 of *Handbook of Logic in Computer Science*, pages 117–309. Clarendon Press, 1992.

[3] Roger J. Hindley. *Basic Simple Type Theory*. Cambridge University Press, 1997.

[4] Daniel Leivant. Higher Order Logic. In Dov M. Gabbay, Christopher J. Hogger, J. A. Robinson, and Jörg H. Siekmann, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming (2)*, pages 229–322. Oxford University Press, 1994.

[5] Tobias Nipkow. Programming and Proving in Isabelle/HOL. available from http://www.cl.cam.ac.uk/research/hvg/Isabelle/documentation.html.

[6] Johan van Benthem and Kees Doets. Higher-order logic. In Dov Gabbay and Franz Guenther, editors, *Handbook of Philosophical Logic*, volume 1, pages 189–243. Kluwer, 2nd edition, 2001.